

LUCAS EGA ACADEMY

Robótica



Curso Completo de Robótica

Autor: Esteban

lucasegaacademy.com

*Guía completa con explicaciones extensas por unidad,
prácticas y examen final.*

CRÉDITOS

Curso Completo de Robótica

© 2026 Esteban Gutiérrez

Colibrí Editorial EGA

Hecho en México

Primera edición digital · septiembre de 2026

Sin ISBN

PRÓLOGO

Bienvenido a "Curso Completo de Robótica", el libro oficial de Lucas EGA Academy (lucasegaacademy.com) para aprender robótica desde cero. Si alguna vez viste un robot moverse solo y pensaste "yo quiero construir uno", este libro es para ti. No necesitas saber electrónica ni programación: empezamos desde lo más básico y terminamos con un robot completo que tú mismo vas a armar, programar y controlar desde tu celular.

Qué es este libro

Este libro es la versión escrita y extendida del curso de robótica de 30 clases de la academia. Cada unidad agrupa tres clases del curso y las desarrolla a fondo: con la teoría explicada con palabras sencillas, analogías de la vida diaria, ejemplos paso a paso con código real, prácticas con soluciones y un resumen para repasar. Al final encontrarás un examen de 50 preguntas para poner a prueba todo lo aprendido.

La robótica es una de las disciplinas más emocionantes que existen porque junta varias áreas: la mecánica (cómo se mueve el robot), la electrónica (cómo se conecta todo), la programación (cómo piensa el robot) y hasta un poco de física y matemáticas. No te asustes: cada tema se explica desde cero y con ejemplos que puedes tocar con tus manos.

El robot que vas a construir

A lo largo del libro no solo vas a leer: vas a construir. Empezarás encendiendo un LED, luego medirás distancias con un sensor ultrasónico, controlarás motores, moverás servomotores con precisión, armarás un robot que sigue una línea negra, otro que esquiva obstáculos solo, un brazo robótico que puedes mover a tu antojo y, como proyecto final, un robot que combina todo: se mueve solo cuando quiere y obedece tu celular por Bluetooth cuando tú se lo pides.

El "cerebro" de todos estos robots será Arduino, una plaquita pequeña, barata y muy famosa en todo el mundo. Arduino es ideal para aprender porque es sencillo de programar, hay miles de tutoriales y una comunidad enorme que comparte proyectos. Todo el código del libro está probado en Arduino Uno, la placa más común para empezar.

Materiales que necesitas

Para aprovechar el libro al máximo conviene tener un kit básico de robótica. No es obligatorio comprarlo todo el primer día: cada unidad indica exactamente qué materiales usa. La lista completa es la siguiente:

- Placa Arduino Uno (con su cable USB)
- Protoboard (tablero de conexiones sin soldadura)
- Cables de conexión macho-macho y macho-hembra
- LEDs de varios colores y resistencias de 220 ohmios
- Pulsadores (botones)
- Potenciómetros
- Sensor ultrasónico HC-SR04
- Sensores infrarrojos de reflectancia TCRT5000 (2 o 3)
- Fotorresistencia LDR
- Sensor de temperatura LM35 o DHT11
- Buzzer (zumbador)
- Módulo controlador de motores L298N
- 2 motores DC con ruedas (o un chasis de carrito robot)
- 3 o 4 servomotores SG90
- Módulo Bluetooth HC-05
- Baterías o portapilas (para alimentar los motores)
- Multímetro básico (muy recomendado)
- Cinta aislante, tornillería pequeña y una base o chasis

Con este kit puedes hacer absolutamente todos los proyectos del libro. Si tu presupuesto es ajustado, empieza con lo de las primeras unidades (Arduino, protoboard, LEDs, cables y botones) y ve agregando componentes conforme avances.

Seguridad: léeme antes de conectar nada

Trabajar con electrónica de bajo voltaje (5 voltios) es seguro si sigues unas reglas básicas. Léelas con atención:

- NUNCA conectes tus circuitos a la corriente eléctrica de tu casa (110/220V). Todo en este libro funciona con 5V del USB o con pilas/baterías pequeñas.
- Conecta y desconecta componentes con el circuito APAGADO. Acostúmbrate a quitar la alimentación antes de mover cables.
- Revisa la polaridad antes de energizar: el cable rojo va al positivo (+) y el negro a tierra (GND). Un componente al revés puede dañarse.
- No excedas los límites del Arduino: máximo 40 mA por pin y 200 mA en total. Los motores y varios servos SIEMPRE se alimentan con fuente externa, nunca directo del Arduino.
- Si algo se calienta, huele raro o saca humo, desconecta de inmediato y revisa tus conexiones. Un cortocircuito se evita revisando dos veces antes de encender.
- Los servomotores no se fuerzan con la mano cuando están energizados: puedes romper sus engranes.

Estas reglas se repiten en cada unidad donde aplican. La seguridad no es opcional: es parte de ser un buen robotista.

Cómo estudiar este libro

Cada unidad sigue la misma estructura para que siempre sepas qué esperar:

1. Introducción: qué vas a aprender y por qué importa.
2. Explicación detallada: la teoría con analogías, errores comunes y buenas prácticas.
3. Ejemplos paso a paso: proyectos guiados con código comentado línea por línea.
4. Práctica: ejercicios para hacer por tu cuenta, todos con solución explicada.
5. Resumen: lo esencial de la unidad en pocas palabras.

Mi recomendación: lee la explicación, ARMA los ejemplos con tus manos (la robótica se aprende haciendo, no solo leyendo), intenta los ejercicios antes de ver la solución y usa el resumen para repasar antes del examen final. Si algo no te sale, el monitor serie de Arduino será tu mejor amigo para descubrir qué está pasando: en la Unidad 2 aprenderás a usarlo.

No tengas prisa. Es mejor entender bien cada unidad que correr. Los robots premian la paciencia: un cable mal conectado te enseña más que diez páginas leídas por encima.

Un consejo de Esteban

Yo aprendí robótica a prueba y error, quemando algún LED en el camino (a todos nos pasa). Lo que me hubiera gustado tener es justo esto: una guía que explique el POR QUÉ de cada cosa, no solo el "conecta el cable rojo aquí". Por eso cada ejemplo de este libro explica qué hace cada línea de código y cada conexión. Cuando termines, no solo tendrás un robot funcionando: entenderás cómo funciona, y eso te permitirá inventar los tuyos.

Nos vemos en la Unidad 1. ¡A construir!

Esteban Lucas EGA Academy - lucasegaacademy.com

UNIDAD 1: FUNDAMENTOS DE LA ROBÓTICA

Bienvenido al "Curso Completo de Robótica" de Lucas EGA Academy. Si llegaste hasta aquí, lo mas probable es que alguna vez hayas visto un video de un robot que camina, de un dron que entrega paquetes o de un brazo mecánico que arma coches, y te hayas preguntado: "¿como le hace?", "¿que tiene adentro?", "¿yo podría construir uno?". La respuesta a esa última pregunta es si, y este libro existe exactamente para demostrarlo. En esta primera unidad no vas a soldar nada ni a programar nada. Vas a hacer algo todavía mas importante: vas a entender que es un robot de verdad, que tipos de robots existen en el mundo, y cuales son las piezas que forman a todos los robots del planeta, desde el juguete mas sencillo hasta los que exploran Marte.

Esta unidad cubre las clases 1 a 3 del curso, y cada una de ellas responde una pregunta grande. La clase 1 responde: ¿que es la robótica? Vas a conocer la definición de trabajo que vamos a usar en todo el libro: un robot es una máquina que percibe su entorno, piensa con base en lo que percibio y actúa sobre ese entorno. También vas a conocer un poco de historia, porque entender de donde vienen los robots te ayuda a entender hacia donde van. Verás automatas antiguos de hace siglos, la invención de la palabra "robot" en 1921 por el escritor checo Karel Capek, y el nacimiento de la robótica industrial moderna. Al final de la clase 1 sabrás donde hay robots hoy: en fabricas, en hospitales, en el campo y en tu propia casa, aunque a veces no los notes.

La clase 2 responde: ¿que tipos de robots existen? El mundo de los robots es enorme y conviene ordenarlo. Vas a conocer a los robots industriales, esos brazos gigantes que trabajan en las líneas de ensamblaje; a los robots móviles, que se desplazan con ruedas, orugas o patas; a los drones, que vuelan; a los humanoides, que intentan parecerse a nosotros; y a los robots de servicio y domésticos, como las aspiradoras robot que ya viven en miles de hogares. También, y esto es muy importante, vas a saber que tipo de robots vamos a construir en este libro: robots móviles basados en Arduino. No vamos a construir un robot industrial de millones de pesos, ni un humanoide. Vamos a construir máquinas pequeñas, baratas y muy didacticas, que se mueven, que "ven" obstáculos, que siguen líneas y que puedes controlar desde tu celular. Esa decisión es deliberada: el robot móvil con Arduino es la mejor escuela de robótica que existe, porque en un aparato pequeño aparecen todos los conceptos grandes.

La clase 3 responde: ¿de que esta hecho un robot? Aquí entra el concepto mas importante de toda la unidad, y probablemente de todo el libro: el ciclo percibir-procesar-actuar. Todo robot, sin excepción, funciona así: primero percibe el mundo con sus sensores (que son como sus sentidos), después procesa esa información con su controlador (que es como su cerebro), y finalmente actúa con sus actuadores (que son como sus músculos). Para que nada de esto se

quede flotando en el aire, vas a usar una analogía que te va a acompañar todo el curso: el robot es como un cuerpo humano. Los sensores son los sentidos, el controlador es el cerebro, los actuadores son los músculos y la batería es la comida, la energía que mantiene todo funcionando. También en la clase 3 vas a conocer al protagonista técnico del libro: Arduino. Vas a aprender que es una placa Arduino Uno y una Arduino Nano, que es el microcontrolador ATmega328P que llevan dentro, y que son los pines digitales y analógicos, que son como las terminales nerviosas por donde el cerebro se conecta con los sentidos y los músculos. Al final de la unidad tendrás la lista completa de materiales del curso, para que puedas conseguir todo con calma antes de empezar a armar.

¿por que importa todo esto? Porque la robótica no es un pasatiempo exótico ni un tema solo para ingenieros. Es una de las habilidades mas útiles que puedes aprender hoy. Los robots ya están en todas partes y cada año hay mas: en las fabricas que producen lo que compras, en los hospitales donde operan con precisión milimétrica, en los campos donde siembran y cosechan, en los almacenes donde preparan tus pedidos en línea. Entender como funcionan te convierte de espectador en creador. Y no necesitas ser genio de las matemáticas ni saber electrónica de antemano: este libro empieza desde cero absoluto, y esta unidad es el cimiento. Si entiendes bien el ciclo percibir-procesar-actuar, las clases prácticas que vienen después (conectar un sensor, mover un motor, programar un comportamiento) te van a parecer la consecuencia natural de lo que ya sabes, en lugar de una serie de pasos mágicos que hay que memorizar.

¿que vas a entender al terminar esta unidad? Vas a entender la definición formal de robótica y de robot, vas a poder nombrar y distinguir los principales tipos de robots con ejemplos reales, y vas a poder desarmar mentalmente cualquier robot en sus cuatro grandes bloques: sensores, controlador, actuadores y fuente de poder. Vas a saber que hace un Arduino, que puede y que no puede hacer, y por que lo elegimos para este curso. Vas a conocer el vocabulario básico que usaremos en todo el libro: sensor, actuador, microcontrolador, pin digital, pin analógico, voltaje, corriente, y el ciclo percibir-procesar-actuar. Ese vocabulario es tu caja de herramientas mental; cada unidad posterior va a construir sobre ella.

¿que vas a construir? En esta unidad todavía no construimos hardware, y eso es a propósito. Lo que vas a construir aquí es mas valioso para un principiante: un modelo mental correcto. Vas a hacer ejercicios de papel y lápiz donde "desarmas" robots imaginarios en sus componentes, donde dibujas el ciclo percibir-procesar-actuar aplicado a situaciones de la vida real (como cruzar la calle o como cocinar siguiendo una receta), y donde planificas que haría tu primer robot. También vas a armar tu lista de materiales y vas a aprender a leerla como un profesional: que es cada pieza, para que sirve en el ciclo, y en que clase la vamos a usar. Al final de la unidad tendrás un "mapa del tesoro": sabrás exactamente que piezas vas a necesitar, que hace cada una, y que vas a lograr con ellas en las unidades siguientes.

Una advertencia honesta antes de empezar: la robótica tiene fama de difícil porque mezcla tres mundos (mecánica, electrónica y programación), y muchos principiantes se asustan cuando ven los tres juntos. La estrategia de este libro es separarlos al principio y juntarlos poco a poco. En esta unidad solo hay ideas, nada de cables. En la unidad 2 empezarás con Arduino y tus primeros programas, encendiendo LEDs, que es el "hola mundo" de la robótica. En unidades posteriores conectarás sensores de verdad, moveras motores de verdad, y al final tendrás un robot móvil que evita obstáculos y sigue líneas. Cada paso se apoya en el anterior. Confía en el proceso, no te saltes esta unidad pensando que "es pura teoría": la teoría de aquí es la que hace que la práctica de después tenga sentido.

Un último consejo práctico: ten a la mano un cuaderno o un archivo de notas mientras lees. La robótica se aprende haciendo, pero también se aprende nombrando. Cada vez que aparezca un termino nuevo (y van a aparecer varios), escríbelo con tus propias palabras. Si puedes explicar con tus palabras que es un sensor, que es un actuador y que es el ciclo percibir-procesar-actuar, ya tienes el ochenta por ciento de esta unidad dominado. El otro veinte por ciento son los detalles, la historia y la lista de materiales, que también importan, pero que se aprenden repasando. Vamos a empezar.

Explicación detallada

Que es la robótica: la definición de trabajo

Empecemos por lo básico: ¿que es un robot? Hay muchas definiciones dando vueltas, algunas muy técnicas y otras muy de película. En este libro vamos a usar una definición de trabajo, sencilla y poderosa: un robot es una máquina que percibe su entorno, procesa información y actúa sobre ese entorno para lograr un objetivo. Tres verbos: percibir, procesar, actuar. Si una máquina hace las tres cosas, es un robot. Si solo hace una o dos, probablemente sea otra cosa: una herramienta, un electrodomestico o una computadora.

Vamos a probar la definición con ejemplos. Una licuadora: ¿percibe? No. ¿procesa información del entorno? No. ¿actúa? Si, licua, pero sin percibir ni decidir. Conclusión: la licuadora no es un robot, es una herramienta eléctrica. Un termostato moderno de esos que prenden la calefacción solos: ¿percibe? Si, mide la temperatura con un sensor. ¿procesa? Si, compara la temperatura medida con la que tu programaste. ¿actúa? Si, prende o apaga la calefacción. Conclusión: el termostato si es un robot, aunque sea un robot muy sencillo y aburrido. Una aspiradora robot: percibe (detecta paredes y escalones), procesa (decide por donde pasar) y actúa (mueve sus ruedas y succiona). Es un robot clarísimo. Un carro de control remoto: percibe poco o nada por si mismo (el que percibe eres tu con tus ojos), así que estrictamente es una máquina teledirigida, no un robot autónomo. Pero si a ese mismo carrito le pones un sensor que detecta paredes y un programa que gira solo cuando hay un obstáculo, se convierte en robot. ¿Ves como la definición funciona como una prueba? Mas adelante, cuando diseñes tus propios robots, esta prueba te va a servir para saber si lo que construiste es realmente un robot o solo un juguete con motor.

La robótica, entonces, es la disciplina que estudia, diseña, construye y programa robots. Y fíjate que es una disciplina mezclada: toma prestado de la mecánica (como se mueven las piezas), de la electrónica (como se conectan y alimentan los componentes), de la programación (como se le dan instrucciones al cerebro), y hasta de la inteligencia artificial y la psicología (como toma decisiones una máquina). Por eso a veces intimida: parece que hay que saber de todo. Pero la buena noticia es que no necesitas dominar las cuatro áreas para empezar; necesitas entender como se conectan entre si, y eso es exactamente lo que hace el ciclo percibir-procesar-actuar: es el puente que une la mecánica, la electrónica y el software en una sola idea.

Un error común de principiante es pensar que "robot" es sinonimo de "robot humanoide de película". No. La inmensa mayoría de los robots del mundo no se parecen a un humano: son brazos en fabricas, son discos que aspiran tu sala, son máquinas con ruedas que llevan cajas en un almacén. Otro error común es pensar que un robot necesita ser "inteligente" en el sentido de la inteligencia artificial avanzada. No: un robot puede ser muy tonto y seguir siendo robot. El seguidor de línea que vas a construir en este curso no "piensa" en ningún sentido profundo; simplemente lee sensores y reacciona. Y sin embargo percibe, procesa y actúa, así que es un robot legítimo. Guarda esta idea: en robótica, la autonomía (hacer cosas por si mismo siguiendo reglas) vale mas que la inteligencia espectacular.

Un poco de historia: de los automatatras a las fabricas

Los robots no nacieron ayer. La idea de construir máquinas que se muevan solas tiene miles de años. En la antigua Grecia ya había historias de estatuas que se movían, y en el siglo XVIII los relojeros europeos construían automatatras: muñecos mecánicos de una complejidad asombrosa, que escribían cartas, tocaban música o jugaban ajedrez, todo con engranes, resortes y levas, sin un solo cable eléctrico. Eran las computadoras mecánicas de su época, y demostraban algo que seguimos usando hoy: que el movimiento "inteligente" puede salir de mecanismos bien diseñados, no solo de cerebros.

La palabra "robot" como tal nació en 1921, en una obra de teatro del escritor checo Karel Capek titulada "R.U.R." (Rossum's Universal Robots). En checo, "robota" significa trabajo forzado o servidumbre, y en la obra los robots eran seres artificiales creados para trabajar. La palabra pegó tan fuerte que se quedó para siempre, y hoy la usamos en todos los idiomas. Es curioso: la palabra nació en la literatura antes de que existiera la tecnología, lo que muestra que los humanos llevamos mucho tiempo soñando con máquinas trabajadoras antes de poder construirlas.

La robótica moderna, la de verdad, nació en las fabricas. En la década de 1960 aparecieron los primeros brazos robóticos industriales, máquinas programables que podían soldar, pintar y mover piezas pesadas sin cansarse, sin distraerse y sin pedir aumento. La industria automotriz fue la gran adoptante: un brazo robótico puede soldar la carrocería de un coche con una precisión y una velocidad que ningún humano sostiene durante ocho horas seguidas. Con el tiempo los robots industriales se volvieron más precisos, más rápidos y más baratos, y se extendieron a la electrónica, a los alimentos, a la logística. Hoy hay fabricas donde cientos de robots trabajan coordinados, y el concepto de "fábrica inteligente" es una realidad cotidiana.

Paralelamente nació la robótica móvil: robots que se desplazan. Primero en laboratorios y universidades, después en aplicaciones reales: robots que exploran otros planetas (los rovers de Marte son robots móviles famosos), robots que desactivan explosivos, robots que reparten en hospitales, y finalmente robots domésticos como las aspiradoras robot, que llevaron la robótica móvil a millones de hogares. Y en los últimos años llegaron los drones, que son robots móviles que vuelan, usados en fotografía, agricultura, rescate y reparto. Cada una de estas ramas tiene su propia historia, pero todas comparten el mismo ciclo percibir-procesar-actuar.

¿por que te cuento todo esto? Por dos razones. Primera: para que veas que la robótica no es magia ni moda pasajera; es una tecnología con raíces profundas que lleva décadas resolviendo problemas reales. Segunda: para que ubiques donde estas tu en esa historia. Tu, con un Arduino de unos cuantos dólares y los componentes de la lista de materiales, vas a recorrer en unas semanas el mismo camino conceptual que la humanidad recorrió en décadas:

percibir con sensores, decidir con un controlador y actuar con motores. Los robots que vas a construir son primos lejanos y sencillos de los rovers de Marte, y usan exactamente los mismos principios.

Donde hay robots hoy

Mira a tu alrededor con ojos de robotista y vas a encontrar robots en todas partes. En las fabricas, como ya vimos, brazos robóticos sueldan, pintan, atornillan y empaacan. En la medicina, hay robots quirurgicos que permiten a los cirujanos operar con una precisión mayor que la de la mano humana, y hay robots que reparten medicamentos dentro de los hospitales. En la agricultura, hay tractores que se manejan solos con GPS, drones que vigilan los cultivos y robots que cosechan frutas. En el hogar, las aspiradoras robot y las podadoras robot ya son comunes, y los asistentes con bocina (que escuchan, procesan tu voz y actúan) son robots de un tipo particular. En la logística, los almacenes gigantes usan enjambres de robots móviles que llevan estantes completos hasta los trabajadores humanos. En el espacio, rovers y sondas exploran donde ningún humano puede llegar todavía.

Y hay una lección escondida en esta lista: fijate que casi ningún robot de la lista se parece a un humano. Los robots exitosos no imitan la forma humana; imitan la función necesaria. Un brazo industrial no necesita piernas; una aspiradora robot no necesita brazos. Esto es un principio de diseño que te va a servir: un buen robot es el mas sencillo que resuelve el problema. Cuando diseñes tus robots en este curso, resistiras la tentación de hacerlos complicados; los harás sencillos y funcionales, como los profesionales.

Tipos de robots

Ahora ordenemos el zoológico. Hay muchas formas de clasificar robots, pero la mas util para un principiante es por su forma de moverse y su propósito.

Los robots industriales son los veteranos. Normalmente son brazos articulados fijos a una base, con varios "grados de libertad" (formas en que se pueden mover: girar la base, doblar el codo, rotar la muñeca, etc.). Trabajan en fabricas haciendo tareas repetitivas con gran precisión: soldar, pintar, ensamblar, paletizar. Son caros, potentes y peligrosos si no se manejan bien (por eso suelen trabajar en jaulas de seguridad). Tu no vas a construir uno en este libro, pero es importante conocerlos porque son la rama mas grande de la robótica en términos economicos.

Los robots móviles son los que se desplazan por el entorno, y aquí hay tres subtipos según como se muevan. Los de ruedas son los mas comunes y los mas fáciles de construir: dos ruedas motrices mas una rueda loca de apoyo es la configuración clásica de los robots

didacticos (y es la que usaremos nosotros). Los de orugas usan bandas como los tanques: son mas lentos pero pasan sobre terrenos difíciles. Los de patas caminan como animales o humanos: son fascinantes pero mecánicamente complejos, así que los dejaremos para cuando tengas mas experiencia. Los robots móviles necesitan resolver un problema extra que los industriales no tienen: saber donde están y a donde van, lo que se llama navegación. Nuestro robot evitará obstáculos y seguirá líneas, que son las dos formas mas sencillas y didacticas de navegación.

Los drones son robots móviles que vuelan. Llevan hélices, sensores de altura y orientación, y un controlador que ajusta los motores cientos de veces por segundo para mantenerlos estables. Son espectaculares, pero construir uno desde cero como principiante es mala idea: si algo falla, se cae del cielo. Por eso en este libro nos quedamos en tierra, donde los errores cuestan una pila y no un dron estrellado.

Los humanoides son robots con forma humana: cabeza, tronco, dos brazos, dos piernas. Son los que salen en las noticias y en las películas, y la investigación en ellos es apasionante (caminar en dos piernas es un problema de control muy difícil). Pero son caros, frágiles y complejos; no son material de primer curso.

Los robots de servicio y domésticos son los que interactúan con personas en entornos humanos: aspiradoras robot, robots meseros, robots de compañía, robots que reparten en hospitales. Muchos de ellos son, en el fondo, robots móviles con sensores y comportamientos sencillos, igual que los que vamos a construir. De hecho, tu aspiradora robot y el robot que vas a armar en este curso son primos cercanos: ambos perciben obstáculos, deciden y se mueven.

Entonces, ¿que tipo construiremos en este libro? Robots móviles terrestres con ruedas, controlados por Arduino. Es la mejor decisión pedagogica por cuatro razones: son baratos (todo el kit cuesta poco), son seguros (nada vuela, nada pesa toneladas), contienen todos los conceptos (sensores, controlador, actuadores, programación, navegación básica) y son divertidos (ver moverse algo que tu construiste no tiene comparación). Todo lo que aprendas con ellos se transfiere después a drones, brazos o humanoides, porque el ciclo percibir-procesar-actuar es el mismo en todos.

El ciclo percibir-procesar-actuar: el corazón de todo

Llegó el momento de profundizar en la idea mas importante del libro. Todo robot funciona en un ciclo continuo de tres pasos. Vamos a verlo primero con la analogía del cuerpo humano, que te va a acompañar todo el curso.

Percibir es captar información del entorno. Tu lo haces con los sentidos: ojos, oídos, piel. Un robot lo hace con sensores: un sensor de distancia es como sus ojos (mide que tan lejos hay un obstáculo), un sensor de luz es como su visión del brillo, un micrófono es como sus oídos. Sin sensores, el robot está ciego y sordo: puede moverse, pero no sabe donde está ni que hay alrededor.

Procesar es decidir que hacer con la información percibida. Tu lo haces con el cerebro. El robot lo hace con su controlador, que en nuestro caso será el Arduino. El controlador lee los sensores, ejecuta el programa que tú escribiste, y decide: "hay un obstáculo a 20 centímetros, debo girar a la derecha". Sin controlador, el robot no decide nada; sería como un cuerpo sin cerebro.

Actuar es ejecutar la decisión en el mundo físico. Tú lo haces con los músculos. El robot lo hace con actuadores: motores que mueven ruedas, servomotores que giran brazos o sensores, bocinas que emiten sonidos, LEDs que se encienden. Sin actuadores, el robot podría pensar pero no hacer nada; sería un cerebro en un frasco.

Y hay un cuarto elemento que a veces se olvida: la energía. Tu cuerpo necesita comida; el robot necesita una fuente de poder, normalmente baterías. Sin energía no hay percepción, ni procesamiento, ni acción. En la práctica, elegir bien la alimentación (cuántas baterías, de que voltaje) es una de las decisiones más importantes y más subestimadas por los principiantes.

El ciclo es continuo: percibir, procesar, actuar, y otra vez percibir, procesar, actuar, cientos o miles de veces por segundo. Piensa en cómo manejas una bicicleta: tus ojos ven el camino (percibir), tu cerebro decide girar un poco (procesar), tus brazos giran el manubrio (actuar), y de inmediato tus ojos ven el resultado y el ciclo se repite. Un robot hace exactamente lo mismo, solo que más rápido y sin distraerse. A este ciclo cerrado también se le llama "lazo de control", y es la base de toda la robótica y de gran parte de la ingeniería.

Vamos a aplicarlo a un ejemplo concreto que vas a construir: el robot evita-obstáculos. Percibir: el sensor ultrasónico HC-SR04 mide la distancia al objeto de enfrente. Procesar: el Arduino compara la distancia con un umbral, por ejemplo 20 centímetros; si la distancia es menor, decide girar. Actuar: el Arduino le ordena al controlador de motores L298N que gire las ruedas para cambiar de dirección. Y el ciclo se repite: el sensor vuelve a medir, el Arduino vuelve a decidir, los motores vuelven a responder. Cuando entiendes este ejemplo, entiendes el patrón de todos los robots del libro: solo cambian los sensores, los actuadores y las reglas de decisión.

Errores comunes de principiantes con el ciclo: el primero es poner actuadores sin sensores, es decir, construir algo que se mueve "a ciegas" con movimientos pregrabados y esperar que funcione en el mundo real (no funciona: el mundo real siempre sorprende). El segundo es el

contrario: llenar el robot de sensores carísimos sin tener claro que decisión va a tomar el programa con esos datos (sensores sin propósito son adornos). El tercero es olvidar la energía: un robot con grandes motores y una pila pequeña se apaga a los dos minutos o se comporta de forma errática. Buena práctica: antes de armar nada, dibuja en papel tu ciclo: ¿que va a percibir?, ¿que va a decidir?, ¿como va a actuar?, ¿con que energía? Si puedes dibujarlo, puedes construirlo.

Arduino: el cerebro pequeño y barato

El controlador que usaremos en todo el libro es Arduino, y conviene entender bien que es y que no es. Arduino no es una empresa que vende una sola cosa; es una plataforma abierta de hardware y software para crear dispositivos electrónicos. La "placa Arduino" es una tarjeta con un microcontrolador (el chip que ejecuta tu programa), conectores para enchufar componentes sin soldar, y un puerto USB para programarla desde tu computadora. Lo revolucionario de Arduino es que volvió accesible algo que antes era difícil y caro: por pocos dólares tienes un cerebro programable listo para conectar sensores y motores.

La placa mas famosa es el Arduino Uno. Es del tamaño de una tarjeta de crédito, robusta, con muchos conectores, y es la que recomendamos para empezar porque es fácil de manipular y hay miles de tutoriales sobre ella. Su hermano menor es el Arduino Nano: hace lo mismo pero en tamaño miniatura, ideal cuando tu robot ya funciona y quieres hacerlo mas compacto. Ambas placas (en sus versiones clásicas) llevan el mismo microcontrolador: el ATmega328P. No necesitas memorizar ese nombre, pero si entender que significa: es el chip que ejecuta tu programa instrucción por instrucción, a una velocidad de 16 millones de instrucciones por segundo. Para que te des una idea, eso es lentísimo comparado con tu celular, y sin embargo es mas que suficiente para leer sensores, tomar decisiones y mover motores, porque esas tareas son sencillas para una computadora.

Analogía: si tu cerebro fuera una computadora de escritorio potente, el Arduino seria una calculadora programable de bolsillo. No le pidas que reconozca caras ni que juegue videojuegos; pidanle que lea un sensor, compare un número y prenda un motor, y lo hará perfecto y sin cansarse. El error clásico del principiante es sobreestimar o subestimar al Arduino: unos esperan que haga magia de inteligencia artificial (no puede), otros piensan que es "muy poquita cosa" para robótica seria (se equivocan: con el se han construido miles de robots reales y funcionales). La medida justa es esta: Arduino es el cerebro ideal para robots pequeños con comportamientos basados en reglas claras.

El programa que escribes para el Arduino sigue la analogía de la receta de cocina: es una lista de pasos ordenados que el cocinero (el microcontrolador) ejecuta al pie de la letra, sin improvisar. Si la receta dice "lee el sensor, si hay obstáculo gira, si no avanza", el Arduino hará

exactamente eso, ni mas ni menos. Por eso programar robots exige pensar como receta: pasos claros, condiciones claras ("si pasa X, haz Y"), y ningún paso ambiguo. En la unidad 2 aprenderás el lenguaje (una version simplificada de C++), pero desde ahora quédate con la idea: tu eres el chef que escribe la receta, el Arduino es el cocinero obediente que la ejecuta.

El programa como receta de cocina: piensa como programador desde hoy

Todavía no vas a escribir código, pero si vas a empezar a pensar como programador, porque la mitad de la robótica es pensar con claridad. La analogía de la receta de cocina merece desarrollarse, porque describe exactamente como "piensa" un Arduino.

Una receta tiene ingredientes (los datos de los sensores), pasos en orden (las instrucciones) y condiciones ("si la salsa esta muy espesa, agrega agua"). El Arduino ejecuta tu programa igual que un cocinero novato pero obediente: sigue los pasos en orden, evalúa las condiciones tal como están escritas, y nunca improvisa. Si tu receta dice "si hay obstáculo, gira a la derecha", el robot girara a la derecha aunque a la izquierda haya mas espacio libre. El no "sabe" que la izquierda era mejor; solo sabe lo que tu escribiste. Esta es la lección mas profunda de la programación de robots: la máquina solo es tan lista como las reglas que tu le das.

Por eso los buenos roboticistas escriben primero la receta en español, en papel, antes de tocar código. Se llama pseudocódigo, y es tu borrador. Ejemplo para el robot evita-obstáculos, escrito como receta:

Paso 1: mide la distancia con el sensor. Paso 2: si la distancia es menor a 20 centímetros, gira a la derecha durante medio segundo. Paso 3: si no, avanza recto. Paso 4: regresa al paso 1 y repite para siempre.

Fíjate en tres cosas. Primera: los pasos son concretos, sin ambigüedad ("20 centímetros", "medio segundo"). Segunda: hay una condición clara (si... / si no...). Tercera: el último paso dice "repite para siempre", que es el ciclo percibir-procesar-actuar convertido en instrucciones. Cuando en la unidad 2 veas este mismo ejemplo escrito en código Arduino, te vas a sorprender de lo parecido que es a esta receta. El código no es magia: es la receta traducida al idioma del cocinero.

El error típico aquí es escribir recetas ambiguas: "si ve algo, haz algo". ¿Que es "algo"? El Arduino no entiende "algo". Buena práctica: cada vez que escribas una regla, pregúntate si un niño de ocho años podría ejecutarla sin preguntarte nada. Si el niño tendría que preguntar, tu receta esta ambigua y el Arduino la ejecutará mal. Precisión en la receta, éxito en el robot.

Pines digitales y analógicos: las terminales nerviosas

¿cómo se conecta el cerebro con los sentidos y los músculos? A través de los pines: pequeñas terminales metálicas en la placa donde enchufas cables. Hay dos tipos fundamentales y entender la diferencia es clave.

Los pines digitales trabajan con dos valores: encendido o apagado, 1 o 0, HIGH o LOW. Son como interruptores. Sirven para leer sensores que responden si o no (¿hay un obstáculo o no?, ¿el botón está presionado o no?) y para controlar actuadores de dos estados (prende o apaga un LED, enciende o detiene un motor). El Arduino Uno tiene 14 pines digitales. Piensa en ellos como las respuestas de "sí o no" de tu cuerpo: ¿te quemaste? Sí o no. ¿la puerta está abierta? Sí o no.

Los pines analógicos leen valores continuos, no solo sí o no, sino "cuanto". El Arduino convierte el voltaje que recibe (entre 0 y 5 voltios) en un número entre 0 y 1023. Sirven para sensores que miden cantidades: ¿cuanta luz hay?, ¿qué tan cerca está el objeto?, ¿cuál es la temperatura? El Uno tiene 6 pines analógicos. Son como tus sentidos graduados: no solo sabes si hace calor o frío, sabes más o menos cuánto calor hace.

Esta distinción digital-analógico aparece en todo el libro, así que memorízala con un truco: digital es como un apagador (solo dos posiciones), analógico es como una perilla de volumen (muchas posiciones intermedias). Cuando elijas un sensor, una de tus primeras preguntas será: ¿me da un sí/no (digital) o me da un cuanto (analógico)?

Historias que enseñan: tres robots famosos

La teoría se fija mejor con historias reales. Aquí van tres, cada una con una lección.

El Pato de Vaucanson (1739). El inventor francés Jacques de Vaucanson construyó un pato mecánico que aleteaba, graznaba, comía grano y hasta lo "digería". Todo con mecanismos de relojería, sin electricidad. Lección: el comportamiento complejo puede salir de mecanismos simples bien diseñados; no todo necesita un microcontrolador. Cuando tu robot haga algo ingenioso con pocas piezas, estarás en la tradición de Vaucanson.

Unimate (1961). El primer brazo robótico industrial trabajó en una planta de General Motors, manipulando piezas de metal calientes y peligrosas. Era programable y repetía su tarea sin cansarse. Lección: los primeros robots triunfaron haciendo trabajos peligrosos y aburridos, no trabajos "inteligentes". Tu robot también debe resolver un problema concreto, aunque sea pequeño.

Sojourner (1997). El primer rover en Marte era del tamaño de un microondas, se movía lentísimo y tenía que decidir solo cómo rodear rocas porque las órdenes de la Tierra tardaban

minutos en llegar. Lección: la autonomía importa mas cuando la comunicación es lenta o imposible. Tu robot evita-obstáculos es autónomo en miniatura: decide solo, sin que tu le digas nada por control remoto.

Estas tres historias cubren tres siglos y un mismo mensaje: percibir, procesar y actuar, con los medios de cada época.

Sensores, actuadores y energía: el reparto del trabajo

Cerremos la explicación con el mapa completo de componentes, organizado por su papel en el ciclo.

Los sensores son los sentidos. En el curso usaremos varios: el HC-SR04, sensor ultrasónico que mide distancias (los "ojos" del robot evita-obstáculos); el TCRT5000, sensor infrarrojo que detecta si hay línea negra o superficie clara debajo (los "ojos" del robot seguidor de línea); el LDR, una resistencia que cambia con la luz (para robots que buscan o huyen de la luz). Cada sensor traduce una propiedad del mundo físico en una señal eléctrica que el Arduino puede leer. Buena práctica: un sensor debe responder a la pregunta "¿que necesita saber mi robot para decidir?". Si no responde a ninguna pregunta util, sobra.

Los actuadores son los músculos. Usaremos el motor DC (de corriente directa), que gira y mueve las ruedas; el driver L298N, que es el "intermediario de fuerza" entre el Arduino y los motores (el Arduino solo no tiene fuerza para mover un motor, igual que tu cerebro solo no levanta una pesa: necesita los músculos); el servomotor SG90, un motor pequeño que gira a posiciones exactas, util para mover un sensor de un lado a otro como si fuera un cuello; además de LEDs y zumbadores para señales visuales y sonoras. Cada actuador traduce una orden eléctrica del Arduino en una acción física.

La fuente de poder es la energía. Usaremos baterías o portapilas, y aprenderás algo fundamental: los motores consumen mucha corriente y deben alimentarse por separado del Arduino (a través del L298N), porque si conectas todo a la misma pila débil, el Arduino se reinicia o se comporta raro. Es como si tu cerebro se apagará cada vez que corres: mala idea. Buena práctica: alimenta la lógica (Arduino y sensores) y la fuerza (motores) por caminos separados pero con tierra común, como aprenderás en la unidad de armado.

Los componentes de soporte (protoboard, cables, resistencias) son el "tejido conectivo": la protoboard es la tabla donde enchufas todo sin soldar, los cables son los nervios que llevan señales, y las resistencias protegen componentes delicados como los LEDs. Sin ellos, nada se conecta con nada.

Y con esto tienes el cuadro completo: sensores que perciben, Arduino que procesa, actuadores que actúan, baterías que energizan, y una lista de materiales concreta que vamos a desglosar al final de la unidad. En la siguiente sección lo vas a practicar con papel y lápiz, desarmando robots imaginarios y diseñando comportamientos, para que cuando lleguen los componentes reales ya sepas exactamente que papel juega cada uno.

Ejemplos paso a paso

En esta sección no necesitas ningún componente. Solo papel, lápiz y tu cabeza. El objetivo es entrenar el músculo mas importante del roboticista: desarmar cualquier situación en el ciclo percibir-procesar-actuar. Haremos cuatro ejemplos completos, cada uno con pasos numerados en texto. Hazlos de verdad, escribiendo en tu cuaderno; leerlos sin hacerlos es como aprender a nadar viendo videos.

Ejemplo 1: desarma una aspiradora robot

Imagina la aspiradora robot redonda que limpia sola la sala. Vamos a desarmarla mentalmente en los cuatro bloques del ciclo. Toma tu cuaderno y sigue los pasos.

Paso 1: lista lo que la aspiradora percibe. Escribe: detecta paredes y muebles cuando los toca o los "ve" con sensores; detecta escalones para no caerse; detecta cuanta batería le queda; algunos modelos detectan que tan sucio esta el piso. Cada uno de esos es un sensor haciendo su trabajo de "sentido".

Paso 2: lista lo que procesa o decide. Escribe: decide por donde pasar para cubrir la sala (algunas van en zigzag, otras en espiral, otras mapean la casa); decide que hacer cuando choca con una pared (girar un angulo y seguir); decide cuando volver a su base a cargarse. Todo eso son reglas en su programa, su "receta de cocina".

Paso 3: lista como actúa. Escribe: mueve sus ruedas para desplazarse; gira su cepillo para barrer; enciende la succion para aspirar; emite sonidos o enciende luces para avisar. Esos son sus actuadores, sus "músculos".

Paso 4: identifica la energía. Escribe: batería recargable de litio, y su base de carga. Sin batería cargada, no hay percepción, ni decisión, ni acción.

Paso 5: verifica con la prueba de los tres verbos. ¿Percibe? Si. ¿Procesa? Si. ¿Actúa? Si. Conclusión: es un robot legítimo, aunque no se parezca a los de las películas.

¿que aprendiste? Que ya sabes analizar un robot real sin abrirlo. Este mismo análisis lo harás con cada robot que construyas: antes de conectar un cable, sabrás que percibe, que decide, como actúa y con que energía.

Ejemplo 2: tu cuerpo cruzando la calle

Este ejemplo usa la analogía del cuerpo humano para fijar el ciclo. Imagina que vas a cruzar una avenida.

Paso 1: percibir. Tus ojos miran a la izquierda y a la derecha buscando coches; tus oídos escuchan motores; tu piel siente si llueve. Anota cada sentido y que información aporta.

Paso 2: procesar. Tu cerebro combina todo: "viene un coche rápido por la izquierda, mejor espero; ya paso, ahora esta despejado, puedo cruzar". Fíjate que el cerebro no solo recibe datos: los combina, los compara con la experiencia y toma una decisión.

Paso 3: actuar. Tus músculos mueven tus piernas para caminar; si a mitad de la calle escuchas un claxon, tus músculos te hacen correr o detenerte. La acción cambia el mundo (tu posición) y eso genera nueva percepción.

Paso 4: repite el ciclo. Mientras cruzas, tus ojos siguen mirando, tu cerebro sigue evaluando, tus piernas siguen moviéndose. No es "miro una vez y cruzo a ciegas"; es un ciclo continuo hasta llegar a la otra banqueta.

Paso 5: traduce al robot. Ahora reescribe los cuatro pasos cambiando cuerpo por robot: sensores en vez de ojos, Arduino en vez de cerebro, motores en vez de músculos, batería en vez de comida. Te queda el diseño conceptual de un robot que cruza calles. Eso es exactamente lo que harás en el curso, solo que con obstáculos de cartón en tu mesa en vez de avenidas.

La moraleja: ya conoces el ciclo percibir-procesar-actuar porque lo ejecutas miles de veces al día. La robótica no te pide aprender algo ajeno; te pide imitar en una máquina algo que tu cuerpo ya hace.

Ejemplo 3: diseña en papel un robot que riega plantas

Vamos a diseñar un robot imaginario, solo con papel y lápiz, para practicar el diseño antes del armado. El problema: tienes plantas en macetas y a veces se te olvida regarlas.

Paso 1: define el objetivo en una frase. Escribe: "el robot debe detectar cuando la tierra esta seca y regar la planta automáticamente". Un objetivo claro evita robots que hacen de todo y nada bien.

Paso 2: decide que debe percibir. Para saber si la tierra esta seca necesita un sensor de humedad de suelo (existe y es barato). Escribe: "sensor de humedad: percibe que tan húmeda esta la tierra (valor analógico: cuanta humedad)".

Paso 3: decide que debe procesar. Escribe la regla como receta: "lee el sensor cada hora; si la humedad esta por debajo del umbral, activa el riego durante 10 segundos; si no, no hagas nada". Fíjate que es una regla simple de si/no basada en un número.

Paso 4: decide como debe actuar. Necesita un actuador que mueva agua: una mini bomba de agua eléctrica. Escribe: "bomba de agua: actuador que riega cuando el Arduino la enciende".

Paso 5: define la energía. Escribe: "eliminador de corriente o batería, con alimentación separada para la bomba (consume mas corriente que el Arduino)".

Paso 6: dibuja el ciclo con flechas de texto. Escribe algo así: "sensor de humedad -> Arduino (compara con umbral) -> bomba (riega o no) -> la tierra se humedece -> el sensor lo nota -> ciclo". Si tu dibujo cierra el ciclo, tu diseño esta completo.

Paso 7: haz la prueba de los tres verbos y busca errores de principiante. ¿Percibe? Si. ¿Procesa? Si. ¿Actúa? Si. ¿Hay sensores sin propósito? No. ¿Hay actuadores sin sensor que los controle? No. ¿La energía alcanza para la bomba? Lo anotaste por separado: bien.

Este ejercicio es el ensayo general de todo el curso: cada proyecto empezara así, en papel, antes de tocar un cable.

Ejemplo 4: clasifica robots como un profesional

Vamos a practicar la clasificación de la clase 2 con robots reales. Dibuja en tu cuaderno una tabla de cuatro columnas: "robot", "tipo", "como se mueve", "ciclo percibir-procesar-actuar".

Paso 1: el brazo que suelda coches en una fábrica. Tipo: robot industrial. Como se mueve: fijo a una base, articulado. Ciclo: percibe la posición de la pieza con sensores, procesa la trayectoria programada, actúa moviendo sus articulaciones para soldar.

Paso 2: el rover que explora Marte. Tipo: robot móvil de ruedas (seis ruedas). Ciclo: percibe el terreno con cámaras y sensores, procesa que ruta es segura (con ayuda de la Tierra a veces), actúa moviendo sus ruedas y su brazo.

Paso 3: el dron que toma fotos de una boda. Tipo: dron (robot móvil aéreo). Ciclo: percibe su inclinación y altura con sensores, procesa cientos de ajustes por segundo para mantenerse estable, actúa variando la velocidad de sus hélices.

Paso 4: la aspiradora robot de tu casa. Tipo: robot de servicio/domestico (y también móvil de ruedas). Ciclo: ya lo desarmaste en el ejemplo 1.

Paso 5: un robot humanoide que juega futbol en una competencia universitaria. Tipo: humanoide. Ciclo: percibe la pelota y a los rivales con cámara, procesa a donde patear, actúa moviendo piernas y cuerpo.

Paso 6: el robot que TU vas a construir en este curso. Tipo: robot móvil de ruedas con Arduino. Ciclo: lo definirás tu en los ejercicios de práctica, y lo construirás en las unidades siguientes.

Al terminar tu tabla tendrás seis robots clasificados y analizados. Guarda esa hoja: cuando alguien te pregunte "¿que tipos de robots hay?", la podrás explicar con ejemplos, no con definiciones memorizadas. Y fíjate en el patrón: todos, del brazo industrial al tuyo, usan el mismo ciclo. Esa es la belleza de la robótica: principios universales, aplicaciones infinitas.

Práctica

Resuelve cada ejercicio en tu cuaderno antes de leer la solución. La gracia esta en intentarlo: equivocarse aquí es gratis, equivocarse con componentes conectados cuesta dinero y tiempo.

Ejercicio 1: la prueba de los tres verbos

Enunciado: de la siguiente lista, indica cuales son robots según la definición de este libro (percibir, procesar y actuar) y cuales no, justificando cada caso: a) un ventilador con control remoto, b) un semaforo peatonal con botón, c) una puerta automática de supermercado, d) un reloj despertador, e) un carro de juguete que avanza hasta chocar con la pared y entonces gira solo.

Objetivo: aplicar la definición de robot como prueba práctica y distinguir robots de simples máquinas automáticas.

Solución del ejercicio 1

Vamos caso por caso con los tres verbos.

a) Ventilador con control remoto: ¿percibe el entorno? No; solo obedece tu control. ¿Procesa información del entorno? No. ¿Actúa? Si, mueve aire. Le falta percibir y procesar por si mismo: no es un robot, es una herramienta teledirigida.

b) Semaforo peatonal con botón: percibe (el botón presionado es una señal de entrada), procesa (su controlador cambia la secuencia de luces según una lógica), actúa (enciende las

luces). Cumple los tres verbos de forma sencilla: si es un robot, aunque sea uno muy simple y fijo.

c) Puerta automática de supermercado: percibe (sensor de presencia), procesa (decide abrir cuando detecta a alguien), actúa (el motor abre la puerta). Es un robot.

d) Reloj despertador: ¿percibe el entorno? No percibe nada del mundo exterior; solo cuenta su propio tiempo interno. No es un robot, es un temporizador con bocina.

e) Carrito que choca y gira solo: percibe (detecta el choque con un sensor o interruptor), procesa (decide girar), actúa (mueve sus ruedas). Es un robot móvil autentico, y de hecho es el ancestro conceptual del robot evita-obstáculos que vas a construir.

La lección: la definición no depende de que tan impresionante se vea la máquina, sino de que cumpla el ciclo. Un semaforo con botón es mas "robot" que un ventilador caro con control remoto.

Ejercicio 2: clasifica y justifica

Enunciado: clasifica cada robot en uno de los tipos vistos en la clase 2 (industrial, móvil de ruedas, móvil de orugas, móvil de patas, dron, humanoide, servicio/domestico) y escribe su ciclo percibir-procesar-actuar en una línea por cada paso: a) robot que lleva medicamentos dentro de un hospital, b) brazo que pinta coches en una fábrica, c) cuadricoptero que revisa líneas eléctricas, d) robot con dos piernas que reparte volantes en una plaza, e) podadora que corta el pasto sola.

Objetivo: dominar la clasificación de robots y practicar el análisis por ciclo en casos variados.

Solución del ejercicio 2

a) Robot de hospital: tipo servicio, móvil de ruedas. Percibe: pasillos, personas y obstáculos con sensores. Procesa: calcula la ruta al cuarto destino. Actúa: se desplaza con sus ruedas llevando los medicamentos.

b) Brazo que pinta: tipo industrial. Percibe: posición de la carrocería (sensores o posiciones programadas). Procesa: sigue la trayectoria de pintado. Actúa: mueve sus articulaciones y activa la pistola de pintura.

c) Cuadricoptero de líneas eléctricas: tipo dron. Percibe: inclinación, altura y posición con sensores. Procesa: se estabiliza y sigue la línea. Actúa: ajusta sus cuatro hélices y mueve su cámara.

d) Robot de dos piernas con volantes: tipo humanoide (y de servicio). Percibe: personas y obstáculos con cámara y sensores. Procesa: decide a quien acercarse y mantiene el equilibrio. Actúa: camina con sus piernas y ofrece volantes con sus brazos.

e) Podadora sola: tipo servicio/domestico, móvil de ruedas. Percibe: limites del jardín (cable perimetral o sensores) y obstáculos. Procesa: decide el patrón de corte. Actúa: se mueve y gira sus cuchillas.

Observa que varios robots pertenecen a dos categorías a la vez (servicio y móvil): las clasificaciones se combinan, no son cajas excluyentes.

Ejercicio 3: diseña el ciclo de un robot seguidor de luz

Enunciado: diseña en papel un robot móvil de ruedas que siempre avanza hacia la fuente de luz mas intensa (como una planta que busca el sol). Escribe: objetivo en una frase, que percibe (que sensor usaría y que mide), que procesa (la regla de decisión como receta), como actúa (que motores y que movimientos) y con que energía. Usa los componentes de la lista del curso donde sea posible.

Objetivo: practicar el diseño completo de un robot usando el ciclo, conectando sensores, decisiones y actuadores reales del curso.

Solución del ejercicio 3

Objetivo: "el robot gira y avanza siempre hacia el punto con mas luz".

Percibir: dos sensores LDR, uno a la izquierda y uno a la derecha del frente del robot. Cada LDR mide cuanta luz recibe de su lado (valor analógico: mas luz, número distinto según como se conecte; lo importante es que da un "cuanto").

Procesar (receta): "lee el LDR izquierdo y el derecho; si el izquierdo recibe mas luz, gira a la izquierda; si el derecho recibe mas luz, gira a la derecha; si reciben casi lo mismo, avanza recto". Es una comparación simple repetida en ciclo.

Actuar: dos motores DC, uno por rueda, controlados por el L298N. Para girar a la izquierda, el motor derecho avanza mas rápido que el izquierdo; para avanzar recto, ambos igual. Así el robot "nada" hacia la luz.

Energía: baterías para los motores (por el L298N) y alimentación para el Arduino y los LDR, con tierra común.

Verificación: ¿percibe? Si, luz por dos lados. ¿Procesa? Si, compara y decide. ¿Actúa? Si, mueve ruedas. ¿Sensores con propósito? Si, ambos alimentan la decisión. Este robot es real y construible con el kit del curso; de hecho, es primo del seguidor de línea que harás después.

Ejercicio 4: digital o analógico

Enunciado: para cada sensor o señal, indica si el Arduino lo leería por un pin digital (si/no) o analógico (cuanto), y explica por que: a) botón que el usuario presiona, b) LDR que mide la luz del cuarto, c) sensor que detecta si hay línea negra bajo el robot o no, d) sensor ultrasónico HC-SR04 que mide distancia en centímetros, e) interruptor de choque que se activa al golpear una pared.

Objetivo: fijar la diferencia entre señal digital y analógica, una distinción que usarás en todo el curso.

Solución del ejercicio 4

a) Botón: digital. Solo hay dos estados: presionado o no presionado. Es un si/no perfecto.

b) LDR: analógico. La luz del cuarto no es solo "hay o no hay"; hay niveles: penumbra, normal, muy brillante. Necesitas el "cuanto", así que va a un pin analógico que te da un número entre 0 y 1023.

c) Detector de línea negra: depende del diseño, y esto es instructivo. Si el sensor esta ajustado para responder "veo negro / veo blanco", es digital. Si te entrega que tan oscuro es, es analógico. En el curso lo usaremos en modo digital (si/no), que es mas sencillo para seguir líneas. Moraleja: a veces tu decides como leer un sensor según lo que necesites.

d) HC-SR04: el resultado que te interesa es la distancia en centímetros, un "cuanto" continuo. En la práctica este sensor se lee con pines digitales usando medicion de tiempo (el Arduino mide cuanto tarda el eco), pero conceptualmente entrega una cantidad, no un si/no. Lo importante aquí: lo que define digital o analógico es la naturaleza de la información (cantidad vs. si/no), no solo el pin físico.

e) Interruptor de choque: digital. O choco o no choco; no hay "choco un poco".

Truco para no fallar: pregúntate "¿la respuesta es si/no o es cuanto?". Si/no va por digital, cuanto va por analógico.

Ejercicio 5: encuentra el error de diseño

Enunciado: un principiante diseña este robot: "mi robot lleva 5 sensores ultrasónicos, 3 sensores de luz, cámara, 4 motores, 2 servos, bocina y 20 LEDs, todo conectado a un Arduino Uno alimentado por una pila cuadrada de 9 voltios". Señala al menos cuatro errores de diseño según lo aprendido en la unidad y explica como corregiría cada uno.

Objetivo: aplicar buenas prácticas y detectar los errores típicos de principiante antes de cometerlos con hardware real.

Solución del ejercicio 5

Error 1: demasiados sensores sin propósito claro. El diseño no dice que decisiones tomará el robot con 5 ultrasónicos y 3 LDR. Corrección: empezar con un objetivo y un ciclo dibujado; agregar sensores solo cuando respondan a una pregunta que el programa necesita.

Error 2: el Arduino Uno no tiene pines suficientes para todo eso. Entre ultrasónicos (2 pines cada uno), servos, motores y 20 LEDs, se acaban los pines. Corrección: diseñar con la cuenta de pines en mente, o simplificar el robot.

Error 3: una pila de 9 voltios no puede con 4 motores. Los motores consumen mucha corriente; esa pila se agotaría en minutos y el voltaje caería, reiniciando el Arduino. Corrección: baterías adecuadas para los motores (portapilas con varias AA o batería recargable) y alimentación separada para lógica y fuerza, como manda la buena práctica.

Error 4: complejidad innecesaria. Un primer robot debe ser el mas sencillo que cumpla su objetivo. Corrección: version 1 con 1 sensor, 2 motores y 1 comportamiento; las versiones 2 y 3 agregan lo demás cuando la version 1 funcione.

Error extra (punto bonus): 20 LEDs no aportan nada al ciclo percibir-procesar-actuar; son decoración. Nada contra decorar tu robot, pero primero que funcione, después que brille.

Ejercicio 6: tu primer diseño completo

Enunciado: diseña en papel tu propio robot móvil con Arduino usando solo componentes de la lista del curso. Debe tener: nombre del robot, objetivo en una frase, diagrama del ciclo percibir-procesar-actuar (que sensor, que decisión, que actuador, que energía), lista de componentes que usarías, y la regla de decisión escrita como receta de cocina con al menos una condición "si... entonces...".

Objetivo: integrar todo lo aprendido en la unidad en un diseño propio y realizable, listo para construir en unidades futuras.

Solución del ejercicio 6

No hay una única respuesta correcta; aquí va un ejemplo modelo para que compares con el tuyo.

Nombre: "Explorador de mesa".

Objetivo: "recorrer la mesa sin caerse de los bordes".

Percibir: dos sensores TCRT5000 apuntando hacia abajo, uno al frente-izquierda y uno al frente-derecha; detectan si hay superficie (mesa) o vacío (borde).

Procesar (receta): "lee ambos sensores en ciclo; si el izquierdo no ve superficie, gira a la derecha; si el derecho no ve superficie, gira a la izquierda; si ambos ven superficie, avanza recto; repite siempre".

Actuar: dos motores DC con ruedas, manejados por el L298N; giros reduciendo la velocidad de una rueda.

Energía: portapilas para los motores vía L298N y alimentación al Arduino, con tierra común.

Componentes: Arduino Uno, protoboard, cables, 2 TCRT5000, L298N, 2 motores DC con ruedas, chasis, portapilas y baterías.

Verificación: percibe (superficie), procesa (regla del borde), actúa (gira o avanza). Es sencillo, usa pocos pines, y resuelve un problema real (no caerse). Si tu diseño cumple el ciclo, cabe en los pines del Uno y la energía esta separada para motores, esta bien diseñado aunque sea distinto a este ejemplo. Guárdalo: en las siguientes unidades lo vas a construir de verdad.

Ejercicio 7: el robot de tu casa

Enunciado: elige un aparato de tu casa (lavadora, microondas, television, aire acondicionado, Alexa o bocina inteligente, aspiradora normal) y aplicale la prueba de los tres verbos. Escribe que percibe (si es que percibe algo), que procesa y como actúa. Concluye si es un robot o no, y que le faltaria para serlo.

Objetivo: entrenar el ojo roboticista en la vida cotidiana y reforzar la definición con objetos cercanos.

Solución del ejercicio 7

Ejemplo con lavadora moderna: percibe (nivel de agua, peso aproximado de la ropa en algunos modelos), procesa (elige duración y ciclos según el programa), actúa (mueve el tambor, abre válvulas de agua). Es un robot domestico sencillo, aunque nadie la llama así. Ejemplo con microondas básico: no percibe nada del alimento (solo cuenta el tiempo que tu le pones), así que no es un robot: es un temporizador con un generador de calor. Ejemplo con bocina inteligente: percibe (tu voz con micrófonos), procesa (entiende el comando, a veces con ayuda de internet), actúa (responde con sonido, prende focos conectados). Es un robot de un tipo especial, sin ruedas ni brazos. La conclusión general: hay mas robots en tu casa de los que creías, y ahora sabes detectarlos.

Ejercicio 8: calcula los pines

Enunciado: tu robot usa un Arduino Uno (14 pines digitales, 6 analógicos). Quieres conectar: 1 sensor HC-SR04 (usa 2 pines digitales), 2 sensores TCRT5000 en modo digital (1 pin digital cada uno), 2 LDR (1 pin analógico cada uno), 1 driver L298N para 2 motores (4 pines digitales), 1 servomotor SG90 (1 pin digital) y 2 LEDs (1 pin digital cada uno). ¿Alcanzan los pines? Muestra la cuenta. ¿Que harías si quisieras agregar un tercer TCRT5000?

Objetivo: practicar la planificacion de pines, una habilidad práctica que evita rediseños a mitad del armado.

Solución del ejercicio 8

Cuenta de pines digitales: $HC-SR04 \ 2 + TCRT5000 \ 2 + L298N \ 4 + servo \ 1 + LEDs \ 2 = 11$ pines digitales. El Uno tiene 14: alcanzan y sobran 3. Pines analógicos: $2 \ LDR = 2$ de 6 disponibles: sobran 4. Conclusión: el diseño cabe sin problema. Si quisieras agregar un tercer TCRT5000, usarías 1 pin digital mas (total 12 de 14): todavía alcanza. Esta cuenta, hecha en papel antes de comprar o conectar nada, es lo que separa al aficionado improvisado del roboticista que termina sus proyectos. Acostúmbrate a hacerla siempre.

Lista de materiales del curso

Estos son los componentes que usaremos a lo largo del libro. Consiguelos con calma antes de la unidad 2; no necesitas todo el primer día, pero tener el kit completo evita pausas a mitad de los proyectos.

LISTA DE MATERIALES - CURSO COMPLETO DE ROBOTICA

Lucas EGA Academy (lucasegaacademy.com)

CONTROLADOR

- 1x Arduino Uno (placa principal del curso)

CONEXIONES

- 1x Protoboard de 400 puntos
- Cables jumper macho-macho (paquete de 20 o mas)
- Cables jumper macho-hembra (paquete de 10 o mas)

SALIDAS BASICAS

- 5x LEDs (rojo, verde, amarillo, los colores que haya)
- 5x Resistencias de 220 ohmios (para los LEDs)
- 1x Buzzer o zumbador pasivo (para sonidos)

SENSORES

- 1x HC-SR04 (sensor ultrasonico de distancia)
- 2x TCRT5000 (sensores infrarrojos seguidor de linea)
- 2x LDR (fotoresistencias, sensores de luz)
- 2x Resistencias de 10k ohmios (para los LDR)

MOVIMIENTO

- 1x Driver L298N (controlador de motores)
- 2x Motores DC con caja reductora y ruedas
- 1x Rueda loca (rueda libre de apoyo)
- 1x Servomotor SG90 (movimientos de precision)

COMUNICACION

- 1x Modulo Bluetooth HC-05 (control desde el celular)

ENERGIA

- 1x Portapilas para 4 pilas AA (para los motores)
- 4x Pilas AA recargables (o alcalinas)
- 1x Pila de 9V con conector (para el Arduino, opcional)
- 1x Cable USB tipo A-B (para programar el Arduino)

ESTRUCTURA

- 1x Chasis para robot movil (o base de acrilico/madera)
- Tornillos, separadores y tuercas surtidas

Nota: cada componente de esta lista aparecera en su momento con su explicación y su conexión paso a paso. Si algo no lo encuentras con el nombre exacto, busca su equivalente: lo importante es la función (sensor de distancia, driver de motores, etc.), no la marca.

Glosario rápido de la unidad

- Robot: máquina que percibe, procesa y actúa para lograr un objetivo.
- Robótica: disciplina que estudia, diseña, construye y programa robots.
- Ciclo percibir-procesar-actuar: el lazo de control que todo robot repite continuamente; también llamado lazo de control.
- Sensor: componente que percibe una propiedad del entorno y la convierte en señal eléctrica; es el "sentido" del robot.
- Actuador: componente que convierte una orden eléctrica en acción física; es el "músculo" del robot.
- Controlador: dispositivo que lee sensores, ejecuta el programa y comanda actuadores; es el "cerebro" del robot.
- Arduino: plataforma abierta de hardware y software; su placa lleva un microcontrolador listo para conectar sensores y actuadores.
- Microcontrolador: chip que ejecuta un programa instrucción por instrucción; el de nuestro Arduino es el ATmega328P.
- Pin digital: terminal que maneja dos valores (encendido/apagado); como un apagador.
- Pin analógico: terminal que lee valores continuos convertidos a un número entre 0 y 1023; como una perilla de volumen.
- Driver (L298N): intermediario de fuerza entre el Arduino y los motores de corriente directa.
- Servomotor (SG90): motor pequeño que se mueve a posiciones exactas.
- Autonomía: capacidad del robot de decidir y actuar por si mismo siguiendo sus reglas, sin control remoto.
- Pseudocódigo: receta escrita en español con los pasos del programa, antes de traducirla a código.

Resumen de la unidad

En esta unidad construiste el cimiento de todo el curso. Aprendiste la definición de trabajo: un robot es una máquina que percibe su entorno, procesa información y actúa sobre ese entorno para lograr un objetivo. Esa definición es también una prueba: si una máquina cumple los tres verbos, es un robot; si no, es otra cosa. Viste que la robótica nace de la unión de la mecánica, la electrónica y la programación, y que el puente que las une es el ciclo percibir-procesar-actuar, el famoso lazo de control que se repite cientos de veces por segundo en todo robot.

Recorriste la historia: de los automatismos mecánicos del siglo XVIII a la palabra "robot" inventada por Karel Capek en 1921, hasta los brazos industriales de los años sesenta y los robots móviles, drones y aspiradoras robot de hoy. Ubicaste a los robots en el mundo actual: fábricas, hospitales, campos, almacenes, hogares y hasta Marte. Y ordenaste el zoológico con la clasificación por tipos: industriales, móviles (ruedas, orugas, patas), drones, humanoides y de servicio o domésticos. Decidimos juntos que en este libro construiremos robots móviles de ruedas con Arduino: baratos, seguros, didácticos y divertidos.

Desarmaste el robot en sus cuatro bloques usando la analogía del cuerpo humano: los sensores son los sentidos (HC-SR04, TCRT5000, LDR), el controlador es el cerebro (Arduino Uno o Nano con el microcontrolador ATmega328P), los actuadores son los músculos (motores DC con el driver L298N, servomotor SG90, LEDs, buzzer) y la fuente de poder es la energía (baterías, con alimentación separada para lógica y fuerza). Aprendiste la diferencia entre pines digitales (sí/no, como un apagador) y pines analógicos (cuanto, como una perilla de volumen), y la analogía del programa como receta de cocina que el Arduino ejecuta al pie de la letra.

También entrenaste como roboticista: desarmaste una aspiradora robot en su ciclo, cruzaste una calle con tu cuerpo para sentir el lazo de control, diseñaste en papel un robot que riega plantas y clasificaste seis robots reales. En la práctica aplicaste la prueba de los tres verbos, distinguiste señales digitales de analógicas, detectaste los errores típicos del principiante (sensores sin propósito, pines insuficientes, energía débil, complejidad innecesaria) y diseñaste tu primer robot completo en papel.

Llévate estas ideas grabadas: todo robot percibe, procesa y actúa en ciclo continuo; el mejor robot es el más sencillo que resuelve el problema; diseña en papel antes de conectar un cable; y separa la alimentación de motores y lógica. En la unidad 2 encenderás tu primer LED con Arduino y escribirás tu primera receta de cocina en código. El cimiento está puesto: ahora a construir.

UNIDAD 2: PRIMEROS PASOS CON ARDUINO

Introducción

Bienvenido a la Unidad 2 de tu Curso Completo de Robótica. En la unidad anterior aprendiste que es un robot, cuales son sus partes y que hace que una máquina se convierta en un verdadero robot: la capacidad de sentir su entorno, pensar sobre lo que siente y actuar en consecuencia. Hoy vas a dar el paso mas emocionante de todo el curso: vas a ponerle un cerebro a tu primer robot, o mejor dicho, vas a aprender a programar el cerebro que usaremos durante todo el curso.

Ese cerebro se llama Arduino, y es la razón por la que miles de personas en todo el mundo, desde estudiantes de secundaria hasta ingenieros profesionales, han podido construir robots, drones, impresoras 3D, sistemas de riego automático, alarmas, juguetes inteligentes y cientos de inventos mas. Arduino es una pequeña tarjeta electrónica que puedes programar para que haga lo que tu quieras: prender luces, leer botones y sensores, mover motores y comunicarse con tu computadora. Es barata, es fácil de conseguir y, lo mas importante, fue diseñada desde el principio para que personas sin experiencia previa en electrónica ni en programación pudieran usarla. No necesitas ser un genio de las matemáticas ni haber estudiado ingeniería. Solo necesitas curiosidad, tu computadora y las ganas de experimentar.

Pero antes de conectar cables y escribir código, vale la pena entender por que Arduino se convirtió en el estándar del mundo para aprender robótica. Hace muchos años, trabajar con microcontroladores era cosa de expertos: había que comprar programadores caros, leer hojas de datos de cientos de páginas en inglés técnico y lidiar con circuitos delicados donde un error podía quemar el chip. En el año 2005, un grupo de estudiantes y profesores en Italia decidió cambiar eso. Crearon una placa sencilla, con un conector USB para programarla como si fuera una memoria, y un programa gratuito donde escribes instrucciones en un lenguaje fácil de entender. Esa placa fue el Arduino, y su éxito fue tan grande que hoy existen clones y versiones compatibles en todos los países, incluyendo México y toda Latinoamérica, a precios que cualquier estudiante puede pagar.

En esta unidad vas a dominar lo esencial: instalar el programa para programar tu Arduino (se llama IDE de Arduino), escribir tu primer programa (que en el mundo Arduino se llama sketch), subirlo a la placa y ver como un LED parpadea con las instrucciones que tu mismo escribiste. Ese momento, cuando la lucecita parpadea por primera vez al ritmo que tu decidiste, es un momento que ningún aficionado a la robótica olvida. Es la prueba de que tu puedes darle órdenes a una máquina y la máquina te obedece. De ahí en adelante, todo lo que construyas será una variación de esa misma idea: tu escribes las reglas, el robot las ejecuta.

Además del primer parpadeo, vas a aprender a controlar varios LEDs para crear secuencias como un semáforo de verdad, vas a conectar botones para que tu programa reaccione cuando alguien los presione, y vas a usar el monitor serie, que es como una ventana de chat entre tu Arduino y tu computadora, perfecta para ver que esta pasando dentro del cerebro de tu robot mientras trabaja. Cada una de estas habilidades es un ladrillo, y al final de la unidad tendrás los cimientos completos para la Unidad 3, donde empezaremos a mover motores y a construir robots que se desplazan solos.

Una pregunta que muchos se hacen al empezar: "De verdad puedo hacer esto sin saber nada de programación?" La respuesta es si, y este capítulo esta escrito exactamente para ti si nunca has escrito una línea de código. Vamos a explicar cada concepto desde cero, con analogías de la vida diaria, y cada línea de código nueva se explicara la primera vez que aparezca. No vamos a saltarnos pasos ni a suponer que sabes cosas que no te hemos enseñado. Si alguna vez sientes que algo va muy rápido, simplemente vuelve a leer la sección anterior: el material esta aquí para que lo consultes cuantas veces quieras.

Otra pregunta frecuente: "Necesito comprar el Arduino ya?" Lo ideal es que si, porque la robótica se aprende haciendo, tocando cables y viendo resultados reales. Un Arduino Uno original o compatible cuesta lo mismo que unas cuantas tortas, y un kit básico con LEDs, botones, resistencias y cables te servirá para todo el curso. Sin embargo, si todavía no lo tienes, no te detengas: puedes seguir toda la unidad leyendo el código, entendiendo la lógica y practicando con el simulador gratuito que mencionaremos mas adelante. Cuando consigas tu placa, todo lo aprendido encajara en minutos.

La estructura de esta unidad sigue el formato del curso. Primero, una explicación detallada de todos los conceptos: que es un sketch, las dos funciones magicas `setup()` y `loop()`, como instalar el IDE, como seleccionar tu placa y tu puerto, que son los pines digitales, que significan HIGH y LOW, y como se usan las funciones `digitalWrite()` y `digitalRead()`. También veremos dos formas de manejar el tiempo (`delay()` y `millis()`), aprenderemos a usar el monitor serie para depurar programas, y repasaremos los errores mas comunes que cometen los principiantes para que no te atores con ellos.

Después vienen los ejemplos paso a paso: cuatro proyectos completos con código comentado línea por línea. El clásico Blink que hace parpadear un LED, un semáforo con tres LEDs que cambia de color como el de la calle, un botón que enciende un LED usando una técnica elegante llamada INPUT_PULLUP, y un contador de pulsaciones que muestra los resultados en el monitor serie de tu computadora. Cada ejemplo incluye la lista de materiales, los pasos numerados de conexión y la explicación de cada línea de código.

Luego encontrarás la sección de práctica con cinco ejercicios diseñados para que pongas a prueba lo aprendido: un LED con ritmo personalizado, dos LEDs alternados, un botón que cambia de modo, un contador con reinicio y el famoso patrón SOS en código morse. Cada ejercicio trae su enunciado, su objetivo y su solución completa explicada, pero te recomiendo intentar resolverlos por tu cuenta antes de mirar la solución: ahí es donde de verdad se aprende.

Al final, un resumen de la unidad con los puntos clave para repasar rápido antes de seguir adelante.

Un consejo antes de empezar: ten a la mano un cuaderno o un archivo de notas. Cuando aprendas robótica, anotar tus dudas, tus ideas y los errores que cometiste (y como los resolviste) vale oro. Los mejores inventores no son los que nunca se equivocan, sino los que aprenden rápido de cada error. Y en esta unidad vas a cometer errores, te lo garantizo: a todos nos pasa. La diferencia es que aquí te vamos a enseñar a entenderlos y a corregirlos tu mismo.

Así que respira, abre tu mente y prepárate. En las próximas páginas vas a convertirte en alguien que programa cerebros electrónicos. Empezamos.

Explicación detallada

Que es Arduino y por que es el cerebro de nuestros robots

Un Arduino es una pequeña tarjeta de circuito impreso que lleva un microcontrolador, que es un chip capaz de ejecutar las instrucciones que tu le escribas. Piensa en el microcontrolador como un cerebro muy obediente pero muy literal: hace exactamente lo que le dices, ni mas ni menos, y lo hace rapidísimo, millones de instrucciones por segundo. La tarjeta Arduino agrega a ese cerebro todo lo necesario para usarlo comodamente: un conector USB para programarlo y alimentarlo, un regulador de voltaje, un botón de reinicio y filas de pines, que son como las manos y los sentidos del cerebro, por donde se conecta con el mundo exterior.

El modelo que usaremos en todo el curso es el Arduino Uno, la placa mas popular y la mas documentada del planeta. Tiene 14 pines digitales (numerados del 0 al 13) que pueden encender y apagar cosas o leer si un botón esta presionado, y 6 pines analógicos (A0 a A5) que pueden medir valores variables como la luz o la temperatura. Para esta unidad solo necesitamos los pines digitales, así que no te preocupes por los analógicos todavía: llegaran en la Unidad 4 con los sensores.

La analogía que mejor funciona para entender un Arduino es la de un cocinero que sigue una receta al pie de la letra. Tu le escribes la receta (el programa), el la lee y ejecuta cada paso en orden, una y otra vez, sin cansarse, sin distraerse y sin improvisar. Si la receta dice "prende el LED, espera un segundo, apágalo, espera un segundo" y luego "repite desde el principio", el cocinero hará exactamente eso por horas, días o semanas. Tu trabajo como programador de robots es escribir buenas recetas.

Que es un sketch

En el mundo Arduino, a los programas se les llama sketches, que en inglés significa boceto o dibujo rápido. El nombre no es casualidad: la filosofía de Arduino es que programar hardware debería ser tan rápido y experimental como hacer un boceto en una libreta. Escribes una idea, la subes a la placa en segundos, ves que pasa, la ajustas y repites. Esa velocidad para probar ideas es lo que hace a Arduino perfecto para aprender robótica.

Un sketch es simplemente un archivo de texto con instrucciones escritas en un lenguaje basado en C++, pero simplificado para que sea amable con principiantes. No necesitas aprender todo C++ para usar Arduino: con unas 20 instrucciones básicas puedes construir proyectos impresionantes. Cada sketch tiene extensión .ino y siempre, sin excepción, contiene dos funciones especiales que vamos a conocer ahora mismo.

setup() y loop(): las dos funciones magicas

Todo sketch de Arduino gira alrededor de dos funciones: setup() y loop(). Son obligatorias: si falta alguna, el programa no compila. Entender que hace cada una es entender el 50 por ciento de la programación de Arduino, así que prestales mucha atención.

La función setup() se ejecuta una sola vez, justo cuando el Arduino se enciende o cuando presionas el botón de reinicio. Su trabajo es preparar todo: decirle a la placa cuales pines seran salidas y cuales seran entradas, iniciar la comunicación con la computadora, poner valores iniciales a las variables. Piensa en setup() como la rutina de la mañana: te levantas una vez al día, te lavas los dientes, te vistes y desayunas. No repites eso cada cinco minutos.

La función `loop()` se ejecuta inmediatamente después de `setup()`, y cuando termina, vuelve a empezar, una y otra vez, para siempre, mientras el Arduino tenga energía. Ahí va la lógica principal de tu robot: leer sensores, tomar decisiones y mover actuadores. Siguiendo la analogía de la rutina, `loop()` es tu jornada de trabajo: una secuencia de tareas que repites todo el día. Un robot que evita obstáculos, por ejemplo, tiene en su `loop` algo como: "mide la distancia, si hay algo cerca gira, si no sigue adelante", repetido cientos de veces por segundo.

Esta estructura de "prepara una vez, repite siempre" es tan útil que casi todos los sistemas embebidos del mundo funcionan así. Memorízala: `setup()` configura, `loop()` ejecuta.

El IDE de Arduino: tu taller de programación

El IDE (siglas en inglés de Entorno de Desarrollo Integrado) de Arduino es el programa gratuito que instalas en tu computadora para escribir sketches, revisarlos y subirlos a la placa. Es como el taller donde el cocinero recibe sus recetas: ahí las escribes, las revisas y se las entregas.

Instalación paso a paso

El IDE funciona en Windows, macOS y Linux, y la instalación es sencilla. Sigue estos pasos:

- Entra a la página oficial: arduino.cc. Busca la sección de Software y descarga la versión 2.x, que es la más moderna y la que usaremos en el curso. Es completamente gratuita.
- En Windows, descarga el instalador .exe y ejecútalo aceptando las opciones por defecto. El instalador te pedirá permiso para instalar controladores (drivers): acepta, porque esos controladores son los que permiten que tu computadora reconozca la placa.
- En macOS, descarga el archivo .dmg, ábrelo y arrastra el icono de Arduino IDE a la carpeta de Aplicaciones.
- En Linux, descarga el archivo AppImage, dale permiso de ejecución y ejecútalo con doble clic.

Una vez instalado, abre el programa. Verás una ventana con un área blanca grande para escribir código, una barra de herramientas arriba con botones como Verificar (una palomita) y Subir (una flecha), y abajo una consola donde aparecen mensajes. No te asustes por los menús: en esta unidad solo usaremos lo básico.

Seleccionar la placa y el puerto: el error número uno de los principiantes

Antes de subir tu primer sketch, el IDE necesita saber dos cosas: a que modelo de placa le hablas y por que puerto USB esta conectada. Este paso causa mas del 80 por ciento de los problemas de los principiantes, así que hazlo con calma.

Primero conecta tu Arduino Uno a la computadora con un cable USB. Ojo: algunos cables USB baratos solo sirven para cargar y no transmiten datos. Si tu computadora no detecta nada, prueba con otro cable antes de pensar que la placa esta danada.

Luego, en el IDE, ve al menú Herramientas, luego Placa, luego Arduino AVR Boards y selecciona "Arduino Uno". Después ve a Herramientas, luego Puerto, y selecciona el puerto donde aparezca tu placa. En Windows se verá algo como COM3 o COM4. En macOS aparecera algo como /dev/cu.usbmodem14101. En Linux algo como /dev/ttyACM0. Si no aparece ningún puerto, revisa el cable, prueba otro puerto USB de tu computadora y, en Windows, verifica que los controladores se hayan instalado.

Un truco util: abre el menú de puertos antes y después de conectar el Arduino. El puerto que aparezca nuevo es el de tu placa. Anótalo si quieres, aunque normalmente no cambia mientras uses el mismo puerto USB de tu computadora.

Compilar y subir: del texto al cerebro

Escribir el sketch es solo la mitad del trabajo. La computadora necesita traducir tu texto a instrucciones que el microcontrolador entienda, un lenguaje de unos y ceros llamado código máquina. Ese proceso de traducción se llama compilar, y el IDE lo hace por ti con el botón Verificar (la palomita). Si tu código tiene errores, como una palabra mal escrita o un paréntesis sin cerrar, el compilador te lo dirá en la consola de abajo con mensajes en rojo. No te asustes: esos mensajes son tus amigos, te dicen exactamente en que línea esta el problema.

Una vez que el código compila sin errores, el botón Subir (la flecha) hace dos cosas: compila de nuevo por si acaso y luego envía el programa a la placa por el cable USB. Verás parpadear unas lucecitas en el Arduino mientras se transfiere, y al final la consola dirá "Subida exitosa" o "Done uploading". En ese momento, tu programa ya vive dentro del Arduino y empezara a ejecutarse solo, incluso si desconectas el cable USB y alimentas la placa con una batería o un eliminador. El programa se queda guardado en la memoria del Arduino hasta que subas otro.

Memoriza este ciclo porque lo repetirás cientos de veces en el curso: escribe, verifica, corrige errores, sube, observa, ajusta. Así trabajan los profesionales.

Pines digitales: las manos del cerebro

Los pines digitales son los puntos de conexión numerados que ves en los bordes del Arduino Uno, del 0 al 13. La palabra "digital" significa que solo manejan dos estados: encendido o apagado, sin términos medios. Es como un interruptor de luz: esta prendido o apagado, no hay "medio prendido". En el lenguaje de la electrónica digital, a estos dos estados se les llama HIGH (alto) y LOW (bajo).

Cuando un pin funciona como salida, el Arduino puede ponerlo en HIGH, lo que significa que entrega 5 voltios por ese pin, o en LOW, que significa 0 voltios. Esos 5 voltios son suficientes para encender un LED, activar un zumbador o mandar una señal a otro circuito. Cuando un pin funciona como entrada, el Arduino lee si lo que esta conectado le entrega un voltaje alto (cerca de 5V, lo interpreta como HIGH) o bajo (cerca de 0V, lo interpreta como LOW). Así puede saber si un botón esta presionado o no.

Una regla de oro: antes de usar un pin, debes decirle al Arduino si será entrada o salida. Eso se hace en `setup()` con la función `pinMode()`. Si olvidas este paso, el pin queda en un estado indefinido y tu circuito se comportara de forma rara e impredecible. Este es el error número dos de los principiantes, después del puerto equivocado.

HIGH y LOW: el idioma de dos palabras

Todo lo digital se reduce a dos valores, y en Arduino se escriben HIGH y LOW. HIGH significa 5 voltios (o "encendido", "verdadero", "1"), LOW significa 0 voltios (o "apagado", "falso", "0"). Es un idioma de solo dos palabras, pero con el se construyen computadoras enteras.

Es importante escribirlos exactamente así, en mayúsculas, porque el lenguaje de Arduino distingue entre mayúsculas y minúsculas: `high` no es lo mismo que `HIGH`, y el compilador te marcara error si lo escribes mal. Esta sensibilidad a mayúsculas aplica a todo: `setup` no es `Setup`, `loop` no es `Loop`, `digitalWrite` no es `digitalwrite`. Acostúmbrate desde ahora a copiar los nombres exactamente como aparecen en los ejemplos.

`pinMode()`: declarar las intenciones de cada pin

La función `pinMode()` le dice al Arduino como vas a usar un pin. Se usa normalmente dentro de `setup()`, una vez por cada pin que utilices. Su forma es:

```
pinMode(numeroDePin, modo);
```

El primer dato es el número del pin, del 0 al 13 en el Uno. El segundo dato es el modo: `OUTPUT` si el Arduino va a enviar señales por ese pin (para prender un LED, por ejemplo),

INPUT si va a leer señales que vienen de afuera (como un botón), o INPUT_PULLUP, una variante especial de entrada que veremos en detalle mas adelante y que simplifica muchísimo la conexión de botones.

Por ejemplo, pinMode(13, OUTPUT); significa "el pin 13 será una salida". Y pinMode(2, INPUT_PULLUP); significa "el pin 2 será una entrada con resistencia pull-up interna activada". Cada vez que uses un pin en tu circuito, debe haber una línea pinMode() correspondiente en setup(). Sin excepciones.

digitalWrite(): encender y apagar el mundo

La función digitalWrite() es la que pone un pin de salida en HIGH o en LOW. Su forma es:

```
digitalWrite(numeroDePin, valor);
```

Donde valor es HIGH o LOW. Por ejemplo, si conectaste un LED al pin 13 (con su resistencia, como veremos en los ejemplos), la línea digitalWrite(13, HIGH); enciende el LED, y digitalWrite(13, LOW); lo apaga. Así de simple, así de poderoso: con esta sola función puedes controlar LEDs, relees, zumbadores y las entradas de otros módulos.

Un detalle importante: digitalWrite() solo tiene sentido en pines configurados como OUTPUT. Si lo usas en un pin configurado como INPUT, no pasa nada útil (en placas viejas activaba la resistencia pull-up, pero eso es historia antigua: hoy usamos INPUT_PULLUP directamente).

digitalRead(): escuchar lo que pasa afuera

La función digitalRead() lee el estado de un pin configurado como entrada y te devuelve HIGH o LOW según lo que detecte. Su forma es:

```
digitalRead(numeroDePin);
```

Por ejemplo, int estado = digitalRead(2); guarda en la variable estado el valor HIGH o LOW según lo que este pasando en el pin 2 en ese instante. Si hay un botón conectado y alguien lo presiona, digitalRead() lo detectara. Esta función es como el oído del robot: así se entera de lo que ocurre en el mundo exterior.

Nota que digitalRead() no cambia nada, solo observa y reporta. La usaremos muchísimo dentro de loop(), porque un robot necesita estar preguntando constantemente "que esta pasando?" para reaccionar a tiempo.

INPUT_PULLUP: el truco elegante para los botones

Aquí viene uno de los conceptos mas útiles de esta unidad. Para leer un botón necesitas una resistencia pull-up o pull-down, y entender por que te ahorrará muchos dolores de cabeza.

Imagina un pin configurado como entrada sin nada conectado: esta "flotando" en el aire eléctricamente, y capta interferencia de todo lo que lo rodea, como una antena. `digitalRead()` te devolvera HIGH y LOW al azar, como si un fantasma estuviera presionando un botón invisible. Para evitarlo, el pin siempre debe estar conectado firmemente a 5V o a 0V (tierra, GND).

Una resistencia pull-up conecta el pin a 5V a través de una resistencia, de modo que cuando el botón NO esta presionado, el pin lee HIGH de forma estable. Cuando presionas el botón, este conecta el pin directamente a tierra (0V), y el pin lee LOW. Es decir: con pull-up, botón sin presionar = HIGH, botón presionado = LOW. Parece al revés, pero te acostumbraras rápido.

Una resistencia pull-down hace lo contrario: conecta el pin a tierra, así que sin presionar lee LOW y al presionar (conectando a 5V) lee HIGH. Funciona igual de bien, pero requiere que agregues una resistencia física en tu circuito.

El truco elegante es que el Arduino Uno ya trae resistencias pull-up dentro del chip, listas para usarse. Cuando configuras un pin con `pinMode(pin, INPUT_PULLUP)`; activas esa resistencia interna y ya no necesitas agregar ninguna resistencia externa para tu botón: solo conectas un lado del botón al pin y el otro lado a tierra (GND). Menos componentes, menos cables, menos errores. En este curso usaremos INPUT_PULLUP siempre que trabajemos con botones, y los ejemplos de esta unidad están diseñados así.

Recuerda la regla de oro del pull-up interno: botón suelto = HIGH, botón presionado = LOW. Cuando escribas tus condicionales `if`, compararas contra LOW para detectar la presión. Al principio se siente contraintuitivo, pero después de dos o tres prácticas se vuelve natural.

delay(): la pausa mas fácil del mundo

La función `delay()` detiene el programa durante la cantidad de milisegundos que le indiques. Su forma es:

```
delay(milisegundos);
```

Por ejemplo, `delay(1000)`; pausa todo durante un segundo (1000 milisegundos = 1 segundo), `delay(500)`; pausa medio segundo, `delay(100)`; una decima de segundo. Es la forma mas sencilla de controlar el ritmo de tu programa: prender un LED, esperar, apagarlo, esperar, y así crear parpadeos.

Pero `delay()` tiene un costo importante: mientras dura la pausa, el Arduino no hace absolutamente nada más. No lee botones, no lee sensores, no responde a nada. Es como si el cocinero de nuestra analogía se quedara dormido con la cabeza en la mesa durante la pausa. Para parpadear un LED está perfecto, pero para un robot que debe reaccionar rápido (como uno que evita obstáculos), quedarse "dormido" un segundo puede significar chocar contra la pared. Por eso existe una alternativa más avanzada.

`millis()`: medir el tiempo sin dormirse

La función `millis()` te dice cuántos milisegundos han pasado desde que el Arduino se encendió. No pausa nada, solo consulta el reloj interno y te devuelve el número. Su forma es simplemente `millis()`, sin datos entre paréntesis.

La técnica para usarla es comparar tiempos: guardas el momento en que hiciste algo y en cada vuelta de `loop()` preguntas "¿ya paso suficiente tiempo desde entonces?". Así puedes parpadear un LED cada segundo sin detener el programa, y mientras tanto seguir leyendo botones y sensores. Es como cocinar con temporizador: pones la alarma y sigues haciendo otras cosas en lugar de quedarte mirando la olla.

El patrón básico es este: necesitas una variable para guardar el último momento en que actuaste y una constante con el intervalo deseado. En `loop()`, si $(\text{millis()} - \text{ultimoMomento}) \geq \text{intervalo}$, entonces actúas y actualizas `ultimoMomento` con el valor actual de `millis()`. Esta resta funciona incluso cuando el contador de `millis()` da la vuelta después de muchos días, siempre que uses variables del tipo correcto.

Hablando de tipos: `millis()` devuelve un número entero muy grande, así que las variables que lo guardan deben ser de tipo `unsigned long` (entero largo sin signo). Si usas un `int` normal, el número no cabrá y tu temporizador fallará misteriosamente después de unos 32 segundos. Este es un error clásico: si tu temporizador con `millis()` se comporta raro, revisa que tus variables sean `unsigned long`.

Para esta unidad usaremos `delay()` en los ejemplos básicos porque es más fácil de entender, pero es importante que conozcas `millis()` desde ahora porque en las siguientes unidades, cuando el robot tenga que hacer varias cosas a la vez, será tu mejor aliada.

El monitor serie: una ventana al cerebro del robot

El monitor serie es una herramienta del IDE que abre una ventana de chat entre tu computadora y el Arduino. El Arduino puede enviarte mensajes de texto y tú puedes leerlos en tiempo real. Es la herramienta de depuración más valiosa para un principiante: cuando tú

programa no hace lo que esperabas, en lugar de adivinar, le pides al Arduino que te cuente que esta pasando.

Para usarlo necesitas tres pasos. Primero, en `setup()`, iniciar la comunicación con la línea `Serial.begin(9600);`. El número 9600 es la velocidad de comunicación en baudios: ambas partes deben hablar a la misma velocidad, y 9600 es el estándar para empezar. Segundo, en tu código, enviar mensajes con `Serial.println()`, que imprime el texto o el número que le des y salta a la siguiente línea, o `Serial.print()`, que imprime sin saltar de línea. Tercero, abrir el monitor serie en el IDE con el botón de la lupa en la esquina superior derecha, verificando que la velocidad seleccionada abajo a la derecha también sea 9600.

Por ejemplo, `Serial.println("Hola desde Arduino");` mostrará ese texto en el monitor. Y `Serial.println(estadoBoton);` mostrará el valor de la variable, 0 o 1. Puedes combinar texto y variables con varias líneas: `Serial.print("El botón vale: ");` `Serial.println(estadoBoton);` mostrará "El botón vale: 1" en una sola línea.

Usa el monitor serie para todo: verificar que un botón se lee bien, ver el valor de una variable que no entiendes, confirmar que tu programa llega a cierta parte del código. Los programadores profesionales de sistemas embebidos lo usan todos los días. Acostúmbrate a poner mensajes de depuración mientras desarrollas y quitarlos (o dejarlos) cuando el proyecto funcione.

Un error común: abrir el monitor serie con una velocidad distinta a la que pusiste en `Serial.begin()`. Si ves símbolos raros en lugar de texto, casi seguro es eso: iguala las velocidades y listo.

Analogías para no olvidar

Repasemos con analogías de la vida diaria, porque lo que se entiende con una historia se recuerda para siempre:

- El Arduino es el cerebro, los pines son sus manos (salidas) y sus sentidos (entradas).
- El sketch es la receta que el cocinero (microcontrolador) sigue al pie de la letra.
- `setup()` es la rutina de la mañana: se hace una vez. `loop()` es la jornada laboral: se repite todo el día.
- Compilar es traducir tu receta del español al idioma del cerebro; subir es entregarsela en mano.
- HIGH y LOW son "sí" y "no", "prendido" y "apagado": el idioma mas simple del mundo.

- Un pin de entrada sin resistencia pull-up es como una puerta sin seguro: cualquiera (el ruido eléctrico) puede entrar y hacer desorden.
- `delay()` es quedarse dormido: fácil pero te pierdes de todo. `millis()` es poner el temporizador de la cocina: sigues trabajando mientras esperas.
- El monitor serie es el diario del robot: te cuenta que esta pensando mientras trabaja.

Errores comunes y como resolverlos

Todos los principiantes se topan con los mismos obstáculos. Conocerlos de antemano te ahorrará horas de frustración. Aquí están los mas frecuentes, con su síntoma y su solución.

El primero es la placa no detectada o el puerto equivocado. Síntoma: al subir el sketch aparece un error como "no se encontró la placa" o el menú de puertos esta vacío. Soluciones: cambia el cable USB (muchos solo cargan), prueba otro puerto USB de la computadora, reinstala los controladores en Windows, y verifica que seleccionaste "Arduino Uno" en el menú de placas. En clones baratos de Arduino a veces se necesita el controlador CH340, que se descarga gratis de internet.

El segundo es olvidar `pinMode()`. Síntoma: el LED no prende, o prende muy tenue, o el botón se comporta de forma aleatoria. Solución: revisa que cada pin que uses tenga su línea `pinMode()` en `setup()` con el modo correcto. Hazlo un hábito: cada vez que conectes un componente nuevo, escribe su `pinMode()` antes de cualquier otra cosa.

El tercero son los errores de sintaxis. Síntoma: la consola muestra mensajes rojos al compilar, como "expected ';' before '}'". Los culpables habituales: olvidar el punto y coma al final de cada instrucción, no cerrar un paréntesis o una llave, escribir mal el nombre de una función (recuerda las mayúsculas y minúsculas). Solución: lee el mensaje de error, fíjate en el número de línea que indica y revisa esa zona. Con la práctica, corregiras estos errores en segundos.

El cuarto es el LED conectado al revés. Síntoma: todo el código esta bien, el `pinMode` esta bien, pero el LED no prende. Los LEDs tienen polaridad: la pata larga (anodo) va hacia el pin del Arduino (el positivo) y la pata corta (catodo) va hacia tierra (GND), siempre con una resistencia en serie. Si lo conectas al revés, no prende. No se daña con 5V en la mayoría de los casos, simplemente no enciende. Dale la vuelta y listo.

El quinto es olvidar la resistencia del LED. Síntoma: el LED prende pero muy brillante, se calienta, o peor, deja de funcionar para siempre. Un LED conectado directo entre 5V y tierra recibe demasiada corriente y se quema. Siempre usa una resistencia de 220 ohmios en serie con cada LED. Es barata y salva vidas (de LEDs). En los ejemplos de esta unidad siempre la incluiremos.

El sexto es comparar mal el botón con INPUT_PULLUP. Síntoma: el programa hace lo contrario de lo esperado, como si el botón estuviera "al revés". Causa: con INPUT_PULLUP, el botón presionado se lee como LOW, no como HIGH. Solución: en tus condicionales, pregunta `if (digitalRead(pinBoton) == LOW)` para detectar la presión. Si vienes de tutoriales que usan resistencia pull-down externa, este es el punto donde mas se confunde la gente.

El séptimo es el monitor serie con velocidad incorrecta. Síntoma: aparecen símbolos extraños en lugar de tu texto. Solución: la velocidad en `Serial.begin()` debe coincidir con la seleccionada en la esquina del monitor serie. Usa 9600 en ambos lados y olvídate del problema.

El octavo es subir el sketch con el monitor serie abierto en el puerto equivocado o con otro programa usando el puerto. Síntoma: error de subida aunque todo lo demás este bien. Solución: cierra el monitor serie antes de subir (las versiones nuevas del IDE lo manejan solas, pero en algunas computadoras ayuda) y asegúrate de que ningún otro programa este usando ese puerto USB.

Buenas prácticas desde el primer día

Los buenos hábitos se forman al principio. Estos cinco te convertiran en un programador ordenado desde tu primer sketch:

Primero, comenta tu código. Un comentario es una nota que escribes para tu yo del futuro (o para otra persona) y que el Arduino ignora. Se escribe con dos diagonales: `//` esto es un comentario. Explica el "por que", no el "que": en lugar de `// prende el LED`, escribe `// prende el LED de alarma porque el botón de emergencia fue presionado`. Tu yo de dentro de tres meses te lo agradecerá.

Segundo, usa nombres de variables descriptivos. En lugar de `int x = 13;`, escribe `int pinLED = 13;`. Cuando tu programa crezca y tenga diez pines, agradeceras saber cual es cual sin tener que adivinar. Una convención util: que el nombre diga que es y para que sirve, como `pinBoton`, `estadoAnterior`, `intervaloParpadeo`.

Tercero, define tus pines al inicio. En lugar de escribir el número 13 regado por todo el código, define `const int pinLED = 13;` al principio del sketch y usa `pinLED` en todas partes. Si un día cambias el LED al pin 8, solo cambias una línea en lugar de cazar todos los 13 del programa. La palabra `const` significa que ese valor no cambiara nunca, y el compilador te avisara si intentas cambiarlo por accidente.

Cuarto, prueba en partes pequeñas. No escribas un programa gigante de 200 líneas y luego preguntes por que no funciona. Escribe un pedacito, subelo, verifica que funcione, y luego agrega el siguiente pedacito. Esta técnica se llama desarrollo incremental y es como los profesionales construyen sistemas enormes sin volverse locos.

Quinto, guarda versiones de tus sketches. Antes de hacer un cambio grande, guarda una copia con otro nombre, como MiProyecto_v1, MiProyecto_v2. Así, si algo se rompe, puedes volver a la version que funcionaba. El IDE tiene la opción "Guardar como" en el menú Archivo para esto.

El simulador: práctica sin hardware

Si todavía no tienes tu Arduino físico, puedes practicar toda esta unidad en un simulador gratuito en línea. El mas popular es Tinkercad Circuits, de Autodesk: creas una cuenta gratuita, arrastras un Arduino Uno virtual, conectas LEDs y botones con el mouse, escribes tu sketch y lo "subes" al Arduino virtual para verlo funcionar, incluso con el monitor serie simulado. No es idéntico a la experiencia real (nada supera tocar componentes de verdad), pero la lógica, el código y los conceptos son exactamente los mismos. Cuando consigas tu placa física, todo lo que practicaste en el simulador funcionará igual.

Lo que debes llevarte de esta sección

Antes de pasar a los ejemplos, verifica que puedes responder estas preguntas sin mirar: que se ejecuta una sola vez, `setup()` o `loop()`? Que función configura un pin como salida? Como se enciende un LED conectado al pin 13? Que devuelve `digitalRead()` cuando un botón con `INPUT_PULLUP` esta presionado? Que hace `delay(500)`? Para que sirve el monitor serie? Si respondiste (`setup`, `pinMode`, `digitalWrite(13, HIGH)`, `LOW`, pausa medio segundo, depurar viendo mensajes), estas listo para construir. Si alguna te costo trabajo, vuelve a leer su sección: no hay prisa, los cimientos bien puestos sostienen todo el edificio.

Ejemplos paso a paso

En esta sección vas a construir cuatro proyectos reales. Cada uno incluye la lista de materiales, los pasos numerados de conexión, el código completo comentado y la explicación línea por línea. Te recomiendo armarlos en orden, porque cada ejemplo usa lo aprendido en el anterior.

Antes de empezar, una nota sobre el protoboard: es esa tablilla blanca con agujeritos donde conectarás los componentes sin soldar. Los agujeros están conectados por dentro en

columnas, así que para unir dos cosas basta meter sus patitas en agujeros de la misma columna. No necesitas entenderlo a fondo ahora: sigue los pasos y funcionará.

Ejemplo A: Blink, tu primer parpadeo

Este es el "hola mundo" de la robótica: hacer parpadear un LED. Parece simple, pero incluye todo el ciclo completo: configurar un pin, encenderlo, esperar, apagarlo, esperar y repetir.

Materiales:

- 1 Arduino Uno con cable USB
- 1 LED (cualquier color)
- 1 resistencia de 220 ohmios (bandas: rojo, rojo, café)
- 2 cables de conexión
- 1 protoboard

Pasos de conexión:

1. Conecta un cable del pin 13 del Arduino a una columna libre del protoboard.
2. Inserta la resistencia de 220 ohmios: una pata en esa misma columna y la otra pata en otra columna libre.
3. Inserta el LED: la pata larga en la columna donde quedó el segundo extremo de la resistencia, y la pata corta en otra columna libre.
4. Conecta un cable desde la columna de la pata corta del LED hasta un pin GND del Arduino.
5. Revisa: el camino de la electricidad va del pin 13, pasa por la resistencia, entra por la pata larga del LED, sale por la pata corta y regresa a tierra (GND). La resistencia limita la corriente para que el LED no se queme.

Código:

```
// Ejemplo A: Blink - parpadea un LED conectado al pin 13
// La resistencia de 220 ohmios protege al LED del exceso de corriente

const int pinLED = 13; // el LED esta conectado al pin digital 13

void setup() {
  pinMode(pinLED, OUTPUT); // configura el pin 13 como salida
}

void loop() {
  digitalWrite(pinLED, HIGH); // enciende el LED (5 voltios en el pin)
  delay(1000);                // espera 1 segundo con el LED encendido
  digitalWrite(pinLED, LOW);  // apaga el LED (0 voltios en el pin)
  delay(1000);                // espera 1 segundo con el LED apagado
}
```

Explicación línea por línea:

- Las dos primeras líneas son comentarios: el Arduino las ignora, son notas para ti.
- `const int pinLED = 13;` crea una constante llamada `pinLED` con valor 13. Así, si cambias el LED de pin, solo editas esta línea.
- `void setup() { ... }` se ejecuta una sola vez al encender.
- `pinMode(pinLED, OUTPUT);` declara que el pin 13 será salida: el Arduino enviara voltaje por ahí.
- `void loop() { ... }` se repite para siempre.
- `digitalWrite(pinLED, HIGH);` pone 5V en el pin: el LED enciende.
- `delay(1000);` congela el programa un segundo. El LED se queda encendido ese tiempo.
- `digitalWrite(pinLED, LOW);` pone 0V: el LED se apaga.
- `delay(1000);` espera otro segundo con el LED apagado. Al terminar `loop()`, vuelve a empezar y el LED parpadea eternamente.

Prueba esto: cambia los dos `delay(1000)` por `delay(200)` y sube el sketch de nuevo. El LED parpadeara rapidísimo. Cambialos por `delay(2000)` y parpadeara lento. Ya estas controlando el ritmo de una máquina con numeros que tu eliges. Ese es el poder de programar.

Ejemplo B: semaforo con tres LEDs

Ahora subimos de nivel: tres LEDs (rojo, amarillo y verde) que se encienden en secuencia como un semaforo de verdad. Aprenderás a manejar varias salidas y a organizar secuencias con `delay()`.

Materiales:

- 1 Arduino Uno con cable USB
- 3 LEDs: uno rojo, uno amarillo (o naranja) y uno verde
- 3 resistencias de 220 ohmios
- Cables de conexión y protoboard

Pasos de conexión:

1. Conecta el pin 11 a una columna del protoboard, y ahí mismo una resistencia de 220 ohmios. Del otro extremo de la resistencia, conecta la pata larga del LED rojo. La pata corta del LED rojo va a una columna conectada a GND.
2. Repite lo mismo con el pin 10 y el LED amarillo: pin 10, resistencia, pata larga del LED amarillo, pata corta a GND.
3. Repite con el pin 9 y el LED verde: pin 9, resistencia, pata larga del LED verde, pata corta a GND.
4. Puedes usar la misma línea de tierra (GND) para las tres patas cortas: en el protoboard, conecta las tres columnas de las patas cortas entre si con cables cortos, y de ahí un solo cable a GND del Arduino.

Código:

```

// Ejemplo B: semaforo con 3 LEDs
// Rojo = pin 11, Amarillo = pin 10, Verde = pin 9
// Cada LED lleva su resistencia de 220 ohmios en serie

const int pinRojo = 11;
const int pinAmarillo = 10;
const int pinVerde = 9;

void setup() {
  pinMode(pinRojo, OUTPUT);    // los tres pines son salidas
  pinMode(pinAmarillo, OUTPUT);
  pinMode(pinVerde, OUTPUT);
}

void loop() {
  // Fase 1: luz verde (avanza) durante 4 segundos
  digitalWrite(pinVerde, HIGH);
  delay(4000);
  digitalWrite(pinVerde, LOW);

  // Fase 2: luz amarilla (precaucion) durante 1.5 segundos
  digitalWrite(pinAmarillo, HIGH);
  delay(1500);
  digitalWrite(pinAmarillo, LOW);

  // Fase 3: luz roja (alto) durante 4 segundos
  digitalWrite(pinRojo, HIGH);
  delay(4000);
  digitalWrite(pinRojo, LOW);

  // Al terminar, loop() repite y el ciclo del semaforo continua
}

```

Explicación línea por línea:

- Definimos tres constantes, una por color, cada una con su número de pin. Los nombres descriptivos evitan confusiones.
- En setup(), los tres pines se configuran como OUTPUT con tres líneas pinMode(). Cada pin necesita la suya.
- La fase 1 enciende el verde con digitalWrite(pinVerde, HIGH), espera 4 segundos y lo apaga con LOW. Nota el patrón: encender, esperar, apagar. Se repite en cada fase.
- La fase 2 hace lo mismo con el amarillo durante 1.5 segundos (1500 milisegundos).

- La fase 3 hace lo mismo con el rojo durante 4 segundos.
- Cuando loop() termina, empieza de nuevo: el semaforo funciona para siempre sin que hagas nada mas.

Observa como el programa se lee casi como las instrucciones que le darías a una persona: "prende el verde, espera cuatro segundos, apágalo, prende el amarillo...". Programar es eso: describir con precisión lo que quieres que pase, paso a paso. Experimenta cambiando los tiempos o el orden de las fases y observa el resultado.

Ejemplo C: botón que enciende un LED con INPUT_PULLUP

Hasta ahora el Arduino solo ha hablado (salidas). Ahora va a escuchar (entradas). Conectaremos un botón usando INPUT_PULLUP, la técnica elegante que no necesita resistencia externa, y haremos que el LED se encienda solo mientras el botón este presionado.

Materiales:

- 1 Arduino Uno con cable USB
- 1 LED con su resistencia de 220 ohmios
- 1 botón pulsador (pushbutton) de 4 patas
- Cables de conexión y protoboard

Pasos de conexión:

1. Conecta el LED como en el Ejemplo A: pin 13, resistencia de 220 ohmios, pata larga del LED, pata corta a GND.
2. Coloca el botón en el protoboard de modo que quede a caballo sobre la ranura central (así sus dos lados quedan separados eléctricamente).
3. Conecta un cable del pin 2 del Arduino a una de las patas del botón de un lado de la ranura.
4. Conecta un cable desde una pata del botón del otro lado de la ranura hasta GND del Arduino.
5. Eso es todo: no hay resistencia externa porque usaremos la pull-up interna del Arduino.

Como funciona el circuito: con INPUT_PULLUP, la resistencia interna conecta el pin 2 a 5V, así que sin presionar el botón, digitalRead(2) devuelve HIGH. Al presionar el botón, conectas el pin 2 directamente a GND, y digitalRead(2) devuelve LOW. Recuerda: presionado = LOW.

Código:

```
// Ejemplo C: boton que enciende un LED (con INPUT_PULLUP)
// Boton en pin 2: sin presionar lee HIGH, presionado lee LOW
// LED en pin 13 con resistencia de 220 ohmios

const int pinBoton = 2;
const int pinLED = 13;

void setup() {
  pinMode(pinBoton, INPUT_PULLUP); // entrada con pull-up interno
  pinMode(pinLED, OUTPUT);         // salida para el LED
}

void loop() {
  int estadoBoton = digitalRead(pinBoton); // lee el boton: HIGH o LOW

  if (estadoBoton == LOW) {
    // El boton esta presionado (recuerda: con pull-up, presionado = LOW)
    digitalWrite(pinLED, HIGH); // enciende el LED
  } else {
    // El boton no esta presionado
    digitalWrite(pinLED, LOW);  // apaga el LED
  }
}
```

Explicación línea por línea:

- `const int pinBoton = 2;` y `const int pinLED = 13;` definen los pines con nombres claros.
- `pinMode(pinBoton, INPUT_PULLUP);` configura el pin 2 como entrada y activa la resistencia pull-up interna. Sin esta línea mágica necesitarías una resistencia física.
- `pinMode(pinLED, OUTPUT);` configura el pin 13 como salida, como ya sabes.
- `int estadoBoton = digitalRead(pinBoton);` lee el pin 2 y guarda el resultado (HIGH o LOW) en la variable `estadoBoton`. Esta lectura ocurre en cada vuelta de `loop()`, cientos de veces por segundo.
- `if (estadoBoton == LOW) { ... }` es un condicional: "si el botón esta presionado (LOW con pull-up), entonces...". El doble signo de igual `==` significa comparación: pregunta si son iguales. Un solo `=` asignaría un valor, y es un error típico de principiantes confundirlos.
- Dentro del `if`, `digitalWrite(pinLED, HIGH);` enciende el LED.

- `else { ... }` significa "en cualquier otro caso": si el botón no está presionado, `digitalWrite(pinLED, LOW);` apaga el LED.

Prueba mantener el botón presionado: el LED se queda encendido. Suéltalo: se apaga al instante. Tu programa está reaccionando al mundo real en tiempo real. Esa es la base de todo robot: sentir y responder.

Ejemplo D: contador de pulsaciones en el monitor serie

El último ejemplo combina entradas con el monitor serie: cada vez que presiones el botón, el Arduino contará una pulsación y te mostrará el total en la computadora. Además aprenderás a detectar el momento exacto en que se presiona el botón, no solo si está presionado.

Materiales:

- Los mismos del Ejemplo C (Arduino, LED con resistencia, botón, cables, protoboard)

Pasos de conexión:

1. Arma el mismo circuito del Ejemplo C: LED en pin 13, botón entre pin 2 y GND.
2. Conecta el Arduino a la computadora por USB.

Código:

```

// Ejemplo D: contador de pulsaciones con monitor serie
// Cada presion del boton suma 1 y muestra el total en el monitor serie
// Boton en pin 2 con INPUT_PULLUP, LED en pin 13

const int pinBoton = 2;
const int pinLED = 13;

int contador = 0;           // lleva la cuenta de pulsaciones
int estadoAnterior = HIGH;  // el boton empieza sin presionar (HIGH con pull-up)

void setup() {
  pinMode(pinBoton, INPUT_PULLUP);
  pinMode(pinLED, OUTPUT);
  Serial.begin(9600); // inicia la comunicacion con la computadora a 9600 baudios
  Serial.println("Contador listo. Presiona el boton...");
}

void loop() {
  int estadoActual = digitalRead(pinBoton); // lee el boton ahora mismo

  // Detecta el flanco de bajada: el momento exacto en que se presiona
  // Antes estaba HIGH (suelto) y ahora esta LOW (presionado)
  if (estadoAnterior == HIGH && estadoActual == LOW) {
    contador = contador + 1; // suma una pulsacion
    Serial.print("Pulsacion numero: ");
    Serial.println(contador); // muestra el total en el monitor serie
    digitalWrite(pinLED, HIGH); // parpadea el LED como confirmacion
    delay(150);
    digitalWrite(pinLED, LOW);
  }

  estadoAnterior = estadoActual; // guarda el estado para la proxima vuelta
  delay(20); // pequena pausa para estabilidad
}

```

Explicación línea por línea:

- `int contador = 0;` crea la variable que llevara la cuenta, empezando en cero. Esta declarada fuera de `setup()` y `loop()` para que su valor se conserve entre vueltas (es una variable global).
- `int estadoAnterior = HIGH;` guarda como estaba el botón en la vuelta anterior. Empieza en HIGH porque con pull-up el botón suelto lee HIGH.

- En `setup()`, `Serial.begin(9600)`; abre la comunicación con la computadora. `Serial.println("Contador listo...");` envía un mensaje de bienvenida que verás en el monitor serie.
- En `loop()`, `int estadoActual = digitalRead(pinBoton)`; lee el estado actual del botón.
- El `if` compara dos cosas a la vez con `&&` (que significa "y"): `estadoAnterior == HIGH && estadoActual == LOW`. Esto solo es verdadero en el instante exacto en que el botón pasa de suelto a presionado. Se llama detección de flanco, y es clave: sin ella, el contador sumaría cientos de veces mientras mantienes el botón presionado, porque `loop()` se repite rapidísimo.
- Dentro del `if`: `contador = contador + 1`; suma uno. También podría escribirse `contador++`;, que es la forma corta que usan los programadores.
- `Serial.print("Pulsacion número: ");` imprime el texto sin saltar de línea, y `Serial.println(contador)`; imprime el número y salta de línea. Juntos muestran "Pulsacion número: 1", "Pulsacion número: 2", etc.
- El LED parpadea 150 milisegundos como confirmación visual de que la pulsacion se registro.
- `estadoAnterior = estadoActual`; actualiza la memoria del estado para la siguiente vuelta. Sin esta línea, la detección de flanco no funcionaria.
- `delay(20)`; es una pausa mínima que ayuda a la estabilidad de la lectura.

Para verlo funcionar, sube el sketch y abre el monitor serie (botón de la lupa, verifica que diga 9600 baudios). Presiona el botón varias veces y observa como el contador aumenta. Acabas de construir tu primera interfaz humano-máquina: presionas algo físico y la computadora responde con información. Los robots usan este mismo principio para contar vueltas de rueda, detectar objetos y mucho mas.

Práctica

Llegó la hora de ensuciarse las manos. Los ejemplos anteriores te mostraron el camino; estos ejercicios son para que camines solo. Intenta resolver cada uno por tu cuenta antes de mirar la solución: esta permitido (y recomendado) releer las secciones anteriores, consultar los ejemplos y equivocarse. Solo después de intentarlo en serio, compara con la solución propuesta. Recuerda que en programación casi siempre hay mas de una respuesta correcta: si tu código funciona y se entiende, esta bien aunque sea distinto al mío.

Ejercicio 1: LED con ritmo personalizado

Enunciado: conecta un LED al pin 13 con su resistencia de 220 ohmios. Escribe un sketch que haga parpadear el LED con tu ritmo personal: dos parpadeos rápidos seguidos de una pausa larga, repitiendo el patrón para siempre. Los parpadeos rápidos deben durar 150 milisegundos encendido y 150 milisegundos apagado, y la pausa larga debe ser de 2 segundos.

Objetivo: practicar la secuencia básica encender-esperar-apagar-esperar y entender que los tiempos de delay() definen el comportamiento visible del programa. También practicarás organizar un patrón repetitivo dentro de loop().

Solución del ejercicio 1

```
// Solucion ejercicio 1: ritmo personalizado (dos parpadeos rapidos + pausa larga)
const int pinLED = 13;

void setup() {
  pinMode(pinLED, OUTPUT);
}

void loop() {
  // Primer parpadeo rapido
  digitalWrite(pinLED, HIGH);
  delay(150);
  digitalWrite(pinLED, LOW);
  delay(150);

  // Segundo parpadeo rapido
  digitalWrite(pinLED, HIGH);
  delay(150);
  digitalWrite(pinLED, LOW);
  delay(150);

  // Pausa larga de 2 segundos antes de repetir el patron
  delay(2000);
}
```

Explicación: el patrón completo vive dentro de loop(), así que se repite solo. Cada parpadeo es el clásico encender-esperar-apagar-esperar con 150 milisegundos. Después de los dos parpadeos, un delay(2000) crea la pausa larga. Si quisieras tres parpadeos, simplemente agregarías otro bloque igual. Observa como un comportamiento complejo (un ritmo) se construye repitiendo un comportamiento simple (un parpadeo): esa es la esencia de programar secuencias.

Ejercicio 2: dos LEDs alternados

Enunciado: conecta un LED rojo al pin 12 y un LED verde al pin 11, cada uno con su resistencia de 220 ohmios. Escribe un sketch donde los LEDs se alternen: mientras el rojo esta encendido el verde esta apagado, y viceversa, cambiando cada 500 milisegundos. Debe parecer una luz de patrulla sencilla.

Objetivo: practicar el control de varias salidas al mismo tiempo y coordinar sus estados. Aprenderás que cada pin se controla de forma independiente con su propio digitalWrite().

Solución del ejercicio 2

```
// Solucion ejercicio 2: dos LEDs alternados como luz de patrulla
const int pinRojo = 12;
const int pinVerde = 11;

void setup() {
  pinMode(pinRojo, OUTPUT);
  pinMode(pinVerde, OUTPUT);
}

void loop() {
  // Estado A: rojo encendido, verde apagado
  digitalWrite(pinRojo, HIGH);
  digitalWrite(pinVerde, LOW);
  delay(500);

  // Estado B: rojo apagado, verde encendido
  digitalWrite(pinRojo, LOW);
  digitalWrite(pinVerde, HIGH);
  delay(500);
}
```

Explicación: la clave esta en poner los dos digitalWrite() juntos antes de cada delay(). En el estado A, el rojo se enciende y el verde se apaga al mismo tiempo; esperamos medio

segundo; luego invertimos ambos y esperamos de nuevo. Como `loop()` repite, la alternancia continua para siempre. Fíjate que nunca hay un momento en que ambos estén encendidos o ambos apagados: los estados son opuestos por diseño. Este patrón de "estados alternados" aparece en robótica real, por ejemplo en las direccionales de un carrito.

Ejercicio 3: botón que cambia de modo

Enunciado: usa el LED del pin 13 y el botón del pin 2 con `INPUT_PULLUP`. Escribe un sketch con dos modos: modo 0, el LED parpadea lento (1 segundo encendido, 1 segundo apagado); modo 1, el LED parpadea rápido (200 milisegundos encendido, 200 milisegundos apagado). Cada vez que presiones el botón, el modo debe cambiar al otro. El programa debe arrancar en modo 0. Pista: necesitas una variable para guardar el modo actual y detectar el flanco de presión como en el Ejemplo D.

Objetivo: combinar entradas y salidas en un programa con "memoria": el Arduino debe recordar en que modo esta aunque el botón ya se haya soltado. Practicarás variables de estado, detección de flanco y condicionales `if/else`.

Solución del ejercicio 3

```

// Solucion ejercicio 3: el boton cambia el modo de parpadeo
// Modo 0 = parpadeo lento, Modo 1 = parpadeo rapido

const int pinBoton = 2;
const int pinLED = 13;

int modo = 0;           // 0 = lento, 1 = rapido; empieza en lento
int estadoAnterior = HIGH; // para detectar el momento de la presion

void setup() {
  pinMode(pinBoton, INPUT_PULLUP);
  pinMode(pinLED, OUTPUT);
}

void loop() {
  int estadoActual = digitalRead(pinBoton);

  // Si se acaba de presionar el boton, cambia de modo
  if (estadoAnterior == HIGH && estadoActual == LOW) {
    if (modo == 0) {
      modo = 1; // estaba en lento, pasa a rapido
    } else {
      modo = 0; // estaba en rapido, pasa a lento
    }
    delay(50); // antirrebote simple: ignora rebotes del boton
  }
  estadoAnterior = estadoActual;

  // Ejecuta el parpadeo segun el modo actual
  if (modo == 0) {
    digitalWrite(pinLED, HIGH);
    delay(1000);
    digitalWrite(pinLED, LOW);
    delay(1000);
  } else {
    digitalWrite(pinLED, HIGH);
    delay(200);
    digitalWrite(pinLED, LOW);
    delay(200);
  }
}

```

Explicación: la variable modo es la "memoria" del programa: vale 0 o 1 y conserva su valor entre vueltas de loop(). La detección de flanco (estadoAnterior == HIGH && estadoActual ==

LOW) identifica el instante de la presión, y ahí un if/else invierte el modo. El delay(50) después del cambio es un antirrebote sencillo: los botones mecánicos "rebotan" al presionarse, generando varias señales rapidísimas, y esa pequeña pausa evita que un solo presión cuente como varios. En la segunda parte, otro if/else elige el ritmo de parpadeo según el modo. Este ejercicio es un mini sistema con estados, igual que los robots que cambian de comportamiento según lo que detectan.

Ejercicio 4: contador con reinicio

Enunciado: usa el botón del pin 2 con INPUT_PULLUP y el monitor serie. Escribe un sketch que cuente las pulsaciones como en el Ejemplo D, pero con una mejora: si mantienes el botón presionado durante más de 2 segundos, el contador se reinicia a cero y el monitor muestra el mensaje "Contador reiniciado". Pista: cuando detectes que el botón está presionado (LOW), guarda el momento con millis() y revisa cuánto tiempo lleva presionado; si supera 2000 milisegundos, reinicia.

Objetivo: practicar millis() para medir duraciones sin bloquear el programa, y combinar la detección de presión corta (contar) con presión larga (reiniciar). Es el mismo principio de los botones de los celulares: un toque hace una cosa, mantener presionado hace otra.

Solución del ejercicio 4

```
// Solucion ejercicio 4: contador con reinicio por presion larga
// Toque corto = suma 1. Presion de mas de 2 segundos = reinicia a 0.

const int pinBoton = 2;

int contador = 0;
int estadoAnterior = HIGH;
unsigned long inicioPresion = 0; // momento en que se presiono el boton
bool reiniciado = false;         // evita reiniciar varias veces

void setup() {
  pinMode(pinBoton, INPUT_PULLUP);
  Serial.begin(9600);
  Serial.println("Contador con reinicio listo.");
  Serial.println("Toque corto: suma. Presion larga (2 s): reinicia.");
}

void loop() {
  int estadoActual = digitalRead(pinBoton);
```



```

// Flanco de bajada: acaba de presionarse, guarda el momento
if (estadoAnterior == HIGH && estadoActual == LOW) {
    inicioPresion = millis();
    reiniciado = false;
}

// Mientras esta presionado, mide cuanto tiempo lleva asi
if (estadoActual == LOW) {
    unsigned long duracion = millis() - inicioPresion;
    if (duracion >= 2000 && reiniciado == false) {
        contador = 0;
        reiniciado = true;
        Serial.println("Contador reiniciado a 0");
    }
}

// Flanco de subida: se acaba de soltar
// Si fue un toque corto (no se reinicio), suma una pulsacion
if (estadoAnterior == LOW && estadoActual == HIGH) {
    if (reiniciado == false) {
        contador = contador + 1;
        Serial.print("Pulsacion numero: ");
        Serial.println(contador);
    }
}

estadoAnterior = estadoActual;
delay(20);
}

```

Explicación: este programa distingue dos eventos: el flanco de bajada (se presiona) y el flanco de subida (se suelta). Al presionarse, guardamos `inicioPresion = millis()`. Mientras el botón sigue presionado, calculamos `duración = millis() - inicioPresion`; si llega a 2000 milisegundos, reiniciamos el contador y marcamos `reiniciado = true` para no reiniciar de nuevo en la siguiente vuelta. Al soltarse, si no hubo reinicio, fue un toque corto y sumamos uno. La variable bool `reiniciado` (que solo vale true o false) evita que un toque largo sume además de reiniciar. Nota que usamos `unsigned long` para los tiempos, como manda la buena práctica con `millis()`. Este patrón de "toque corto vs presión larga" lo verás en controles remotos, videojuegos y robots.

Ejercicio 5: patrón SOS en código morse

Enunciado: conecta un LED al pin 13 con su resistencia. El código morse representa la letra S con tres puntos (parpadeos cortos) y la letra O con tres rayas (parpadeos largos). La señal de auxilio SOS es: S, O, S. Escribe un sketch que repita la secuencia SOS con el LED: punto = 200 ms encendido, raya = 600 ms encendido, pausa entre símbolos = 200 ms apagado, pausa entre letras = 600 ms apagado, pausa entre repeticiones de SOS = 2 segundos. Para no repetir tanto código, crea dos funciones propias: void punto() y void raya(), que enciendan el LED el tiempo correspondiente y manejen sus pausas.

Objetivo: practicar la creación de tus propias funciones para reutilizar código, y construir una secuencia compleja a partir de piezas simples. Las funciones propias son una de las herramientas mas poderosas de la programación.

Solución del ejercicio 5

```

// Solucion ejercicio 5: SOS en codigo morse con funciones propias
// Punto = 200 ms, raya = 600 ms

const int pinLED = 13;

void setup() {
  pinMode(pinLED, OUTPUT);
}

// Funcion propia: emite un punto (parpadeo corto)
void punto() {
  digitalWrite(pinLED, HIGH);
  delay(200);           // duracion del punto
  digitalWrite(pinLED, LOW);
  delay(200);           // pausa entre simbolos
}

// Funcion propia: emite una raya (parpadeo largo)
void raya() {
  digitalWrite(pinLED, HIGH);
  delay(600);           // duracion de la raya
  digitalWrite(pinLED, LOW);
  delay(200);           // pausa entre simbolos
}

void loop() {
  // Letra S: tres puntos
  punto(); punto(); punto();
  delay(400); // completa la pausa entre letras (200 ya pasaron + 400 = 600)

  // Letra O: tres rayas
  raya(); raya(); raya();
  delay(400); // completa la pausa entre letras

  // Letra S: tres puntos
  punto(); punto(); punto();

  delay(2000); // pausa larga antes de repetir el SOS
}

```

Explicación: void punto() y void raya() son funciones que tu mismo defines, igual que setup() y loop(). La palabra void significa que no devuelven ningún valor; solo ejecutan acciones. Una vez definidas, puedes "llamarlas" escribiendo su nombre seguido de paréntesis: punto(); ejecuta todo el código dentro de la función. Gracias a ellas, loop() se lee clarísimo: tres puntos,

pausa, tres rayas, pausa, tres puntos, pausa larga. Sin funciones, `loop()` tendría mas de 30 líneas repetidas y seria fácil equivocarse. Cada vez que notes que repites el mismo bloque de código tres veces o mas, es señal de que merece su propia función. Los robots reales usan decenas de funciones propias: `avanzar()`, `girarIzquierda()`, `leerSensor()`, etc.

Ejercicio extra: semaforo con botón de peatón

Si terminaste los cinco ejercicios y quieres un reto, intenta este: modifica el semaforo del Ejemplo B para agregar un botón de peatón en el pin 2 (con `INPUT_PULLUP`). Cuando alguien presione el botón, el semaforo debe terminar su fase actual y luego mostrar la secuencia de cruce: luz roja para los autos durante 5 segundos con el LED verde del peatón (usa otro LED en el pin 8) encendido, y después volver al ciclo normal. No incluimos la solución: es tu primer proyecto de diseño libre. Si lo logras, ya piensas como un programador de sistemas reales.

Resumen de la unidad

Felicidades: terminaste la Unidad 2 y ya sabes programar el cerebro de un robot. Hagamos un repaso de todo lo que ahora forma parte de tu caja de herramientas.

Arduino es una tarjeta con un microcontrolador que ejecuta tus instrucciones al pie de la letra, rapidísimo y sin cansarse. El modelo Arduino Uno, con sus 14 pines digitales, será nuestro cerebro durante todo el curso. Los programas se llaman sketches y siempre tienen dos funciones obligatorias: `setup()`, que se ejecuta una sola vez para configurar todo, y `loop()`, que se repite para siempre con la lógica principal.

El IDE de Arduino es el programa gratuito donde escribes, verificas y subes tus sketches. Aprendiste a instalarlo, a seleccionar la placa Arduino Uno y el puerto USB correcto (el error mas común de los principiantes), y el ciclo de trabajo: escribir, verificar, corregir, subir, observar y ajustar.

Los pines digitales solo manejan dos estados: HIGH (5 voltios) y LOW (0 voltios). Con `pinMode()` declaras si un pin será OUTPUT o INPUT (o `INPUT_PULLUP`). Con `digitalWrite()` pones una salida en HIGH o LOW para encender LEDs y actuadores. Con `digitalRead()` lees una entrada para saber si un botón esta presionado. `INPUT_PULLUP` activa la resistencia interna del Arduino y simplifica los botones: sin presionar lees HIGH, presionado lees LOW, sin necesidad de resistencias externas.

Para manejar el tiempo tienes dos herramientas: `delay()`, que pausa todo durante los milisegundos indicados (simple pero bloqueante), y `millis()`, que consulta el reloj interno sin detener el programa (mas avanzada, ideal cuando el robot debe hacer varias cosas a la vez, usando variables `unsigned long`).

El monitor serie, con `Serial.begin(9600)`, `Serial.print()` y `Serial.println()`, es tu ventana de depuración: te permite ver mensajes y valores del Arduino en tiempo real en tu computadora.

Construiste cuatro proyectos: el parpadeo Blink, el semaforo de tres LEDs, el botón que enciende un LED con `INPUT_PULLUP` y el contador de pulsaciones con detección de flanco en el monitor serie. En la práctica creaste ritmos personalizados, LEDs alternados, un botón que cambia de modo con antirrebote, un contador con reinicio por presión larga usando `millis()` y la señal SOS en código morse con tus propias funciones.

También aprendiste los errores clásicos (puerto equivocado, olvidar `pinMode()`, LED al revés, confundir `==` con `=`, velocidad del monitor serie) y las buenas prácticas (comentar el código, nombres descriptivos, definir pines como constantes, probar en partes pequeñas, guardar versiones).

En la Unidad 3 daremos el siguiente gran salto: moveremos motores y construiremos nuestro primer robot móvil. Todo lo que aprendiste aquí (salidas, entradas, tiempos y depuración) será la base sobre la que rodara ese robot. Nos vemos ahí.

UNIDAD 3: ELECTRÓNICA BÁSICA PARA ROBÓTICA

Introducción

Bienvenido a la tercera unidad del Curso Completo de Robótica de Lucas EGA Academy. En las dos unidades anteriores ya aprendiste que es un robot, cuales son sus partes y como "piensa" un programa que lo controla. Ahora toca un tema que a muchos principiantes les da un poco de miedo, pero que es la base de absolutamente todo lo que vas a construir: la electrónica básica. No te preocupes, no necesitas ser ingeniero, ni saber matemáticas avanzadas, ni haber desarmado nada antes. Esta unidad esta diseñada para alguien que nunca ha tocado un componente electrónico en su vida y que quiere empezar desde cero, con las manos en la masa y con total seguridad.

Vamos a empezar con la pregunta mas obvia: por que un robotista necesita saber electrónica? Después de todo, hoy existen kits que se arman como si fueran legos y placas como el Arduino que simplifican muchísimo el trabajo. La respuesta corta es que los kits te llevan hasta cierto punto, pero la electrónica te libera. Cuando entiendes que pasa realmente con el voltaje, la corriente y los componentes, dejas de ser un simple ensamblador de piezas y te conviertes en un creador. Si un motor no gira, sabes si es por falta de corriente, por un cable mal conectado o por un componente quemado, y lo diagnosticas en minutos en lugar de pasar horas adivinando. Si quieres conectar un sensor que no venía en el kit, lees su hoja de datos, calculas la resistencia que necesita, lo conectas con seguridad y funciona a la primera. Si tu robot se apaga misteriosamente cada vez que los motores arrancan, sabes que el problema es que la fuente no entrega suficiente corriente, y sabes como solucionarlo. Esa independencia es la diferencia entre copiar proyectos de internet y diseñar los tuyos.

Además, la electrónica es el idioma común de la robótica. Un robot no es otra cosa que electrónica, mecánica y programación trabajando juntas. El programa decide, la mecánica ejecuta, pero es la electrónica la que lleva las órdenes del cerebro a los músculos. Cada vez que tu Arduino enciende un LED, esta entregando una pequeña corriente por uno de sus pines. Cada vez que lee un sensor, esta midiendo un voltaje que cambia. Cada motor que gira lo hace porque una corriente lo atraviesa en la dirección correcta. Si no entiendes esos conceptos, la programación se vuelve magia y el hardware se vuelve una caja negra. Con electrónica básica, abres esa caja y ves lo que hay dentro, y entonces todo el curso (sensores, motores, comunicación) se vuelve mucho mas claro.

Otra razón muy práctica: la electrónica te ahorra dinero y frustración. Los componentes básicos cuestan centavos: una resistencia, un LED, un botón. Pero si los conectas mal porque no sabes lo que haces, puedes quemar un Arduino que cuesta mucho mas, o una batería que puede ser peligrosa. Aprender las reglas básicas de seguridad y los cálculos sencillos de esta unidad te protege el bolsillo y te protege a ti. Y no lo decimos para asustarte: todo lo que vamos a hacer en este curso trabaja con voltajes bajos (5 voltios, 9 voltios, 12 voltios como máximo), que son seguros al tacto. El peligro real en este nivel no es que te de un toque, sino que conectes mal algo y quemes componentes o, en el peor de los casos, una batería. Por eso la seguridad es la primera gran sección de esta introducción.

Hablemos de seguridad eléctrica con total claridad. Regla número uno, la mas importante de todo el curso: NUNCA conectes tus circuitos directamente a la red eléctrica de tu casa (110 o 220 voltios, el enchufe de la pared). Todo lo que hagamos en este curso se alimenta con pilas, baterías pequeñas o el puerto USB de tu computadora, que entrega 5 voltios. La corriente de la pared es otra historia completamente distinta y NO tiene lugar en este curso. Si algún día necesitas alimentar un proyecto grande desde la pared, usarías un eliminador o fuente conmutada certificada que convierte la corriente de la pared a voltaje bajo de forma segura, pero eso ya es tema avanzado y siempre con equipo certificado. En esta unidad y en todo el curso básico, la pared ni se toca.

Regla número dos: trabaja siempre SIN energía cuando estés conectando o desconectando componentes. Esto significa: arma tu circuito en el protoboard con la batería desconectada o el Arduino apagado, revisa dos veces las conexiones, y solo entonces conecta la energía. Es el equivalente electrónico de "mira a ambos lados antes de cruzar". Nueve de cada diez componentes quemados se queman por conectar o mover cables con la energía puesta.

Regla número tres: respeta la polaridad. Muchos componentes tienen un lado positivo y un lado negativo (LEDs, capacitores electrolíticos, baterías, el propio Arduino). Conectarlos al revés puede danarlos o hacer que no funcionen. En esta unidad aprenderás a identificar la polaridad de cada componente antes de conectarlo, así que esta regla se vuelve un hábito natural.

Regla número cuatro: no excedas los límites de corriente. Tu Arduino puede entregar como máximo 40 miliamperios por cada pin de salida y 200 miliamperios en total sumando todos los pines. Si le pides mas, el pin se quema. Por eso nunca conectas un motor directamente a un pin del Arduino: los motores piden mucha mas corriente de la que el pin puede dar. En esta unidad aprenderás a calcular cuanta corriente pide cada circuito con la ley de Ohm, así que siempre sabrás si estas dentro de los límites seguros.

Regla número cinco: cuidado con las baterías. Las pilas alcalinas comunes (AA, AAA, de 9V) son muy seguras. Las baterías recargables de litio (las de los celulares, o las LiPo de los drones) guardan mucha energía en poco espacio y hay que tratarlas con respeto: no las perfores, no las aplastes, no las cortocircuites y cargalas solo con su cargador. En este curso básico trabajaremos sobre todo con pilas y con USB, así que no tendrás problemas, pero es bueno que lo sepas desde ahora.

Regla número seis: si algo huele a quemado, se calienta demasiado o echa humo, desconecta la energía de inmediato. No intentes "arreglarlo" con la energía puesta. Desconecta, espera, y luego revisa que hiciste mal. Un componente caliente es la forma que tiene el circuito de decirte "algo esta mal conectado".

Con esas seis reglas ya puedes trabajar con total tranquilidad en todo el curso. Guárdalas en tu memoria porque las vamos a repetir en cada unidad práctica.

Ahora, que vas a aprender a medir y a armar en esta unidad? Vamos a desglosarlo. Primero, vas a entender los tres conceptos fundamentales de la electricidad: voltaje, corriente y resistencia. Usaremos la analogía del agua, que es la forma mas intuitiva de visualizarlos: el voltaje es como la presión del agua en una tubería, la corriente es el caudal (cuanta agua fluye por segundo) y la resistencia es que tan angosta esta la tubería. Con esa imagen mental, la famosa ley de Ohm ($\text{voltaje} = \text{corriente} \times \text{resistencia}$) deja de ser una formula abstracta y se vuelve sentido común. Harás cálculos numéricos reales paso a paso, con los valores que de verdad usamos en robótica: 5 voltios del Arduino, LEDs que consumen 20 miliamperios, resistencias de 220 ohmios.

Segundo, vas a conocer los componentes pasivos mas importantes: las resistencias (y su código de colores, ese sistema de banditas de colores que parece un acertijo pero que dominarás en esta unidad), el protoboard (la tablilla blanca con agujeritos donde se arman los circuitos sin soldar), los LEDs (y por que tienen polaridad), y los pulsadores o botones. También aprenderás a leer lo básico de una hoja de datos, ese documento técnico que acompaña a cada componente y que te dice sus limites y características. Leer hojas de datos es una superpotencia: es lo que separa al que adivina del que diseña.

Tercero, vas a dominar el multímetro, la herramienta mas importante del robotista después de sus manos. Aprenderás a medir voltaje (para saber si tu batería esta cargada o si un pin del Arduino esta entregando los 5V que debería), a medir continuidad (para saber si dos puntos están conectados, util para encontrar cables rotos) y a medir resistencia (para identificar una resistencia sin leer su código de colores). El multímetro es tu "sexto sentido" electrónico: te permite ver lo invisible.

Cuarto, vas a entender las fuentes de poder: que diferencia hay entre alimentar tu proyecto con el USB de la computadora, con pilas, con una batería recargable o con un regulador de voltaje. Cada opción tiene ventajas y límites, y elegir mal la fuente es una de las causas más comunes de robots que "no funcionan". Aprenderás a elegir la fuente correcta para cada proyecto.

Quinto, vas a entender que es un cortocircuito, por que es peligroso y como evitarlo. Un cortocircuito es cuando la corriente encuentra un camino de resistencia casi cero y fluye descontrolada, calentando cables y componentes. Es el accidente más común del principiante y también el más fácil de prevenir una vez que sabes como se ve.

Todo esto corresponde a las clases 7, 8 y 9 del curso: la clase 7 cubre voltaje, corriente, resistencia y ley de Ohm; la clase 8 cubre los componentes pasivos (resistencias, protoboard, LEDs, pulsadores) y la lectura de hojas de datos; la clase 9 cubre herramientas y seguridad (multímetro, fuentes, cortocircuitos y reglas de seguridad).

Una nota sobre matemáticas: solo necesitas saber sumar, restar, multiplicar y dividir. La ley de Ohm es una multiplicación y una división, nada más. Si puedes calcular cuanto cambio te deben en la tienda, puedes hacer todos los cálculos de esta unidad. Y cada calculo lo haremos paso a paso, sin saltarnos nada.

Al terminar esta unidad serás capaz de: explicar con tus palabras que es el voltaje, la corriente y la resistencia; calcular con la ley de Ohm la resistencia que necesita un LED; leer el código de colores de cualquier resistencia; armar un circuito en protoboard sin soldar; identificar la polaridad de un LED; medir voltaje, continuidad y resistencia con el multímetro; elegir la fuente de poder adecuada para un proyecto pequeño; y detectar y evitar cortocircuitos. Con esas habilidades, las siguientes unidades (sensores, motores, Arduino en profundidad) se te harán mucho más fáciles, porque ya hablaras el idioma.

Así que respira hondo, consigue tu protoboard, tus resistencias, tus LEDs y tu multímetro (en la sección de práctica te decimos exactamente que comprar, y no es caro), y vamos a encender tus primeros circuitos. La electrónica, una vez que la entiendes, es adictiva: hay algo mágico en el momento en que conectas unas piezas sueltas y un foquito se enciende porque TU calculaste todo bien. Ese momento esta a unas páginas de distancia. Empecemos.

Explicación detallada

El trío fundamental: voltaje, corriente y resistencia

Toda la electrónica, desde un LED parpadeante hasta el robot mas avanzado, se reduce a tres conceptos que interactúan entre sí: voltaje, corriente y resistencia. Si los entiendes bien, todo lo demás son detalles. Vamos a construirlos uno por uno usando la analogía del agua, que es la mejor forma de "ver" la electricidad, porque la electricidad real es invisible.

Imagina una tubería por la que fluye agua. El VOLTAJE es la presión del agua: que tan fuerte empuja la bomba. Se mide en voltios (V). Una pila AA tiene 1.5 voltios de "presión", el USB de tu computadora entrega 5 voltios, y una batería de 9V (la rectangular) entrega 9 voltios. Mas voltaje significa mas empuje, mas energía disponible para hacer trabajo. En robótica casi todo funciona entre 3.3V y 12V, así que esos numeros te los vas a aprender de memoria muy rápido.

La CORRIENTE es el caudal: cuanta agua pasa por la tubería cada segundo. Se mide en amperios (A), aunque en nuestros circuitos pequeños casi siempre hablamos de miliamperios (mA): mil miliamperios hacen un amperio. Un LED típico consume unos 20 mA, es decir, un chorrito pequeño. Un motor pequeño puede consumir 500 mA o mas, un chorro mucho mas grande. La corriente es lo que realmente "hace el trabajo": es el flujo de electrones moviendose por el cable. Sin corriente no hay luz, no hay movimiento, no hay nada. Pero ojo: la corriente la "pide" el circuito, no la "empuja" la fuente a la fuerza. Una fuente de 5V capaz de entregar 2A no le mete 2A a todo lo que conectes; entrega solo lo que el circuito le pida, hasta su límite. Esto es importantísimo y lo vamos a repetir: la fuente ofrece voltaje, el circuito decide la corriente.

La RESISTENCIA es que tan angosta o rugosa esta la tubería: que tanto se opone al paso del agua. Se mide en ohmios (símbolo: la letra griega omega, pero en texto escribimos "ohm" u "ohmios"). Un cable grueso y corto tiene resistencia casi cero: el agua pasa libre. Una resistencia (el componente) es como un tramo angosto de tubería puesto a propósito para limitar el flujo. En robótica usamos resistencias todo el tiempo para proteger componentes delicados, como los LEDs, que se queman si les llega demasiada corriente.

Con estas tres ideas ya puedes intuir la relación entre ellas: si subes la presión (voltaje) con la misma tubería, fluye mas agua (corriente). Si angostas la tubería (mas resistencia) con la misma presión, fluye menos agua. Esa intuición ES la ley de Ohm, solo falta ponerle numeros.

Antes de la formula, aclaremos dos ideas que confunden a muchos principiantes. Primera: el voltaje siempre se mide ENTRE dos puntos. Decir "aquí hay 5 voltios" significa "entre este punto y la referencia (que llamamos tierra o GND) hay una diferencia de 5 voltios". Es como la altura: decir que algo esta a 10 metros de altura solo tiene sentido respecto al suelo. En electrónica, GND es nuestro "suelo", el punto de referencia de cero voltios. Segunda: la

corriente siempre fluye en un circuito CERRADO, en círculo. Sale del polo positivo de la fuente, atraviesa los componentes haciendo su trabajo, y regresa al polo negativo. Si el círculo esta abierto en algún punto (un cable desconectado, un interruptor apagado), no fluye nada. Por eso se llama "circuito": porque la corriente da la vuelta completa, como en un circuito de carreras.

La ley de Ohm: la formula mas util de la robótica

La ley de Ohm dice algo muy simple: el voltaje (V) es igual a la corriente (I) multiplicada por la resistencia (R). Escrito como formula:

$$V = I \times R$$

Donde V esta en voltios, I en amperios y R en ohmios. La letra I para la corriente viene de la palabra antigua "intensidad", pero no te preocupes por eso; solo recuerda que I = corriente.

De esta formula se despejan las otras dos que también usarás todo el tiempo:

$$\begin{aligned} I &= V / R && (\text{corriente} = \text{voltaje dividido entre resistencia}) \\ R &= V / I && (\text{resistencia} = \text{voltaje dividido entre corriente}) \end{aligned}$$

Un truco para no olvidarlas: dibuja un triángulo con la V arriba y la I y la R abajo lado a lado. Tapas la letra que quieres calcular y lo que queda a la vista es la operación: tapas V y ves I x R; tapas I y ves V / R; tapas R y ves V / I.

Vamos con ejemplos numéricos reales, paso a paso, con valores típicos de robótica. Ejemplo 1: tienes una fuente de 5V (como el USB o el Arduino) y conectas una resistencia de 250 ohmios entre sus terminales. Cuanta corriente fluye? Usamos $I = V / R$. Sustituimos: $I = 5 / 250 = 0.02$ amperios. Para pasarlo a miliamperios multiplicamos por 1000: $0.02 \times 1000 = 20$ mA. Respuesta: fluyen 20 miliamperios. Fíjate que este es exactamente el consumo típico de un LED, y por eso la resistencia de 220 o 250 ohmios es la compañera clásica del LED con 5V. No es casualidad, es ley de Ohm.

Ejemplo 2: quieres saber que resistencia necesitas para limitar la corriente a 20 mA (0.02 A) con una fuente de 5V. Usamos $R = V / I$. Sustituimos: $R = 5 / 0.02 = 250$ ohmios. Como las resistencias se venden en valores estándar, eliges 220 ohmios (el valor estándar mas cercano por debajo) o 270 ohmios. Con 220 ohmios la corriente será $I = 5 / 220 = 0.0227$ A = 22.7 mA, todavía seguro para el LED. Este calculo, con un pequeño ajuste que veremos en la sección de ejemplos (el LED "se queda" con unos 2V), es exactamente lo que harás cada vez que conectes un LED.

Ejemplo 3: tienes una resistencia de 1000 ohmios (1 kilo-ohmio, se escribe 1k) conectada a 9V. Cuanta corriente pasa? $I = V / R = 9 / 1000 = 0.009 \text{ A} = 9 \text{ mA}$. Poca corriente, porque la resistencia es grande.

Ejemplo 4 (inverso): por una resistencia de 470 ohmios fluyen 10 mA (0.01 A). Que voltaje hay entre sus extremos? $V = I \times R = 0.01 \times 470 = 4.7 \text{ V}$. Este tipo de calculo sirve para verificar mediciones: si mides 4.7V en esa resistencia, todo esta bien.

Ejemplo 5, con unidades que confunden: una resistencia de 4.7 kilo-ohmios (4700 ohmios) con 5V. $I = 5 / 4700 = 0.00106 \text{ A}$. Para pasarlo a mA: $0.00106 \times 1000 = 1.06 \text{ mA}$. Cuando trabajes con kilo-ohmios, un atajo util: si V esta en voltios y R en kilo-ohmios, el resultado de V / R te da directamente miliamperios. Compruébalo: $5 / 4.7 = 1.06 \text{ mA}$. Mismo resultado, menos ceros. Este atajo te ahorrará muchos errores.

Potencia: un concepto extra que vale oro. Cuando la corriente atraviesa una resistencia, parte de la energía se convierte en calor. Esa energía por segundo es la POTENCIA, medida en vatios (W), y se calcula como $P = V \times I$. Ejemplo: en el ejemplo 1, la resistencia de 250 ohmios con 5V y 20 mA disipa $P = 5 \times 0.02 = 0.1 \text{ W}$. Las resistencias comunes soportan 0.25 W (un cuarto de vatio), así que 0.1 W esta bien. Si el calculo te diera mas de 0.25 W, necesitarías una resistencia de mayor potencia (de 0.5W o 1W), porque si no se calentara y podría quemarse. En nuestros circuitos de 5V con LEDs esto casi nunca es problema, pero cuando trabajes con motores o tiras de LEDs grandes, la potencia se vuelve crítica. Regla práctica: calcula siempre $P = V \times I$ en la resistencia y verifica que no pase de lo que la resistencia aguanta.

Circuitos en serie y en paralelo

Los componentes se pueden conectar de dos formas básicas, y todo circuito es una combinación de ellas. En SERIE, los componentes van uno detrás de otro, como vagones de un tren: la corriente tiene un solo camino y atraviesa cada componente en fila. En PARALELO, los componentes van lado a lado, como carriles de una autopista: la corriente se reparte entre los caminos.

En serie, tres reglas: (1) la corriente es LA MISMA en todos los componentes (el agua que entra por un extremo sale por el otro, no tiene a donde mas ir); (2) el voltaje SE REPARTE entre los componentes (la presión total se divide entre los tramos angostos); (3) la resistencia total es la SUMA de las resistencias: $R_{\text{total}} = R1 + R2 + R3$. Ejemplo: dos resistencias de 100 ohmios en serie con 5V. $R_{\text{total}} = 200 \text{ ohmios}$. Corriente: $I = 5 / 200 = 0.025 \text{ A} = 25 \text{ mA}$, la misma en ambas. Voltaje en cada una: $V = I \times R = 0.025 \times 100 = 2.5 \text{ V}$ en cada una; $2.5 + 2.5 = 5\text{V}$, cuadra perfecto. Aplicación típica en robótica: un LED en serie con su resistencia

limitadora. El LED y la resistencia van en fila, la misma corriente los atraviesa a ambos, y el voltaje se reparte entre ellos.

En paralelo, las reglas se invierten: (1) el voltaje es EL MISMO en todos los componentes (todos están conectados entre los mismos dos puntos, como varias mangueras conectadas a la misma llave); (2) la corriente SE REPARTE entre las ramas (cada rama pide lo suyo según su resistencia); (3) la resistencia total DISMINUYE: $1/R_{\text{total}} = 1/R1 + 1/R2 + 1/R3$. Ejemplo: dos resistencias de 100 ohmios en paralelo con 5V. Cada rama tiene 5V completos, así que por cada una fluyen $I = 5 / 100 = 50 \text{ mA}$. La corriente total que entrega la fuente es $50 + 50 = 100 \text{ mA}$. La resistencia total: $1/R = 1/100 + 1/100 = 2/100$, así que $R = 50 \text{ ohmios}$. Comprueba: $I_{\text{total}} = 5 / 50 = 100 \text{ mA}$. Cuadra. Aplicación típica: cuando conectas varios sensores o varios LEDs (cada uno con su resistencia) a los 5V del Arduino, todos van en paralelo: cada uno recibe sus 5V y pide su propia corriente, y la fuente debe poder entregar la suma.

Un error clásico del principiante: conectar dos LEDs en serie a 5V esperando que brillen igual. Cada LED necesita unos 2V, así que dos en serie necesitan 4V y queda 1V para la resistencia: brillan poco. Tres LEDs en serie necesitarían 6V, mas de los 5V disponibles: no encienden bien. Moraleja: con 5V, los LEDs van en paralelo (cada uno con su resistencia), no en serie.

Resistencias y el código de colores

La resistencia es el componente mas común de la electrónica y también el mas barato. Su trabajo es oponerse al paso de la corriente de forma controlada. Físicamente es un pequeño cilindro con dos patitas (terminales) y unas banditas de colores pintadas. Esas banditas son el CÓDIGO DE COLORES, un sistema universal para leer su valor sin necesidad de un multímetro.

Las resistencias comunes tienen 4 bandas. Las tres primeras te dan el valor y la cuarta te da la tolerancia (que tan exacto es el valor real). El sistema: cada color representa un dígito (banda 1 y 2) o un multiplicador (banda 3, cuántos ceros agregar). Los colores y sus dígitos son: negro = 0, café (marrón) = 1, rojo = 2, naranja = 3, amarillo = 4, verde = 5, azul = 6, violeta = 7, gris = 8, blanco = 9. Un truco para memorizarlos en orden (0 a 9) es una frase; en México muchos estudiantes usan: "Negro Café Rojo Naranja Amarillo Verde Azul Violeta Gris Blanco". Aprendetela de memoria, la usarás toda la vida.

La banda 4 (tolerancia) casi siempre es dorada o plateada: dorado = 5% de tolerancia, plateado = 10%. Significa que una resistencia de 220 ohmios con banda dorada puede valer en realidad entre 209 y 231 ohmios, y esta bien: para nuestros propósitos esa variación no importa.

Como se lee? Paso 1: identifica la banda de tolerancia (dorada o plateada); esa va a la DERECHA. Las bandas se leen de izquierda a derecha desde el extremo opuesto. Paso 2: toma el dígito de la banda 1 y el dígito de la banda 2, ponlos juntos como un número de dos cifras. Paso 3: la banda 3 es el multiplicador: agrega esa cantidad de ceros al número. Ejemplo completo: bandas café, negro, rojo, dorado. Café = 1, negro = 0, juntos forman "10". Rojo = 2, agrega dos ceros: $10 + "00" = 1000$ ohmios = 1k. Tolerancia dorada = 5%. Respuesta: 1000 ohmios al 5%.

Mas ejemplos rápidos: rojo, rojo, café, dorado = 22 + un cero = 220 ohmios (la clásica del LED). Amarillo, violeta, café, dorado = 47 + un cero = 470 ohmios. Café, negro, naranja, dorado = 10 + tres ceros = 10,000 ohmios = 10k (muy usada con botones y sensores). Verde, azul, rojo, dorado = 56 + dos ceros = 5600 = 5.6k. Naranja, naranja, rojo, dorado = 33 + dos ceros = 3300 = 3.3k.

Casos especiales de la banda 3: si es dorada, el multiplicador es 0.1 (divide entre 10); si es plateada, es 0.01 (divide entre 100). Ejemplo: verde, azul, dorado, dorado = $56 \times 0.1 = 5.6$ ohmios. Son valores bajos que casi no usarás en este curso, pero ya sabes leerlos.

Valores estándar: las resistencias no se fabrican en todos los valores posibles, sino en series estándar (la mas común es la serie E12: 10, 12, 15, 18, 22, 27, 33, 39, 47, 56, 68, 82, y esos mismos numeros por 10, por 100, por 1000...). Por eso cuando calculas 250 ohmios compras 220 o 270. No busques el valor exacto calculado; elige el estándar mas cercano que mantenga el circuito seguro (en el caso del LED, elige el valor igual o mayor al calculado para no exceder la corriente).

También existen resistencias de 5 bandas (mas precisión): las tres primeras son dígitos, la cuarta es el multiplicador y la quinta la tolerancia. Si ves una resistencia con 5 bandas, el principio es el mismo, solo agregas un dígito mas. En tu kit de principiante casi todas serán de 4 bandas.

El protoboard: tu mesa de trabajo sin soldadura

El protoboard (también llamado breadboard o placa de pruebas) es una tablilla de plástico blanco llena de agujeritos donde puedes armar circuitos metiendo las patitas de los componentes y los cables, sin soldar nada. Es la herramienta de prototipado mas importante para un principiante: te permite armar, probar, desarmar y rearmar en minutos. Todo lo que hagas en este curso antes de un montaje final pasará por el protoboard.

Lo esencial es entender como están conectados los agujeros POR DENTRO, porque a simple vista todos se ven iguales. Un protoboard típico tiene tres zonas. Al centro hay dos bloques grandes de agujeros organizados en filas de 5: en cada fila de 5 agujeros, los 5 están

conectados entre si por una laminita metalica interna. Es decir, si metes una patita en el agujero 1 de la fila y otra patita en el agujero 3 de la misma fila, están conectados como si hubiera un cable entre ellos. Pero la fila de arriba NO esta conectada con la fila de abajo: cada fila de 5 es independiente. En medio del protoboard hay un canal (una ranura) que divide el bloque central en dos mitades: sirve para montar chips con patitas a cada lado sin que se toquen.

A los lados del protoboard hay unas líneas largas marcadas con + (rojo) y - (azul): son los rieles de alimentación. Todos los agujeros de un riel están conectados entre si a lo largo de TODA la tablilla (en algunos protoboards grandes el riel esta cortado a la mitad, revisa que haya continuidad). La idea es: conectas el positivo de tu fuente al riel rojo y el negativo (GND) al riel azul, y desde ahí repartes energía a cualquier punto del circuito con cables cortos. Es como tener una regleta de enchufes para voltaje.

Diagrama de la estructura interna (vista desde arriba, simplificada):

```

riel +  o---o---o---o---o---o---o---o---o---o--- (todos conectados)
riel -  o---o---o---o---o---o---o---o---o---o--- (todos conectados)

      a b c d e   f g h i j   <- columns
+---+---+---+---+---+
1 | o   o   o   o   o |   <- fila 1: a-b-c-d-e conectados
+---+---+---+---+---+
2 | o   o   o   o   o |   <- fila 2: independiente de la fila 1
+---+---+---+---+---+
      CANAL
3 | o   o   o   o   o |   <- del otro lado del canal
+---+---+---+---+---+

```

Reglas de oro del protoboard: (1) nunca conectes el riel + con el riel - directamente con un cable: eso es un cortocircuito; (2) los componentes se conectan "puenteando" entre filas distintas: una patita en una fila, la otra patita en otra fila, y usas cables para unir filas; (3) revisa que las patitas hagan buen contacto: a veces un LED no enciende solo porque una patita quedó floja; (4) los protoboards se desgastan con el uso; si un agujero ya no aprieta, usa otro.

LEDs: luz con polaridad

El LED (diodo emisor de luz) es el componente favorito de los principiantes porque da retroalimentación inmediata: lo conectas bien y se enciende. Pero tiene dos particularidades que debes respetar. Primera: tiene POLARIDAD, es decir, un lado positivo (anodo, la patita LARGA) y un lado negativo (catodo, la patita CORTA). Si lo conectas al revés, no enciende (y

si el voltaje es alto, se puede dañar). Como identificar las patitas? La patita larga es el anodo (+). Si ya cortaste las patitas y no sabes cual era la larga, mira dentro del plástico: el electrodo mas grande (como una copita) es el catodo (-), y el mas pequeño es el anodo (+). Además, muchos LEDs tienen un lado plano en el plástico, del lado del catodo (-). Cuando dudes, el multímetro en modo diodo te saca de dudas (lo verás en la sección de ejemplos).

Segunda particularidad: el LED necesita una resistencia en serie SIEMPRE. Un LED típico (rojo, verde, amarillo) tiene una caída de voltaje de unos 2V y una corriente máxima de unos 20 mA. Si lo conectas directo a 5V sin resistencia, la corriente se dispara (el LED casi no ofrece resistencia) y se quema en cuestion de segundos, a veces con un pequeño "pop" y un olor característico. La resistencia limita la corriente a un valor seguro. El calculo exacto lo harás en la sección de ejemplos paso a paso, pero la receta rápida es: con 5V, una resistencia de 220 ohmios en serie con un LED normal funciona perfecto.

Colores y voltajes: los LEDs rojos, verdes y amarillos "se quedan" con unos 2V; los azules y blancos, con unos 3V a 3.3V. Esto importa para el calculo de la resistencia, como verás en los ejemplos.

El multímetro: tus ojos para ver la electricidad

El multímetro es un aparato con una pantalla, una perilla selectora y dos puntas (roja y negra). Con el puedes medir voltaje, corriente, resistencia y continuidad, que son las cuatro mediciones que usarás en robótica. Un multímetro digital básico cuesta poco y te va a durar años; es la mejor inversión después del Arduino.

Las puntas: la negra va en el conector marcado COM (común, la referencia) y la roja va en el conector marcado V-ohm (para medir voltaje y resistencia). Hay un tercer conector para medir corriente alta, pero en este curso casi no lo usarás, y medir corriente requiere interrumpir el circuito, así que lo dejamos para después.

Medir VOLTAJE: pon la perilla en V con línea continua (voltaje de corriente directa, DC; no uses el de la ondita, que es corriente alterna). Toca con la punta negra el punto de referencia (GND) y con la roja el punto que quieres medir. La pantalla muestra la diferencia. Ejemplo: puntas en los rieles + y - del protoboard alimentado por USB: debe decir unos 5V. Si dice -5V, tienes las puntas al revés (no pasa nada, solo cambia el signo). Medir voltaje se hace EN PARALELO: las puntas tocan los dos puntos sin desconectar nada, el circuito sigue funcionando. Es una medicion segura y no invasiva.

Medir CONTINUIDAD: pon la perilla en el símbolo de la ondita sonora o el diodo. Toca dos puntos con las puntas: si están conectados (casi cero resistencia), el multímetro PITA. Sirve para verificar que un cable no este roto por dentro, que dos agujeros del protoboard estén

conectados, o para rastrear a donde llega una pista. Es la función que mas usarás para diagnosticar fallas: "este cable debería llevar 5V hasta aquí, veamos si hay continuidad".

Medir RESISTENCIA: pon la perilla en ohmios. IMPORTANTE: mide la resistencia SIN energía en el circuito (desconecta la fuente) y con la resistencia fuera del circuito o al menos con una patita desconectada, porque si esta conectada a otros componentes la lectura sale mal. Toca cada patita con una punta; no importa cual punta en cual patita. La pantalla muestra el valor. Sirve para identificar resistencias sin leer el código de colores.

Medir CORRIENTE: es la medicion mas delicada porque hay que ABRIR el circuito e intercalar el multímetro en serie (la corriente debe pasar POR el multímetro). Además hay que cambiar la punta roja al conector de corriente. Si lo haces mal (por ejemplo, mides corriente con las puntas en paralelo como si fuera voltaje), provocas un cortocircuito a través del multímetro y quemas su fusible. Consejo de esta unidad: para saber la corriente de tus circuitos, CALCULALA con la ley de Ohm en lugar de medirla. Es mas rápido y mas seguro. Deja la medicion de corriente para cuando tengas mas experiencia.

Fuentes de poder: de donde sale la energía

Todo robot necesita energía, y elegir la fuente correcta es una decisión de diseño. Estas son las opciones que usarás, de la mas simple a la mas capaz:

USB de la computadora (5V, hasta 500 mA): es la fuente por defecto cuando programas el Arduino. Ventajas: siempre disponible, voltaje estable de 5V, segura. Limites: solo 500 mA, así que no alcanza para motores (un motor al arrancar puede pedir mas y la computadora se protege cortando el puerto). Regla: mientras solo haya LEDs, sensores y el Arduino, el USB basta. En cuanto agregues motores, necesitas otra fuente.

Pilas alcalinas AA (1.5V cada una): baratas y seguras. En serie se suman los voltajes: 4 pilas AA = 6V, 6 pilas = 9V. Un portapilas de 4 AA es la fuente clásica de los robots pequeños. Ojo: las pilas se agotan y su voltaje baja con el uso (unas pilas gastadas dan 4.5V en lugar de 6V), así que si tu robot "se vuelve loco" de repente, lo primero que se revisa son las pilas. Las pilas en serie suman voltaje pero la corriente máxima sigue siendo la de una pila.

Batería de 9V (la rectangular): cómoda por su forma y su conector, entrega 9V. Pero tiene poca capacidad: se agota rápido con motores. Sirve para circuitos de LEDs y pruebas, no para robots con motores que trabajen mucho.

Baterías recargables de litio (LiPo, 18650): mucha energía en poco peso, son las que usan los drones y robots serios. Una celda da 3.7V nominales. Requieren cargador específico y respeto (no perforar, no cortocircuitar). Para este curso básico no las necesitas; las mencionamos para que las reconozcas cuando avances.

Reguladores de voltaje: a veces tu fuente da mas voltaje del que tu circuito necesita. Ejemplo: tienes una batería de 9V pero tu Arduino y sensores trabajan a 5V. Un regulador (como el 7805 o un módulo reductor) convierte los 9V en 5V estables. El Arduino ya trae un regulador interno: puedes alimentarlo con 7 a 12V por su conector de barril o el pin VIN y el solito entrega 5V regulados. Pero ese regulador interno se calienta si le pides mucha corriente; para proyectos con varios servos o sensores se usa un regulador externo o una fuente de 5V directa.

Como elegir: suma la corriente que piden tus componentes (LEDs, sensores, Arduino) y verifica que la fuente la pueda entregar con margen. Ejemplo: Arduino (50 mA) + 3 LEDs (20 mA cada uno = 60 mA) + sensor ultrasónico (15 mA) = 125 mA. El USB (500 mA) sobra. Si agregas 2 motores de 400 mA cada uno, el total sube a mas de 900 mA: el USB ya no alcanza y necesitas pilas o batería con un driver de motores. Esa suma de corrientes es un hábito que te evitará la mitad de los problemas.

Cortocircuitos: que son y como evitarlos

Un cortocircuito ocurre cuando la corriente encuentra un camino de resistencia casi cero entre el positivo y el negativo de la fuente, saltandose los componentes. Recuerda la ley de Ohm: $I = V / R$. Si R es casi cero, la corriente se vuelve ENORME. Con 5V y una resistencia de 0.1 ohmios (un cable), $I = 5 / 0.1 = 50$ A. Ninguna fuente pequeña entrega eso: las pilas se calientan, los cables se calientan, el Arduino se quema, y una batería de litio puede hasta incendiarse. Por eso el cortocircuito es el accidente número uno del principiante.

Como se ve un cortocircuito típico? El mas común en protoboard: conectar por accidente el riel + con el riel - mediante un cable o mediante las patitas de un componente que tocan donde no deben. Otro común: dos cables pelados que se tocan. Otro: montar un componente "a caballo" entre dos filas que resultan estar conectadas a + y - . Síntomas: la fuente se calienta, el voltaje medido cae a casi cero, algo huele a quemado, o el Arduino se apaga y se calienta.

Como evitarlos, en 5 hábitos: (1) arma SIEMPRE sin energía y conecta la fuente al final; (2) antes de energizar, revisa visualmente: ningún cable une + con - directamente, ninguna patita toca a su vecina sin querer; (3) mantén los cables ordenados y del largo justo; los cables largos y enredados se tocan entre si; (4) cuando termines de armar, mide con el multímetro en continuidad entre el riel + y el riel - ANTES de conectar la fuente: si pita, hay un corto, búscalo y corrígelo sin energía; (5) si algo se calienta o huele mal, desconecta de inmediato y revisa.

Errores comunes y buenas prácticas

Errores comunes del principiante (todos los hemos cometido): conectar el LED sin resistencia ("solo para probar rápido") y quemarlo; conectar el LED al revés y pensar que "el LED no sirve"; leer mal el código de colores (confundir la banda de tolerancia con la primera banda) y poner una resistencia de 2.2k en lugar de 220; medir resistencia con el circuito energizado y obtener lecturas locas; alimentar motores desde los pines del Arduino en lugar de una fuente externa; usar cables tan largos que se convierten en un nido; y el campeón: mover cables en el protoboard con la energía conectada y provocar un corto.

Buenas prácticas que te harán la vida fácil: usa código de colores en tus cables (rojo para positivo, negro para GND, otros colores para señales); arma por secciones y prueba cada sección antes de seguir; ten siempre el multímetro a la mano; calcula antes de conectar (ley de Ohm primero, cables después); documenta tus circuitos con un dibujo o foto antes de desarmarlos; y cuando algo no funcione, no cambies cinco cosas a la vez: cambia una, prueba, y así aislas el problema. Esa disciplina de diagnóstico vale mas que cualquier componente.

Ejemplos paso a paso

Ejemplo A: calcular la resistencia para un LED con la ley de Ohm

Este es EL calculo mas importante de la unidad. Lo harás decenas de veces en tu vida de robotista, así que vamos a hacerlo lento y completo, sin saltarnos ningún paso.

El problema: tienes una fuente de 5V (el Arduino o el USB) y un LED rojo estándar. Quieres que el LED brille bien pero sin quemarse. Que resistencia debes poner en serie con el?

Datos que necesitamos (de la hoja de datos típica de un LED rojo, o de memoria): voltaje del LED (V_f) = 2V, corriente deseada (I) = 20 mA = 0.02 A, voltaje de la fuente (V_{fuente}) = 5V.

Paso 1: entiende que el voltaje se reparte. El LED y la resistencia van en serie, así que entre los dos deben "repartirse" los 5V de la fuente. El LED se queda con sus 2V (es su naturaleza,

siempre que conduzca). Lo que sobra le toca a la resistencia: $V_{\text{resistencia}} = V_{\text{fuente}} - V_{\text{LED}} = 5 - 2 = 3\text{V}$. La resistencia tiene que "absorber" 3 voltios.

Paso 2: la corriente es la misma en todo el circuito serie. Queremos 20 mA (0.02 A) por el LED, así que por la resistencia también pasan 20 mA. Ya tenemos los dos datos de la resistencia: entre sus extremos hay 3V y por ella pasan 0.02 A.

Paso 3: aplica la ley de Ohm para la resistencia: $R = V / I = 3 / 0.02 = 150$ ohmios.

Paso 4: elige el valor estándar mas cercano POR ARRIBA (para no exceder la corriente): 150 ohmios no es estándar en la serie E12; los vecinos son 120 y 180. Elegimos 180 ohmios. Verificación: con 180 ohmios, la corriente real será $I = V / R = 3 / 180 = 0.0167 \text{ A} = 16.7 \text{ mA}$. El LED brillara un poco menos que al máximo pero totalmente bien y con margen de seguridad. También puedes usar 220 ohmios (el clásico): $I = 3 / 220 = 13.6 \text{ mA}$, brillo un poco menor pero aun perfectamente visible. Ambos son correctos; 180 da mas brillo, 220 da mas margen.

Paso 5: verifica la potencia de la resistencia: $P = V \times I = 3 \times 0.0167 = 0.05 \text{ W}$. Una resistencia común de 0.25 W sobra. Listo.

Formula general que acabas de usar (aprendetela): $R = (V_{\text{fuente}} - V_{\text{LED}}) / I$. Apliquemosla a otros casos para practicar: LED azul ($V_f = 3\text{V}$) con 5V y 20 mA: $R = (5 - 3) / 0.02 = 100$ ohmios; estándar cercano por arriba: 120 ohmios. LED rojo con batería de 9V y 20 mA: $R = (9 - 2) / 0.02 = 350$ ohmios; estándar: 390 ohmios. LED rojo con 3.3V (algunas placas) y 15 mA: $R = (3.3 - 2) / 0.015 = 86.7$ ohmios; estándar: 100 ohmios. Ya sabes calcularlo para cualquier combinación.

Ejemplo B: armar el circuito del LED en el protoboard

Ahora llevamos el calculo a la realidad. Materiales: 1 protoboard, 1 LED rojo, 1 resistencia de 220 ohmios (rojo-rojo-café-dorado), 2 cables, fuente de 5V (puede ser el Arduino conectado por USB, usando sus pines 5V y GND).

Esquema del circuito (vista textual):

```
+5V ----[ R 220 ]-----+-----[ LED > ]---- GND
                        |
                    (anodo +)  (catodo -)
```

Diagrama en protoboard (vista desde arriba). Usaremos el riel rojo (+) y el riel azul (-) para la alimentación:

```
riel + (rojo)  o===o===o===o===o===o===o===o    <- conecta aqui el 5V
riel - (azul)  o===o===o===o===o===o===o===o    <- conecta aqui GND

    fila 10:  [R] patita 1 en a10, patita 2 en a12
    fila 12:  cable de a12 (misma fila que patita 2 de R) ... espera,
              mejor: LED anodo (pata larga) en b12, catodo (pata corta) en b15
    fila 15:  cable de c15 al riel azul (GND)
    cable del riel rojo (+) a d10 (misma fila que patita 1 de R)
```

Pasos numerados, sin energía conectada todavía:

1. Desconecta toda fuente de energía. Protoboard vacío sobre la mesa.
2. Identifica la resistencia de 220 ohmios por su código de colores: rojo, rojo, café, dorado. Verificala con el multímetro si quieres (debe leer cerca de 220).
3. Mete una patita de la resistencia en el agujero a10 y la otra en el agujero a14 (dos filas distintas; la resistencia "puentea" entre la fila 10 y la fila 14).
4. Identifica el LED: patita LARGA = anodo (+), patita CORTA = catodo (-).
5. Mete la patita larga del LED en el agujero b14 (la MISMA fila 14 donde esta la segunda patita de la resistencia: así quedan conectados en serie) y la patita corta en el agujero b18 (otra fila distinta).
6. Con un cable, une el agujero c10 (fila 10, misma que la primera patita de la resistencia) con el riel rojo (+).
7. Con otro cable, une el agujero c18 (fila 18, misma que la patita corta del LED) con el riel azul (- / GND).
8. Revisión de seguridad SIN energía: verifica que ningún cable una directamente el riel rojo con el riel azul; verifica que la patita larga del LED este del lado de la resistencia (hacia el positivo) y la corta hacia GND; verifica que cada patita este bien metida.
9. Prueba de continuidad (opcional pero recomendada): con el multímetro en modo continuidad, toca el riel rojo y el riel azul: NO debe pitar. Si pita, hay un cortocircuito: revisa antes de energizar.
10. Ahora si: conecta el 5V al riel rojo y GND al riel azul (desde el Arduino por USB, por ejemplo). El LED debe encenderse de inmediato.

Si no enciende, diagnóstico en orden: (1) esta la fuente conectada y encendida? (2) el LED esta al revés? (prueba volteandolo, con la energía DESCONECTADA); (3) alguna patita floja? empuja suavemente cada componente; (4) la resistencia es la correcta? (5) mide con el multímetro el voltaje entre el riel rojo y el azul: hay 5V? Sigue este orden siempre, del problema mas tonto al mas complejo.

Ejemplo C: medir voltaje con el multímetro

Vamos a medir el circuito del Ejemplo B para verificar que todo esta como lo calculamos. El circuito esta energizado con 5V y el LED encendido.

Paso 1: prepara el multímetro. Punta negra en el conector COM, punta roja en el conector V-ohm. Gira la perilla a voltaje DC (V con línea continua; en muchos multímetros es "20V" dentro de la zona DCV: elige el rango inmediato superior a lo que esperas medir, 20V para medir 5V).

Paso 2: mide la fuente. Toca con la punta negra el riel azul (GND) y con la roja el riel rojo (+). La pantalla debe mostrar entre 4.8 y 5.2V (el USB no siempre da exactamente 5.0). Anota el valor real, digamos 5.0V.

Paso 3: mide el voltaje en el LED. Sin desconectar nada, toca con la punta roja la fila del anodo (patita larga, fila 14) y con la negra la fila del catodo (patita corta, fila 18). Debe leer cerca de 2.0V. Eso confirma que el LED esta conduciendo normal.

Paso 4: mide el voltaje en la resistencia. Punta roja en la fila 10 (lado del positivo), punta negra en la fila 14 (lado del LED). Debe leer cerca de 3.0V. Comprueba: 2.0V (LED) + 3.0V (resistencia) = 5.0V (fuente). La suma cuadra porque en serie el voltaje se reparte. Si mides 5V en la resistencia y 0V en el LED, el LED esta abierto (quemado o al revés). Si mides 0V en ambas, no llega energía.

Paso 5: calcula la corriente real sin medirla directamente: $I = V_{\text{resistencia}} / R = 3.0 / 220 = 0.0136 \text{ A} = 13.6 \text{ mA}$. Todo consistente con el diseño. Así se verifica un circuito: mides voltajes, calculas corrientes, comparas con lo esperado.

Ejemplo D: divisor de voltaje explicado y armado

El divisor de voltaje es un truco fundamental: con dos resistencias en serie puedes obtener un voltaje menor a partir de uno mayor. Se usa muchísimo, por ejemplo para conectar un sensor de 5V a una entrada de 3.3V, o para leer una batería de 9V con el Arduino sin danarlo.

Como funciona: dos resistencias R1 y R2 en serie entre el voltaje de entrada (Vin) y GND. La corriente que pasa por ambas es la misma: $I = V_{in} / (R1 + R2)$. El voltaje en R2 (medido entre la union de ambas y GND) es $V_{out} = I \times R2$. Sustituyendo: $V_{out} = V_{in} \times R2 / (R1 + R2)$. Esa es la formula del divisor.

Ejemplo numérico: quieres obtener 3.3V a partir de 5V. Elige R1 = 1000 (1k) y R2 = 2000 (2k). $V_{out} = 5 \times 2000 / (1000 + 2000) = 5 \times 2000 / 3000 = 5 \times 0.667 = 3.33V$. Perfecto. La corriente que consume el divisor: $I = 5 / 3000 = 1.67 \text{ mA}$, un consumo pequeño y constante.

Otro ejemplo: leer una batería de 9V con una entrada analógica del Arduino que soporta máximo 5V. Elige R1 = 10k y R2 = 10k: $V_{out} = 9 \times 10000 / 20000 = 4.5V$. Seguro para el Arduino, y conociendo la formula puedes calcular hacia atrás el voltaje real de la batería desde lo que lee el Arduino.

Armado en protoboard (R1 = 1k entre Vin y el punto medio, R2 = 2k entre el punto medio y GND):

```
Vin (5V)
|
[R1 1k]
|
+-----> Vout (3.3V, al pin analogico del Arduino)
|
[R2 2k]
|
GND
```

Pasos: 1) sin energía, mete R1 entre la fila 20 (a20) y la fila 22 (a22); 2) mete R2 entre la fila 22 (b22, misma fila que R1) y la fila 26 (b26); 3) cable de la fila 20 al riel rojo (5V); 4) cable de la fila 26 al riel azul (GND); 5) el punto Vout es la fila 22: de ahí sale un cable a la entrada que quieras alimentar o medir; 6) revisa que el riel rojo y el azul no estén unidos; 7) energiza y mide Vout con el multímetro entre la fila 22 y GND: debe dar cerca de 3.3V.

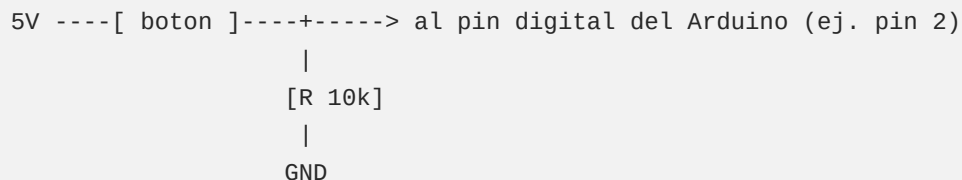
Advertencia importante: el divisor solo funciona bien si lo que conectas a Vout consume MUY poca corriente (como una entrada del Arduino, que casi no consume). Si conectas algo que pide mucha corriente (un motor, varios LEDs), el divisor se "desbalancea" y el voltaje cae. Para alimentar cosas que consumen de verdad se usa un regulador, no un divisor. El divisor es para señales y mediciones, no para dar potencia.

Ejemplo E: circuito de botón con resistencia pull-down

Un pulsador (botón) parece el componente mas simple del mundo, pero conectarlo mal al Arduino es un error clásico. El problema: un pin de entrada del Arduino configurado como INPUT "flota" si no esta conectado a nada definido: capta ruido eléctrico del ambiente y lee unos y ceros al azar. La solución es la resistencia PULL-DOWN (o pull-up): una resistencia que "jala" el pin a un valor conocido (0V o 5V) cuando el botón NO esta presionado.

El circuito pull-down: el botón conecta el pin a 5V cuando lo presionas; una resistencia de 10k conecta el pin a GND permanentemente. Sin presionar: el pin esta unido a GND por la resistencia de 10k, lee 0 (LOW). Al presionar: el botón une directamente el pin con 5V, lee 1 (HIGH). La resistencia de 10k queda entre 5V y GND mientras presionas, consumiendo $I = 5 / 10000 = 0.5 \text{ mA}$, insignificante, y evita el cortocircuito que habría si el botón uniera 5V con GND sin nada en medio.

Diagrama:



Armado en protoboard: 1) el pulsador típico tiene 4 patitas (dos pares conectados internamente); identificalas: presiona el botón y mide continuidad para saber que par se une al presionar, o simplemente usa dos patitas en diagonal; 2) monta el botón a caballo sobre el canal central del protoboard; 3) una patita del botón a la fila del riel rojo (5V) con un cable; 4) la patita opuesta (la que se conecta al presionar) a una fila libre, digamos fila 30; 5) de la fila 30, un cable al pin 2 del Arduino; 6) de la fila 30, la resistencia de 10k (café-negro-naranja-dorado) a otra fila, digamos fila 34; 7) de la fila 34, un cable al riel azul (GND); 8) revisa sin energía, luego energiza.

Prueba: con el programa mas simple (leer el pin 2 y encender el LED integrado si lee HIGH), al presionar el botón el LED del Arduino se enciende, al soltar se apaga, sin parpadeos raros. Si el pin lee valores locos sin presionar, tu resistencia pull-down no esta bien conectada: revisa los pasos 6 y 7.

Nota: el Arduino tiene resistencias pull-up INTERNAS que se activan por programa (`pinMode(pin, INPUT_PULLUP)`), lo que evita la resistencia externa. Pero entender el pull-down externo te enseña el concepto, y hay situaciones (sensores, otros microcontroladores) donde la necesitarás física. Aprende el concepto aquí y usa el atajo cuando quieras.

Práctica

Lo que sigue son 6 ejercicios para que pongas las manos en la masa. Necesitas: protoboard, resistencias surtidas (220, 1k, 10k), 2 o 3 LEDs, un pulsador, cables, multímetro y una fuente de 5V (Arduino por USB). Cada ejercicio trae su enunciado, su objetivo y su solución explicada. Intenta resolverlo TU primero antes de leer la solución: la solución esta para verificar, no para copiar.

Ejercicio 1: cálculos de ley de Ohm

Enunciado: resuelve los siguientes 4 incisos mostrando cada paso: (a) Que corriente fluye por una resistencia de 470 ohmios conectada a 5V? Expresa el resultado en mA. (b) Que resistencia necesitas para limitar la corriente a 15 mA con una fuente de 9V? Elige el valor estándar E12 mas cercano por arriba. (c) Por una resistencia de 2.2k fluyen 2 mA. Que voltaje hay entre sus extremos? (d) Una resistencia de 330 ohmios tiene 4V entre sus extremos. Cuanta potencia disipa? Es segura una resistencia común de 0.25W?

Objetivo: dominar las tres formas de la ley de Ohm y el calculo de potencia, incluyendo conversiones entre amperios y miliamperios y entre ohmios y kilo-ohmios.

Solución del ejercicio 1

(a) $I = V / R = 5 / 470 = 0.0106 \text{ A}$. A miliamperios: $0.0106 \times 1000 = 10.6 \text{ mA}$.

(b) $R = V / I = 9 / 0.015 = 600 \text{ ohmios}$. Valores E12 cercanos: 560 y 680. Elegimos 680 (por arriba, para no exceder los 15 mA). Verificación: $I = 9 / 680 = 13.2 \text{ mA}$, dentro de lo seguro.

(c) Atajo: $V \text{ en voltios} = I \text{ en mA} \times R \text{ en kilo-ohmios} = 2 \times 2.2 = 4.4\text{V}$. (Comprobación larga: $I = 0.002 \text{ A}$, $R = 2200$, $V = 0.002 \times 2200 = 4.4\text{V}$.)

(d) Primero la corriente: $I = V / R = 4 / 330 = 0.0121 \text{ A} = 12.1 \text{ mA}$. Potencia: $P = V \times I = 4 \times 0.0121 = 0.048 \text{ W}$. Es menor que 0.25W, así que una resistencia común esta perfectamente segura (trabaja a menos del 20% de su capacidad).

Ejercicio 2: código de colores

Enunciado: determina el valor (en ohmios y en kilo-ohmios cuando aplique) y la tolerancia de estas resistencias: (a) café, negro, rojo, dorado; (b) rojo, rojo, café, dorado; (c) amarillo, violeta, naranja, dorado; (d) verde, azul, café, dorado; (e) naranja, naranja, rojo, plateado. Además: cual de las cinco usarías para un LED rojo con fuente de 5V buscando unos 15 mA, y por que?

Objetivo: leer el código de colores con fluidez y conectar ese conocimiento con el calculo del Ejercicio 1.

Solución del ejercicio 2

Recordatorio de dígitos: negro 0, café 1, rojo 2, naranja 3, amarillo 4, verde 5, azul 6.

(a) café(1), negro(0) = "10"; rojo = 2 ceros: 1000 ohmios = 1k; dorado = 5%. (b) rojo(2), rojo(2) = "22"; café = 1 cero: 220 ohmios; dorado = 5%. (c) amarillo(4), violeta(7) = "47"; naranja = 3 ceros: 47000 = 47k; dorado = 5%. (d) verde(5), azul(6) = "56"; café = 1 cero: 560 ohmios; dorado = 5%. (e) naranja(3), naranja(3) = "33"; rojo = 2 ceros: 3300 = 3.3k; plateado = 10%.

Para el LED rojo a 5V y 15 mA: $R = (5 - 2) / 0.015 = 200$ ohmios. La mas cercana de la lista es la (b) de 220 ohmios (da 13.6 mA, perfecto). La (d) de 560 daría solo 5.4 mA (muy tenue) y la (a) de 1k daría 3 mA (casi apagado).

Ejercicio 3: diseño en papel

Enunciado: dibuja en papel (o describe con texto) el circuito completo de un semaforo de juguete: 3 LEDs (rojo, amarillo, verde), cada uno con su resistencia, todos alimentados por 5V, cada LED con su propio interruptor (puedes dibujar los interruptores como simples contactos). Calcula la resistencia para cada LED buscando 15 mA (usa $V_f = 2V$ para rojo y amarillo, $V_f = 3V$ para el verde). Calcula además la corriente total cuando los 3 LEDs están encendidos al mismo tiempo. Los LEDs van en serie o en paralelo entre si? Justifica.

Objetivo: aplicar serie/paralelo, el calculo del LED y la suma de corrientes de ramas en paralelo.

Solución del ejercicio 3

Cada LED con su resistencia forma una rama independiente, y las 3 ramas van en PARALELO entre 5V y GND (así cada LED recibe sus 5V completos; en serie no alcanzaria el voltaje). Cada rama lleva su interruptor en serie con su LED y su resistencia.

Cálculos: rojo y amarillo: $R = (5 - 2) / 0.015 = 200$ ohmios -> estándar 220. Verde: $R = (5 - 3) / 0.015 = 133$ ohmios -> estándar 150. Corriente por rama con valores estándar: rojo/amarillo: $3 / 220 = 13.6$ mA; verde: $2 / 150 = 13.3$ mA. Corriente total con los 3 encendidos: $13.6 + 13.6 + 13.3 = 40.5$ mA. El USB (500 mA) lo maneja sin problema. Si hubieras puesto los 3 LEDs en serie, necesitarían $2+2+3 = 7$ V, mas de los 5V disponibles: no encenderían bien. Por eso van en paralelo.

Ejercicio 4: mediciones con el multímetro

Enunciado: arma en el protoboard el circuito del Ejemplo B (LED rojo + resistencia de 220 a 5V). Con el multímetro mide y anota: (a) voltaje de la fuente (rieles + y -); (b) voltaje entre los extremos del LED; (c) voltaje entre los extremos de la resistencia; (d) verifica que (b) + (c) = (a) aproximadamente; (e) con la fuente DESCONECTADA, mide la resistencia (debe dar cerca de 220). Explica que significaría cada uno de estos resultados anómalos: el LED mide 0V y la resistencia mide 5V; la fuente mide 0V; la resistencia mide 150 ohmios en lugar de 220.

Objetivo: practicar medición de voltaje y resistencia, y aprender a interpretar lecturas para diagnosticar.

Solución del ejercicio 4

Valores esperados: (a) ~5V (4.8 a 5.2 según el USB); (b) ~2V en el LED; (c) ~3V en la resistencia; (d) $2 + 3 = 5$, cuadra; la pequeña diferencia con (a) es normal por tolerancias; (e) ~220 ohmios (entre 209 y 231 por la tolerancia del 5%).

Interpretación de anomalías: LED 0V + resistencia 5V significa que por el circuito no pasa corriente pero el voltaje llega: el LED está abierto (quemado) o conectado al revés (no conduce, así que toda la caída de voltaje aparece en la resistencia... en realidad con el LED al revés no hay corriente y la resistencia casi no cae voltaje; si la resistencia mide los 5V completos, el LED está en circuito abierto). Fuente en 0V: la fuente no está conectada, el cable está roto, o hay un cortocircuito que colapsa la fuente (revisa continuidad entre rieles sin energía). Resistencia de 150 en lugar de 220: tomaste la resistencia equivocada del kit (revisa el código de colores); con 150 el LED recibiría 20 mA en lugar de 13.6, aun seguro, pero el ejercicio era verificar que sabes identificar componentes.

Ejercicio 5: divisor de voltaje

Enunciado: necesitas alimentar la entrada analógica de un sensor que soporta máximo 3.3V usando una fuente de 5V. (a) Calcula R1 y R2 para un divisor que entregue 3.3V usando valores estándar E12 y procurando que la corriente del divisor sea de unos 1 a 2 mA. (b)

Ármalo en el protoboard y mide Vout con el multímetro. (c) Explica por que NO debes usar este divisor para alimentar un motor que consume 200 mA.

Objetivo: diseñar un divisor con criterio (voltaje objetivo + corriente de diseño) y entender sus límites.

Solución del ejercicio 5

(a) Queremos $V_{out} = 5 \times R2 / (R1 + R2) = 3.3$, o sea $R2 / (R1 + R2) = 0.66$. Si elegimos $R1 = 1k$ y $R2 = 2k$: $2000/3000 = 0.667$, $V_{out} = 3.33V$. Corriente del divisor: $I = 5 / 3000 = 1.67$ mA, dentro del rango pedido. Ambos son valores E12 estándar. Alternativa valida: $R1 = 2.2k$, $R2 = 4.7k$: $4700/6900 = 0.681$, $V_{out} = 3.4V$ (un poco alto pero aceptable para muchas entradas de 3.3V con tolerancia; mejor la primera opción).

(b) Al medir Vout debes obtener entre 3.2 y 3.4V. Si obtienes 5V, R2 no esta conectada a GND. Si obtienes 0V, R1 no esta conectada a 5V o hay un corto a GND en el punto medio.

(c) El divisor solo mantiene su voltaje si la carga casi no consume corriente. Un motor de 200 mA es 120 veces la corriente del divisor (1.67 mA): el motor "jala" el punto medio hacia abajo, el voltaje colapsa y el motor ni siquiera arranca bien. Además las resistencias disiparian potencia excesiva. Para cargas reales se usa un regulador de voltaje, no un divisor. Regla de oro: el divisor es para señales, el regulador es para potencia.

Ejercicio 6: diagnóstico del circuito que no funciona

Enunciado: un compañero armó el circuito del botón con pull-down del Ejemplo E, pero el Arduino lee HIGH todo el tiempo, este o no presionado el botón. Te da estos datos: el botón esta bien (lo probó con continuidad), el programa esta bien (es el ejemplo básico de lectura digital). Con el multímetro (circuito energizado) mides el voltaje en el pin del Arduino respecto a GND y lees 5V siempre. (a) Que componente o conexión es el principal sospechoso? (b) Que medicion de continuidad (SIN energía) harías para confirmarlo? (c) Describe la reparación. (d) Que habría pasado si en lugar de una resistencia de 10k hubiera puesto un cable directo de la fila del pin a GND?

Objetivo: aplicar diagnóstico sistemático: medir, hipotetizar, confirmar sin energía, reparar; y entender por que la resistencia pull-down no se puede sustituir por un cable.

Solución del ejercicio 6

(a) Si el pin lee 5V siempre, esta permanentemente conectado a 5V o la ruta a GND esta interrumpida. Como el botón esta bien, el sospechoso es la resistencia pull-down de 10k: probablemente una de sus patitas quedó en un agujero equivocado (fila que no es la del pin) o quedó floja, así que el pin quedó "flotando" y el ruido lo mantiene en HIGH, o quedó conectado de forma que no jala a GND.

(b) Sin energía, multímetro en continuidad: una punta en la fila del pin (fila 30 del ejemplo) y otra en el riel azul (GND). Si NO pita a través de la resistencia... ojo, la continuidad a través de 10k normalmente no pita (el umbral suele ser menor a unos 100 ohmios). Mejor medicion: modo resistencia entre la fila del pin y GND: debe leer cerca de 10k. Si lee infinito (OL), la resistencia no esta conectando el pin con GND: ahí esta la falla. También verifica continuidad entre cada patita de la resistencia y su fila correspondiente.

(c) Reparación: con la energía desconectada, reubica la patita de la resistencia a la fila correcta (la misma fila del pin y del cable al Arduino) y la otra patita a una fila conectada con cable al riel azul. Verifica con el multímetro en modo resistencia (10k entre pin y GND), energiza y comprueba que sin presionar lee 0V y presionando lee 5V.

(d) Si hubiera un cable directo del pin a GND, al presionar el botón se uniría 5V con GND sin resistencia en medio: cortocircuito. La fuente entregaría toda la corriente que pudiera por ese cable, se calentaría, y el Arduino podría dañarse. La resistencia de 10k existe precisamente para que, al presionar, la corriente entre 5V y GND sea de solo 0.5 mA. Nunca sustituyas una resistencia pull-down por un cable.

Resumen de la unidad

En esta unidad aprendiste el lenguaje eléctrico de la robótica. Los tres conceptos fundamentales son el voltaje (la "presión" eléctrica, en voltios), la corriente (el flujo de electrones, en amperios o miliamperios) y la resistencia (la oposicion al flujo, en ohmios). La analogía del agua te da la intuicion: mas presión o tubería mas ancha, mas caudal. La ley de Ohm ($V = I \times R$, con sus despejes $I = V / R$ y $R = V / I$) convierte esa intuicion en numeros, y con ella calculas corrientes, resistencias y voltajes en cualquier circuito. También conociste la potencia ($P = V \times I$), que te dice cuanto calor disipa un componente y si tu resistencia aguantará.

Aprendiste que los componentes se conectan en serie (misma corriente, voltaje repartido, resistencias que se suman) o en paralelo (mismo voltaje, corriente repartida, la fuente entrega la suma). Esa distinción explica por que el LED va en serie con su resistencia pero varios LEDs van en paralelo entre si con 5V.

Conociste los componentes pasivos esenciales: la resistencia y su código de colores (memoriza la secuencia negro-café-rojo-naranja-amarillo-verde-azul-violeta-gris-blanco), el protoboard y sus conexiones internas (filas de 5, rieles de alimentación, canal central), el LED con su polaridad (patita larga al positivo, siempre con resistencia en serie) y el pulsador con su resistencia pull-down para lecturas digitales limpias.

Dominaste el multímetro en sus tres mediciones seguras: voltaje en paralelo con el circuito funcionando, continuidad sin energía para rastrear conexiones, y resistencia sin energía y con el componente aislado. Aprendiste las fuentes de poder (USB, pilas, batería de 9V, litio, reguladores) y el criterio para elegirlos: sumar las corrientes y dejar margen. Entendiste el cortocircuito (resistencia casi cero, corriente descontrolada) y los cinco hábitos para evitarlo, empezando por armar siempre sin energía.

Las reglas de seguridad quedan grabadas: nunca la red eléctrica, trabajar sin energía al conectar, respetar polaridad, no exceder 40 mA por pin ni 200 mA totales del Arduino, tratar las baterías con respeto, y desconectar de inmediato si algo se calienta o huele a quemado.

Con esta base, las siguientes unidades serán mucho más fáciles: cuando conectes un sensor estarás midiendo voltajes que ya sabes interpretar; cuando muevas un motor estarás gestionando corrientes que ya sabes calcular; y cuando algo no funcione, tendrás el método de diagnóstico para encontrar la falla en minutos. La electrónica dejó de ser una caja negra: ahora es tu herramienta. Nos vemos en la Unidad 4, donde encenderemos el cerebro del robot: el Arduino.

UNIDAD 4: SENSORES I - DISTANCIA ULTRASONICA

Introducción

Imagina por un momento que entras a un cuarto totalmente oscuro. No ves nada, absolutamente nada. La única forma de saber si hay una pared enfrente de ti es extender los brazos y caminar despacio, esperando topar con algo. Te mueves lento, te tropiezas, y cada paso es una adivinanza. Así de torpe es un robot que no tiene sensores: puede mover sus motores y dar órdenes a sus ruedas, pero no tiene ni idea de que hay a su alrededor. Es, en términos prácticos, ciego.

Los sensores son los sentidos del robot. Así como tu tienes ojos para ver, oídos para escuchar, piel para sentir el tacto y nariz para oler, un robot necesita dispositivos que le permitan percibir el mundo físico que lo rodea. Sin sensores, un robot es solo una máquina que ejecuta órdenes a ciegas: puede avanzar diez centímetros o diez metros, pero no sabe si enfrente hay una pared, un escalón, una persona o un precipicio. Con sensores, el robot empieza a tener algo parecido a la percepción: puede reaccionar, puede tomar decisiones y puede comportarse de forma inteligente. De hecho, la diferencia entre un juguete con motor y un robot de verdad casi siempre esta en los sensores.

En esta unidad vamos a estudiar el sensor mas importante para cualquier robot que se mueve: el sensor de distancia. Si lo piensas bien, de todos los sentidos que podría tener un robot móvil, la distancia es el mas fundamental. Un robot que camina, rueda o vuela necesita saber constantemente que tan lejos están los obstáculos. Sin esa información no puede evitar chocar, no puede seguir una pared, no puede detenerse a tiempo, no puede entrar por una puerta sin pegarle al marco. La distancia es, para el robot móvil, lo que la vista es para ti cuando caminas por la calle: la información que te mantiene a salvo y te permite llegar a donde vas.

Piensa en los robots que ya existen en el mundo real. Los aspiradores robot que limpian las casas solos tienen que detectar muebles, paredes y escaleras para no caerse por ellas. Los carros autónomos están llenos de sensores de distancia que miden constantemente la separación con otros vehículos, con los peatones y con las banquetas. Los drones que vuelan solos usan sensores de distancia para no estrellarse contra árboles o edificios. Incluso los carros comunes y corrientes de hoy traen sensores de reversa que pitan cuando te acercas demasiado a otro carro al estacionarte. Todos estos aparatos, desde el mas sencillo hasta el

mas sofisticado, dependen de una misma idea básica: medir que tan lejos esta lo que tengo enfrente.

El sensor que vamos a usar en esta unidad se llama HC-SR04. Es un sensor de distancia ultrasónico, lo que significa que mide distancias usando ondas de sonido que el oído humano no puede escuchar. Es barato (cuesta apenas unos dólares), es fácil de conseguir en cualquier tienda de electrónica o por internet, se conecta al Arduino con solo cuatro cables y puede medir distancias desde 2 centímetros hasta 400 centímetros, es decir, hasta 4 metros. Con esas características no es raro que sea el sensor de distancia mas popular del mundo entre estudiantes, aficionados y creadores de robots. Si alguna vez viste un robotito casero que evita obstáculos, es casi seguro que tenía un HC-SR04 al frente, como si fueran sus ojos.

La forma en que funciona este sensor es fascinante y muy fácil de entender, porque la naturaleza lo inventó millones de años antes que nosotros. Los murciélagos vuelan en la oscuridad total sin chocar con nada gracias a un sistema llamado ecolocalizacion: emiten chillidos agudos que nosotros no podemos oír, esos sonidos viajan por el aire, rebotan en los objetos y regresan como eco. El murciélago escucha el eco y, según el tiempo que tardó en regresar, su cerebro calcula que tan lejos esta el obstáculo. Si el eco regresa rápido, el objeto esta cerca; si tarda mas, esta lejos. El HC-SR04 hace exactamente lo mismo, pero con electrónica: manda una rafaga de sonido ultrasónico por un "parlante" especial, espera a que el eco regrese a su "micrófono", mide el tiempo que paso entre la emision y la recepcion, y con ese tiempo calcula la distancia. Tu Arduino va a hacer el papel del cerebro del murciélago.

En esta unidad vas a construir dos proyectos completos y funcionales. El primero es un medidor de distancia digital: apuntas el sensor hacia un objeto y el Arduino te dice en la pantalla de tu computadora a cuántos centímetros esta, como una cinta métrica electrónica. El segundo es una alarma de proximidad: un sistema que enciende luces de colores según que tan cerca este un objeto y que hace sonar un zumbador cada vez mas rápido conforme el objeto se acerca, igualito que los sensores de reversa de los automoviles modernos. Cuando termines, vas a tener en tus manos un sistema de percepción real, del tipo que usan los robots de verdad.

Pero no solo vamos a conectar cables y copiar código. Vas a entender a fondo que esta pasando: por que el sensor necesita un pulso de exactamente 10 microsegundos para activarse, de donde sale la formula para calcular la distancia, por que a veces las lecturas brincan o dan valores raros y como corregir esos problemas con técnicas de programación como los promedios móviles. También vas a conocer las limitaciones del sensor, porque todo sensor las tiene: hay materiales que no detecta bien, hay ángulos en los que falla y hay situaciones de ruido que lo confunden. Un buen roboticista no solo sabe usar un sensor, sabe también cuando no confiar ciegamente en el.

Al final de esta unidad serás capaz de conectar el HC-SR04 a tu Arduino Uno, programar la medición de distancia desde cero, mostrar los resultados en el monitor serie, construir un medidor visual con LEDs, armar una alarma de proximidad con buzzer y estabilizar las lecturas para que sean confiables. Estos conocimientos son la base de la Unidad 5, donde usaremos este mismo sensor para construir un robot que evita obstáculos por sí solo. Así que pon atención, porque lo que aprendas aquí lo vas a usar durante todo el resto del curso.

Antes de empezar, asegúrate de tener a la mano tu Arduino Uno, el cable USB, un protoboard, cables de conexión, el sensor HC-SR04, tres LEDs (rojo, amarillo y verde), tres resistencias de 220 ohmios, un buzzer activo de 5 voltios y, si quieres hacer el ejercicio del radar, un servomotor SG90. Todo esto lo consigues en kits básicos de Arduino que venden en línea. Si ya tienes el kit con el que hiciste las unidades anteriores, casi seguro ya tienes todo lo necesario. Manos a la obra: vamos a darle sentidos a tu robot.

Explicación detallada

El principio del eco: como los murciélagos

Todo empieza con una pregunta sencilla: como mides la distancia a algo que no puedes tocar? Los humanos resolvemos esto con la vista: nuestros ojos estiman distancias usando la perspectiva, el tamaño aparente de los objetos y la visión en dos ojos. Pero un Arduino no tiene ojos, así que necesita otro truco. El truco que usa el HC-SR04 es el eco, y para entenderlo bien vamos a ver primero como lo hace la naturaleza.

Los murciélagos son los campeones mundiales de la medición de distancia sin vista. Vuelan de noche, en cuevas oscuras, esquivando rocas, árboles y hasta insectos diminutos que cazan al vuelo. Lo logran con la ecolocalización. El proceso es así: el murciélago emite un chillido muy agudo, tan agudo que el oído humano no lo puede escuchar. Ese sonido viaja por el aire a la velocidad del sonido. Cuando las ondas de sonido chocan contra un objeto, rebotan y regresan hacia el murciélago como un eco. El murciélago escucha ese eco con sus orejas y su cerebro hace un cálculo rapidísimo: mide el tiempo que paso entre el chillido y el eco, y con ese tiempo deduce la distancia. Un eco que regresa en muy poco tiempo significa un objeto muy cercano; un eco que tarda mas significa un objeto lejano. Así de simple y así de poderoso.

Los delfines y las ballenas hacen algo parecido bajo el agua, y los submarinos humanos usan el sonar, que es exactamente la misma idea aplicada con tecnología. La palabra "sonar" viene de "sound navigation and ranging", navegación y medición por sonido. El HC-SR04 es, en esencia, un sonar miniatura que cabe en la palma de tu mano y cuesta menos que un taco.

El sensor tiene dos cilindros metálicos al frente que parecen dos ojitos. Esos dos cilindros son los que hacen todo el trabajo. Uno es el emisor: es como una bocinita que lanza una rafaga de sonido ultrasónico, es decir, sonido de una frecuencia tan alta (40 kilohercios) que los humanos no podemos oirlo. El otro es el receptor: es como un micrófono muy sensible que esta afinado para escuchar justo esa frecuencia y detectar cuando regresa el eco. Entre los dos cilindros hay un circuitito electrónico que se encarga de la comunicación con el Arduino.

El proceso de medicion ocurre en cuatro pasos que se repiten rapidísimo, decenas de veces por segundo si quieres:

Paso 1: el Arduino le ordena al sensor que dispare. Lo hace poniendo en alto (HIGH) un pin llamado Trig durante 10 microsegundos. Esos 10 microsegundos son la senal de "fuego".

Paso 2: el sensor emite una rafaga de 8 pulsos de sonido ultrasónico a 40 kilohercios por el cilindro emisor. Dura una fracción de segundo y es inaudible para nosotros.

Paso 3: el sonido viaja por el aire, choca contra el objeto mas cercano que este enfrente del sensor y rebota de regreso. El cilindro receptor se queda escuchando, esperando el eco.

Paso 4: cuando el eco llega, el sensor pone en alto un pin llamado Echo durante un tiempo exactamente igual al tiempo que el sonido tardó en ir y volver. El Arduino mide cuanto tiempo estuvo en alto ese pin y con ese dato calcula la distancia.

Fíjate bien en el paso 4, porque ahí esta la clave de todo: el tiempo que el pin Echo permanece en alto es el tiempo de vuelo del sonido, ida y vuelta. Si el objeto esta a 1 metro, el sonido tiene que viajar 1 metro de ida y 1 metro de vuelta, o sea 2 metros en total. Como sabemos a que velocidad viaja el sonido, podemos convertir ese tiempo en distancia con una simple multiplicación. Mas adelante veremos la formula exacta con un ejemplo numérico completo.

El sensor HC-SR04: pines y especificaciones

Vamos a conocer al protagonista de esta unidad cara a cara. El HC-SR04 es una pequeña tarjeta rectangular de color azul (casi siempre) que mide aproximadamente 45 milímetros de largo por 20 de ancho. Al frente tiene los dos cilindros metálicos que ya mencionamos, separados por unos 25 milímetros. En la parte de atrás o abajo tiene cuatro pines para conectarlo. Esos cuatro pines son todo lo que necesitas para hacerlo funcionar, y cada uno tiene un trabajo muy claro:

- VCC: es la alimentación del sensor. Se conecta a los 5 voltios del Arduino. El sensor necesita 5V para funcionar correctamente; si le das menos voltaje, las mediciones salen mal o el sensor ni siquiera responde.

- Trig (trigger, disparador): es la entrada por donde el Arduino le ordena al sensor que haga una medicion. Se conecta a cualquier pin digital del Arduino. Cuando el Arduino pone este pin en HIGH durante 10 microsegundos, el sensor dispara su rafaga ultrasonica.
- Echo (eco): es la salida por donde el sensor le avisa al Arduino cuanto tardó el eco en regresar. Se conecta a cualquier pin digital del Arduino configurado como entrada. El sensor pone este pin en HIGH durante un tiempo igual al tiempo de vuelo del sonido.
- GND (tierra): es la referencia eléctrica, el "cero voltios". Se conecta al GND del Arduino. Sin esta conexión el circuito no esta completo y nada funciona.

Las especificaciones técnicas del HC-SR04, según su hoja de datos, son las siguientes. Vale la pena conocerlas porque definen que puedes esperar del sensor y que no:

- Voltaje de operación: 5 voltios de corriente directa.
- Corriente de consumo: alrededor de 15 miliamperios, muy poquito; el Arduino lo alimenta sin problema.
- Rango de medicion: de 2 centímetros a 400 centímetros (4 metros). Mas cerca de 2 cm el sensor se "satura" y da lecturas malas; mas lejos de 400 cm el eco es tan débil que el sensor ya no lo detecta.
- Precisión típica: alrededor de 3 milímetros, o sea 0.3 centímetros. Para un sensor tan barato es una precisión excelente.
- Angulo de medicion: aproximadamente 15 grados. Esto significa que el sensor no "ve" en una línea recta perfecta, sino en un cono que se abre unos 15 grados hacia los lados. Un objeto que este muy a un lado puede no ser detectado, y un objeto dentro del cono si.
- Frecuencia de operación: 40 kilohercios, ultrasonica e inaudible para humanos (aunque algunos perros y gatos si pueden oirla un poco).

Guarda bien estos numeros en tu memoria: 2 cm a 400 cm de rango, 5 voltios de alimentación y 15 grados de angulo. Los vas a necesitar cuando diseñes tus proyectos.

Como funciona la medicion paso a paso

Ahora vamos a desmenuzar la medicion con todo detalle, porque entender esto a fondo es lo que te convierte en alguien que domina el sensor y no en alguien que solo copia código.

Todo empieza cuando tu programa pone el pin Trig en HIGH durante 10 microsegundos y luego lo regresa a LOW. Esos 10 microsegundos son importantes: es el tiempo mínimo que el sensor necesita para reconocer la orden de disparo. Si el pulso es mas corto, el sensor podría no detectarlo; si es mas largo no pasa nada grave, pero 10 microsegundos es el valor estándar

que indica la hoja de datos. En código esto se hace así: `digitalWrite(trigPin, HIGH);`
`delayMicroseconds(10); digitalWrite(trigPin, LOW);`

En cuanto el sensor detecta ese pulso, su circuito interno genera automáticamente una rafaga de 8 ciclos de sonido a 40 kilohercios y la emite por el cilindro transmisor. Tu no tienes que generar esos 40 kilohercios con el Arduino; el sensor lo hace solo. Tu única responsabilidad es darle el pulso de disparo de 10 microsegundos.

El sonido sale disparado a la velocidad del sonido en el aire, que a temperatura ambiente (unos 20 grados Celsius) es de aproximadamente 343 metros por segundo. Esa velocidad es una constante física que vamos a usar en nuestros cálculos. Ojo: la velocidad del sonido cambia un poco con la temperatura (aumenta unos 0.6 metros por segundo por cada grado Celsius), pero para nuestros propósitos 343 m/s es un valor perfectamente bueno.

La rafaga de sonido viaja, choca contra el objeto y rebota. El rebote viaja de regreso y llega al cilindro receptor. En ese momento el sensor pone el pin Echo en HIGH. Fíjate que el pin Echo se pone en HIGH cuando el sensor EMPIEZA a escuchar el eco? No: el sensor pone Echo en HIGH desde el momento en que emite la rafaga y lo mantiene en HIGH hasta que recibe el eco. En la práctica, para ti como programador, lo que importa es esto: la duración del pulso HIGH en el pin Echo es igual al tiempo total que el sonido estuvo viajando, de ida y vuelta.

Para medir esa duración usamos una función de Arduino llamada `pulseIn()`. Esta función espera a que un pin cambie a HIGH, empieza a contar el tiempo, y se detiene cuando el pin regresa a LOW. El resultado lo devuelve en microsegundos. Así de fácil: `duración = pulseIn(echoPin, HIGH);` y ya tienes el tiempo de vuelo en microsegundos.

Ahora viene la parte matematica, pero no te asustes porque es una sola multiplicación y una división. Queremos convertir el tiempo de vuelo (en microsegundos) a distancia (en centímetros). Sabemos que el sonido viaja a 343 metros por segundo. Vamos a convertir esa velocidad a unidades que nos sirvan: centímetros por microsegundo.

343 metros por segundo son 34300 centímetros por segundo. Un segundo tiene 1,000,000 de microsegundos. Entonces la velocidad del sonido es $34300 / 1,000,000 = 0.0343$ centímetros por microsegundo. Redondeando, 0.034 cm por cada microsegundo.

Si el sonido viaja 0.034 centímetros en cada microsegundo, entonces la distancia total recorrida (ida y vuelta) es: $\text{distancia total} = \text{tiempo en microsegundos} \times 0.034$. Pero esa distancia total incluye el viaje de ida Y el de vuelta, y a nosotros solo nos interesa la distancia de ida (del sensor al objeto). Así que dividimos entre 2. La formula final queda así:

$\text{distancia en cm} = (\text{tiempo en microsegundos} \times 0.034) / 2$

Que también se puede escribir como: $\text{distancia} = \text{tiempo} \times 0.017$. Muchos programas usan directamente el 0.017, pero es mejor que entiendas de donde sale: es el 0.034 dividido entre 2.

Vamos a hacer un ejemplo numérico completo para que quede clarísimo. Supón que apuntas el sensor a una pared y la función `pulseIn()` te devuelve 1000 microsegundos. Paso a paso:

1. Tiempo de vuelo medido: 1000 microsegundos.
2. Distancia total recorrida por el sonido (ida + vuelta): $1000 \times 0.034 = 34$ centímetros.
3. Esa distancia es el doble de lo que nos interesa, así que dividimos entre 2: $34 / 2 = 17$ centímetros.
4. Resultado: la pared esta a 17 centímetros del sensor.

Otro ejemplo: `pulseIn()` devuelve 5882 microsegundos. Calculamos: $5882 \times 0.034 = 200$ centímetros de recorrido total; entre 2 = 100 centímetros. La pared esta a 1 metro. Tiene sentido: el sonido tarda mas en ir y volver cuando el objeto esta mas lejos.

Y un ejemplo al revés para verificar: si el objeto esta a 50 centímetros, el sonido debe recorrer 100 centímetros en total. A 0.034 cm por microsegundo, eso toma $100 / 0.034 = 2941$ microsegundos. Así que `pulseIn()` debería devolver aproximadamente 2941. Si haces la prueba en casa con una cinta métrica, verás que los numeros cuadran bastante bien.

Conexiones al Arduino

Conectar el HC-SR04 al Arduino Uno es de lo mas sencillo que hay en electrónica: son solo cuatro cables y no necesitas resistencias ni componentes extra para la lectura básica. Aquí esta la conexión estándar que usaremos en todos los ejemplos de esta unidad:

- VCC del sensor al pin 5V del Arduino.
- GND del sensor al pin GND del Arduino.
- Trig del sensor al pin digital 9 del Arduino.
- Echo del sensor al pin digital 10 del Arduino.

Puedes usar otros pines digitales si quieres (los pines 9 y 10 son solo una convención cómoda), pero asegúrate de actualizar los numeros en tu código. Los pines Trig y Echo pueden ser cualquier pin digital; no necesitan ser pines PWM ni nada especial.

Un consejo práctico: cuando conectes el sensor al protoboard, fíjate en que los cuatro pines queden bien insertados y haciendo buen contacto. Una conexión floja en el pin Echo es una de las causas mas comunes de lecturas en cero o lecturas locas. También verifica dos veces que VCC vaya a 5V y no a 3.3V: con 3.3V el sensor a veces parece funcionar pero las mediciones salen incorrectas.

Otro detalle importante: el sensor debe apuntar hacia donde quieres medir, con los dos cilindros de frente al objeto y sin nada obstruyendo el camino entre el sensor y el objetivo. Si pones el sensor dentro de una caja o detrás de una tapa, el sonido rebota en la tapa y siempre medirá la distancia a la tapa, no a lo que esta mas allá.

Limitaciones del sensor: cuando no confiar ciegamente

Todo sensor tiene limitaciones, y conocerlas es tan importante como saber usarlo. El HC-SR04 es excelente para su precio, pero hay situaciones donde falla o da lecturas engañosas. Vamos a verlas una por una.

Superficies blandas y absorbentes: el sonido rebota muy bien en superficies duras y lisas como paredes, madera, metal, plástico o cartón. Pero los materiales blandos como la tela, la esponja, el algodón o la ropa absorben gran parte del sonido en lugar de reflejarlo. El eco que regresa es muy débil y el sensor puede no detectarlo, reportando una distancia mayor a la real o directamente ningún eco. Si tu robot tiene que detectar un sofá tapizado, ten en cuenta que el sensor lo "verá" peor que a una pared.

Ángulos de incidencia: el sonido rebota siguiendo la misma regla que la luz en un espejo: el ángulo de llegada es igual al ángulo de salida. Si el sensor apunta de frente a una pared (90 grados), el eco regresa directo al sensor y todo bien. Pero si la pared esta inclinada, digamos a 45 grados, el sonido rebota hacia un lado y nunca regresa al receptor. El sensor entonces no detecta la pared aunque este ahí. En la práctica esto significa que las superficies muy inclinadas respecto al sensor son "invisibles" para el. El ángulo útil de detección es de unos 15 grados hacia cada lado del eje central.

Objetos muy pequeños o delgados: un objeto pequeño refleja poco sonido, así que el eco es débil. Un lápiz parado a 2 metros probablemente no será detectado. Los objetos delgados como cables o patas de silla también son difíciles de detectar a distancia.

Objetos muy cercanos: por debajo de los 2 centímetros el sensor no puede medir bien. Esto pasa porque el receptor queda "sordo" por un instante después de la emisión (el sonido de la propia ráfaga lo satura). Si necesitas medir distancias menores a 2 cm, este no es el sensor adecuado.

Interferencia entre sensores: si tienes dos HC-SR04 disparando al mismo tiempo y apuntando en direcciones cercanas, la rafaga de uno puede ser escuchada por el otro y producir lecturas falsas. La solución es dispararlos por turnos, con una pequeña pausa entre uno y otro.

Ruido eléctrico y mecánico: motores, relevadores y fuentes de poder conmutadas pueden generar interferencia eléctrica que afecte las lecturas. Mantener los cables del sensor algo separados de los cables de los motores ayuda mucho.

Temperatura: como vimos, la velocidad del sonido cambia con la temperatura. A 0 grados es de unos 331 m/s y a 30 grados es de unos 349 m/s. Si usas siempre 343 m/s en tu formula, a temperaturas extremas tendrás un pequeño error (de alrededor del 2 por ciento). Para proyectos caseros esto no importa, pero es bueno saber que existe.

Ninguna de estas limitaciones hace inútil al sensor; simplemente significa que debes diseñar tus proyectos teniéndolas en cuenta. Por ejemplo, si tu robot evita obstáculos, es buena idea no confiar en una sola lectura aislada, sino pedir varias y compararlas, como veremos en la sección de estabilización.

Técnicas para lecturas estables

Si ya jugaste un poco con el HC-SR04, habrás notado algo: las lecturas a veces brincan. Un momento dice 50 cm, al siguiente dice 52, luego 48, y de repente salta a 300 o a 0 sin que nada se haya movido. Esto es normal y le pasa a todo el mundo. Las lecturas inestables vienen de ecos débiles, de rebotes múltiples en la habitación, de pequeñas variaciones eléctricas o simplemente de que el objeto no esta perfectamente quieto. La buena noticia es que con un poco de programación puedes convertir esas lecturas brincalonas en mediciones sólidas y confiables. Aquí van las técnicas mas usadas, de la mas simple a la mas robusta.

Descartar valores fuera de rango: lo primero y mas fácil. Sabemos que el sensor mide entre 2 y 400 cm. Si una lectura dice 0, 1 o 500, es físicamente imposible (o al menos no confiable), así que la ignoramos. En código: si la distancia es menor a 2 o mayor a 400, no la tomes en cuenta y conserva la última lectura valida. Esto solo ya elimina la mayoría de los brincos locos.

Promedio simple: en lugar de hacer una sola medicion, haz varias (por ejemplo 5 o 10) y calcula el promedio. Los errores aleatorios tienden a cancelarse entre si: si una lectura salió un poco alta y otra un poco baja, el promedio queda cerca del valor real. La desventaja es que un valor extremo (un brinco loco a 400) puede jalar el promedio lejos del valor real. Por eso conviene combinar el promedio con el descarte de valores fuera de rango.

Mediana: es la técnica favorita de muchos roboticistas porque es muy resistente a los valores extremos. Tomas un número impar de lecturas (por ejemplo 5), las ordenas de menor a mayor

y te quedas con la del medio. Ejemplo: lecturas de 50, 51, 400, 49, 52. Ordenadas: 49, 50, 51, 52, 400. La mediana es 51, que es un valor sensato. Fíjate que el 400, que era un brinco loco, quedó completamente ignorado sin necesidad de ninguna regla especial. La mediana es un poco mas trabajosa de programar que el promedio porque hay que ordenar los valores, pero con 5 elementos es fácil.

Promedio móvil: es como el promedio simple, pero pensado para lecturas continuas en el tiempo. Mantienes una lista de las últimas N lecturas (digamos las últimas 5) y cada vez que llega una lectura nueva, sacas la mas vieja, metes la nueva y calculas el promedio de la lista. Así la medicion mostrada se suaviza: reacciona a los cambios reales pero no brinca con cada lectura ruidosa. Es la técnica ideal para la alarma de proximidad, donde quieres que las luces y el buzzer respondan de forma estable.

Pausas entre mediciones: un error muy común es disparar el sensor lo mas rápido posible, una medicion tras otra sin descanso. El problema es que el eco de una medicion anterior puede seguir rebotando por la habitación cuando ya disparaste la siguiente, y el sensor confunde el eco viejo con el nuevo. La hoja de datos recomienda esperar al menos 50 a 60 milisegundos entre mediciones. En la práctica, un `delay(50)` o `delay(100)` entre lecturas hace maravillas por la estabilidad. Si tu programa necesita ir mas rápido, 30 milisegundos suele ser el mínimo aceptable.

Combinar técnicas: en proyectos serios se usan varias técnicas juntas. Por ejemplo: descartar valores fuera de rango, tomar 5 lecturas con pausas de 50 ms entre ellas, quedarse con la mediana y luego pasarla por un promedio móvil. Suena como mucho trabajo, pero son apenas unas líneas de código y la diferencia en la calidad de las mediciones es enorme. En la sección de ejemplos veremos el promedio móvil implementado paso a paso.

Errores comunes y como evitarlos

Cuando algo no funciona con el HC-SR04, casi siempre es uno de estos errores clásicos. Revisa esta lista antes de desesperarte:

Olvidar configurar el pin Echo como INPUT: el pin Trig debe ser OUTPUT (el Arduino manda la orden) y el pin Echo debe ser INPUT (el Arduino escucha la respuesta). Si configuras Echo como OUTPUT por error, `pulseIn()` no podrá leer nada y siempre obtendras cero. Las dos líneas correctas en el setup son: `pinMode(trigPin, OUTPUT);` `pinMode(echoPin, INPUT);`

Medir demasiado rápido sin pausas: como acabamos de ver, disparar el sensor sin esperar entre mediciones produce ecos cruzados y lecturas inestables. Pon siempre un `delay` de al menos 50 milisegundos entre una medicion y la siguiente.

Confundir microsegundos con milisegundos: la función `pulseIn()` devuelve microsegundos y el pulso de disparo se hace con `delayMicroseconds(10)`. Si por error usas `delay(10)` (milisegundos) en lugar de `delayMicroseconds(10)`, el pulso será mil veces mas largo. No rompe nada, pero no es lo correcto y en algunos sensores puede causar comportamientos raros.

Olvidar la división entre 2: la formula es $(\text{tiempo} \times 0.034) / 2$. Si olvidas dividir entre 2, todas tus distancias saldrán al doble del valor real. Es un error muy común y muy fácil de detectar: mide un objeto a una distancia conocida con una regla y compara.

Usar variables `int` para el tiempo: `pulseIn()` puede devolver valores de hasta unos 24000 microsegundos (para 400 cm). Eso cabe en un `int`, pero si haces operaciones intermedias es mas seguro usar `long`. Declara la duración como `long` o `unsigned long` para evitar desbordamientos silenciosos.

Alimentar el sensor con 3.3V: el HC-SR04 esta diseñado para 5V. Conectarlo al pin de 3.3V del Arduino a veces "funciona" pero las lecturas son poco confiables. Usa siempre el pin 5V.

Probar el sensor apuntando a tu mano muy de cerca: recuerda el límite inferior de 2 cm. Si pegas el sensor a tu mano, las lecturas serán malas. Para las primeras pruebas apunta a una pared que este a mas de 20 cm.

No inicializar el pin Trig en LOW: antes de mandar el pulso de 10 microsegundos es buena práctica asegurarse de que Trig este en LOW con un pequeño delay de 2 microsegundos. Esto garantiza un pulso limpio. En código: `digitalWrite(trigPin, LOW); delayMicroseconds(2); digitalWrite(trigPin, HIGH); delayMicroseconds(10); digitalWrite(trigPin, LOW);`

Con esto cerramos la teoría. Ya sabes como funciona el eco, conoces al HC-SR04 por dentro y por fuera, entiendes la formula de la distancia, sabes conectarlo y conoces sus limitaciones, sus técnicas de estabilización y sus errores típicos. Es hora de ensuciarnos las manos con código real.

Ejemplos paso a paso

En esta sección vamos a construir cuatro proyectos de menor a mayor complejidad. Cada uno incluye la lista de materiales, los pasos de conexión numerados, el código completo comentado y una explicación de como funciona el programa. Todos los códigos compilan en Arduino Uno. Te recomiendo hacerlos en orden, porque cada proyecto usa lo aprendido en el anterior.

Ejemplo A: lectura básica y monitor serie

Este es el "hola mundo" de los sensores de distancia: el programa mide la distancia una vez cada medio segundo y la muestra en el monitor serie de tu computadora. Es el punto de partida de todo lo demás.

Materiales: Arduino Uno, cable USB, protoboard, 4 cables de conexión, sensor HC-SR04.

Pasos de conexión:

1. Conecta el pin VCC del sensor al pin 5V del Arduino.
2. Conecta el pin GND del sensor a un pin GND del Arduino.
3. Conecta el pin Trig del sensor al pin digital 9 del Arduino.
4. Conecta el pin Echo del sensor al pin digital 10 del Arduino.
5. Verifica que el sensor quede con los cilindros apuntando hacia afuera del protoboard, sin obstáculos enfrente.

Código:

```

// Medidor de distancia basico con HC-SR04
// Muestra la distancia en centimetros en el monitor serie

const int trigPin = 9;  // Pin de disparo del sensor
const int echoPin = 10; // Pin de escucha del sensor

void setup() {
  pinMode(trigPin, OUTPUT); // Trig es salida: el Arduino manda la orden
  pinMode(echoPin, INPUT);  // Echo es entrada: el Arduino escucha el eco
  digitalWrite(trigPin, LOW); // Nos aseguramos de empezar con Trig en LOW
  Serial.begin(9600);        // Iniciamos la comunicacion con la computadora
}

void loop() {
  // Paso 1: generar un pulso limpio de disparo
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10); // Pulso de 10 microsegundos: la orden de "fuego"
  digitalWrite(trigPin, LOW);

  // Paso 2: medir cuanto tiempo tarda el eco en regresar
  long duracion = pulseIn(echoPin, HIGH); // Tiempo en microsegundos

  // Paso 3: convertir el tiempo a distancia en centimetros
  // Formula: distancia = (tiempo x 0.034) / 2
  float distancia = (duracion * 0.034) / 2.0;

  // Paso 4: mostrar el resultado en el monitor serie
  Serial.print("Distancia: ");
  Serial.print(distancia);
  Serial.println(" cm");

  delay(500); // Esperar medio segundo antes de la proxima medicion
}

```

Explicación del código: en el setup configuramos los pines y abrimos el puerto serie a 9600 baudios para hablar con la computadora. En el loop repetimos el ciclo de medicion: primero el pulso de disparo de 10 microsegundos en Trig, luego pulseIn() mide la duración del pulso HIGH en Echo, después aplicamos la formula para obtener centímetros y finalmente imprimimos el resultado. El delay(500) al final cumple dos funciones: hace la lectura legible para el ojo humano y le da al sensor tiempo suficiente para que los ecos viejos se apaguen antes de la siguiente medicion.

Para probarlo, sube el programa, abre el monitor serie (el icono de la lupa en el IDE de Arduino, configurado a 9600 baudios) y apunta el sensor a una pared. Mueve tu mano frente al sensor y observa como cambian los numeros. Prueba medir con una regla una distancia conocida y compara: deberías estar dentro de un centimetro del valor real.

Ejemplo B: medidor con 3 LEDs (cerca, medio, lejos)

Ahora vamos a hacer que la distancia se vea sin necesidad de la computadora: tres LEDs que indican si el objeto esta lejos (verde), a distancia media (amarillo) o cerca (rojo). Es como un semaforo de proximidad.

Materiales: todo lo del ejemplo A, mas 3 LEDs (rojo, amarillo, verde), 3 resistencias de 220 ohmios y cables extra.

Pasos de conexión:

1. Arma primero todo el circuito del ejemplo A (el sensor queda igual).
2. Conecta la pata larga (anodo) del LED verde al pin digital 4 del Arduino a través de una resistencia de 220 ohmios.
3. Conecta la pata larga del LED amarillo al pin digital 5 con otra resistencia de 220 ohmios.
4. Conecta la pata larga del LED rojo al pin digital 6 con la tercera resistencia de 220 ohmios.
5. Conecta las patas cortas (catodos) de los tres LEDs a la línea GND del protoboard.
6. Revisa que ningún LED quede conectado al revés: la pata larga siempre va hacia el pin del Arduino (con la resistencia en medio) y la corta hacia tierra.

Código:

```
// Semaforo de distancia con HC-SR04 y 3 LEDs
// Verde: lejos (mas de 60 cm). Amarillo: medio (20 a 60 cm). Rojo: cerca (menos de 20 cm).

const int trigPin = 9;
const int echoPin = 10;
const int ledVerde = 4;
const int ledAmarillo = 5;
const int ledRojo = 6;

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}
```

```

pinMode(ledVerde, OUTPUT);
pinMode(ledAmarillo, OUTPUT);
pinMode(ledRojo, OUTPUT);
Serial.begin(9600);
}

float medirDistancia() {
    // Dispara el sensor y devuelve la distancia en cm
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    long duracion = pulseIn(echoPin, HIGH);
    float distancia = (duracion * 0.034) / 2.0;
    return distancia;
}

void loop() {
    float distancia = medirDistancia();

    Serial.print("Distancia: ");
    Serial.print(distancia);
    Serial.println(" cm");

    // Apagar todos los LEDs antes de decidir cual encender
    digitalWrite(ledVerde, LOW);
    digitalWrite(ledAmarillo, LOW);
    digitalWrite(ledRojo, LOW);

    // Encender el LED que corresponda segun la distancia
    if (distancia > 60) {
        digitalWrite(ledVerde, HIGH);    // Lejos: todo tranquilo
    } else if (distancia >= 20) {
        digitalWrite(ledAmarillo, HIGH); // Medio: precaucion
    } else {
        digitalWrite(ledRojo, HIGH);     // Cerca: peligro
    }

    delay(200); // Pequenna pausa para estabilizar
}

```

Explicación del código: la novedad aquí es que la lógica de medición se movió a una función propia llamada `medirDistancia()`, que devuelve la distancia como número decimal (float). Esto es una excelente práctica: cuando necesites medir en varias partes del programa, llamas a la

función en lugar de repetir las mismas líneas. En el loop, después de medir, apagamos los tres LEDs y encendemos solo el que corresponda usando una cadena de if / else if / else con los umbrales de 60 y 20 centímetros. Nota que los umbrales los puedes cambiar a tu gusto: son los que definen que significa "cerca" y "lejos" para tu proyecto.

Pruébalo acercando tu mano lentamente al sensor: verás como el verde se apaga, se enciende el amarillo y finalmente el rojo cuando tu mano esta muy cerca. Es exactamente el mismo principio que usan muchos juguetes y sistemas de advertencia.

Ejemplo C: alarma de proximidad con buzzer (estilo sensor de reversa)

Este proyecto imita los sensores de reversa de los automoviles: un buzzer que pita, y mientras mas cerca este el objeto, mas rápido pita. Cuando el objeto esta muy cerca, el pitido se vuelve casi continuo. Es un proyecto muy satisfactorio porque lo "sientes" funcionando.

Materiales: todo lo del ejemplo B (puedes dejar los LEDs conectados o quitarlos), mas un buzzer activo de 5V y un cable extra.

Pasos de conexión:

1. Conecta la pata positiva del buzzer (la mas larga, o la marcada con un signo +) al pin digital 8 del Arduino.
2. Conecta la pata negativa del buzzer a GND.
3. Si tu buzzer no tiene marcas, prueba en ambas direcciones: un buzzer activo conectado al revés simplemente no suena, no se daña.
4. El sensor HC-SR04 queda conectado igual que antes (pines 9 y 10).

Código:

```
// Alarma de proximidad con HC-SR04 y buzzer
// Mientras mas cerca el objeto, mas rapido pita. Como el sensor de reversa de un
auto.

const int trigPin = 9;
const int echoPin = 10;
const int buzzerPin = 8;

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(buzzerPin, OUTPUT);
  Serial.begin(9600);
}
```

```

float medirDistancia() {
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    long duracion = pulseIn(echoPin, HIGH);
    float distancia = (duracion * 0.034) / 2.0;
    return distancia;
}

void loop() {
    float distancia = medirDistancia();

    Serial.print("Distancia: ");
    Serial.print(distancia);
    Serial.println(" cm");

    if (distancia < 100) {
        // Solo pitar si hay algo a menos de 1 metro. Mas lejos: silencio.
        // Calculamos el intervalo entre pitidos: cerca = intervalo corto, lejos =
intervalo largo.
        // A 5 cm el intervalo es 50 ms (pita rapidisimo). A 100 cm el intervalo es 1000
ms (pita lento).
        int intervalo = (int)(distancia * 10); // 10 ms por cada centimetro
        if (intervalo < 50) intervalo = 50;      // Limite minimo para que se distinga el
pitido

        digitalWrite(buzzerPin, HIGH); // Encender el pitido
        delay(80);                      // Duracion fija del pitido: 80 ms
        digitalWrite(buzzerPin, LOW);   // Apagar el pitido
        delay(intervalo);               // Esperar antes del siguiente pitido
    } else {
        digitalWrite(buzzerPin, LOW); // Nada cerca: silencio total
        delay(200);
    }
}

```

Explicación del código: la idea central esta en la línea `int intervalo = (int)(distancia * 10);`. Convertimos la distancia en un tiempo de espera: cada centimetro de distancia equivale a 10 milisegundos de silencio entre pitidos. Si el objeto esta a 10 cm, el intervalo es de 100 ms y el buzzer pita muy rápido; si esta a 80 cm, el intervalo es de 800 ms y pita lento y tranquilo. La duración del pitido en si es fija (80 ms) para que siempre se escuche igual. El `if (distancia < 100)` evita que el buzzer pite todo el tiempo cuando no hay nada cerca: mas allá de un metro,

silencio. El límite mínimo de 50 ms evita que a distancias diminutas el intervalo se vuelva cero o negativo, lo que haría un pitido continuo raro.

Pruébalo caminando hacia el sensor: escucharás como el ritmo del pitido se acelera conforme te acercas, igual que cuando un carro se estaciona de reversa. Si quieres, combina este código con los LEDs del ejemplo B y tendrás un sistema de advertencia con luz y sonido a la vez.

Ejemplo D: lectura estabilizada con promedio móvil

Este último ejemplo no agrega hardware nuevo, sino inteligencia al programa. Implementa el promedio móvil que estudiamos en la teoría: guarda las últimas 5 lecturas validas y siempre muestra el promedio, lo que elimina los brincos y entrega una medicion suave y confiable.

Materiales: solo el circuito del ejemplo A (sensor conectado a los pines 9 y 10).

Pasos:

1. Usa el mismo circuito del ejemplo A, sin cambios.
2. Sube el siguiente código y observa el monitor serie.
3. Para comparar, mueve tu mano rápido frente al sensor y nota como la lectura mostrada cambia de forma suave en lugar de brincar.

Código:

```
// Lectura estabilizada con promedio movil
// Guarda las ultimas 5 lecturas validas y muestra su promedio

const int trigPin = 9;
const int echoPin = 10;
const int NUM_LECTURAS = 5;           // Tamanno de la ventana del promedio movil

float lecturas[NUM_LECTURAS];         // Arreglo con las ultimas lecturas
int indice = 0;                       // Posicion donde se guardara la proxima lectura
bool memoriaLlena = false;           // Indica si ya tomamos las primeras 5 lecturas

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  Serial.begin(9600);
  // Inicializar el arreglo con un valor neutro
  for (int i = 0; i < NUM_LECTURAS; i++) {
    lecturas[i] = 0;
  }
}
```



```

    Serial.println("Sistema iniciado. Tomando lecturas...");
}

float medirDistancia() {
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    long duracion = pulseIn(echoPin, HIGH);
    float distancia = (duracion * 0.034) / 2.0;
    return distancia;
}

float promedioMovil(float nuevaLectura) {
    // Paso 1: descartar lecturas fuera del rango valido del sensor
    if (nuevaLectura < 2 || nuevaLectura > 400) {
        // Lectura invalida: no la guardamos, devolvemos el promedio actual
    } else {
        // Lectura valida: la guardamos en la posicion actual del arreglo
        lecturas[indice] = nuevaLectura;
        indice++;
        if (indice >= NUM_LECTURAS) {
            indice = 0; // Volver al inicio: el arreglo es circular
            memoriaLlena = true; // Ya dimos la primera vuelta completa
        }
    }

    // Paso 2: calcular el promedio de las lecturas guardadas
    float suma = 0;
    int cuenta = memoriaLlena ? NUM_LECTURAS : indice;
    if (cuenta == 0) return 0; // Aun no hay lecturas validas
    for (int i = 0; i < cuenta; i++) {
        suma += lecturas[i];
    }
    return suma / cuenta;
}

void loop() {
    float cruda = medirDistancia(); // Lectura directa del sensor
    float suave = promedioMovil(cruda); // Lectura estabilizada

    Serial.print("Cruda: ");
    Serial.print(cruda);
    Serial.print(" cm | Estable: ");

```

```
Serial.print(suave);  
Serial.println(" cm");  
  
delay(100); // Pausa entre mediciones para evitar ecos cruzados  
}
```

Explicación del código: el corazón del programa es el arreglo `lecturas[]`, que funciona como una ventana circular de las últimas 5 mediciones. Cada vez que llega una lectura nueva y valida (entre 2 y 400 cm), se guarda en la posición marcada por índice y el índice avanza; al llegar al final, regresa al inicio y sobrescribe la lectura mas vieja. Así siempre tenemos las 5 lecturas mas recientes. La función `promedioMovil()` suma esas lecturas y divide entre cuantas haya, devolviendo el promedio. En el loop mostramos tanto la lectura cruda como la estabilizada para que compares: verás que la columna "Cruda" brinca y la columna "Estable" se mueve con suavidad. La variable `memoriaLlena` resuelve el arranque: durante las primeras 5 lecturas todavía no hay 5 valores, así que promediamos solo los que ya tenemos.

Este patrón del promedio móvil lo vas a usar muchísimo en robótica, no solo con este sensor. Cada vez que una senal venga ruidosa, un promedio móvil es la primera herramienta que debes probar.

Práctica

La teoría sin práctica se olvida en una semana. Estos ejercicios están diseñados para que modifiques, experimentes y rompas cosas (en el buen sentido) con el HC-SR04. Hazlos en orden: cada uno te prepara para el siguiente. Todos usan el mismo circuito base del sensor en los pines 9 y 10, mas los componentes que se indiquen.

Ejercicio 1: cambia los rangos del semaforo

Enunciado: toma el programa del Ejemplo B (el semaforo de 3 LEDs) y cambia los umbrales para que el LED rojo se encienda a menos de 30 cm, el amarillo entre 30 y 100 cm, y el verde a mas de 100 cm. Además, agrega al monitor serie un mensaje de texto que diga "CERCA", "MEDIO" o "LEJOS" según corresponda.

Objetivo: practicar la modificación de umbrales en condicionales y combinar salidas físicas (LEDs) con mensajes en el monitor serie. Entenderás que los rangos no son mágicos: los defines tu según lo que necesite tu proyecto.

Solución del ejercicio 1

Solo hay que cambiar los numeros en los if y agregar los mensajes. El resto del programa queda igual:

```
void loop() {
    float distancia = medirDistancia();

    digitalWrite(ledVerde, LOW);
    digitalWrite(ledAmarillo, LOW);
    digitalWrite(ledRojo, LOW);

    Serial.print("Distancia: ");
    Serial.print(distancia);
    Serial.print(" cm -> ");

    if (distancia > 100) {
        digitalWrite(ledVerde, HIGH);
        Serial.println("LEJOS");
    } else if (distancia >= 30) {
        digitalWrite(ledAmarillo, HIGH);
        Serial.println("MEDIO");
    } else {
        digitalWrite(ledRojo, HIGH);
        Serial.println("CERCA");
    }

    delay(200);
}
```

Explicación: cambiamos 60 por 100 y 20 por 30 en las condiciones, y agregamos Serial.print del estado. Fíjate en el detalle de usar Serial.print (sin ln) para la primera parte y Serial.println para el final: así el número y la palabra quedan en la misma línea. Este ejercicio parece trivial, pero es exactamente el tipo de ajuste que harás en cada robot: calibrar los umbrales según el tamaño del robot, la velocidad a la que se mueve y el espacio donde opera.

Ejercicio 2: barra de proximidad con 5 LEDs

Enunciado: amplía el semaforo a 5 LEDs que funcionen como una barra de nivel: mientras mas cerca este el objeto, mas LEDs se encienden. Usa los pines 2, 3, 4, 5 y 6. Define estos rangos: mas de 100 cm ningún LED encendido; 80 a 100 cm se enciende 1 LED; 60 a 80 cm se encienden 2; 40 a 60 cm se encienden 3; 20 a 40 cm se encienden 4; menos de 20 cm se encienden los 5. Los LEDs deben encenderse en orden, del pin 2 al pin 6.

Objetivo: practicar el control de múltiples salidas y el uso de arreglos para manejar pines. Aprenderás un truco muy útil: guardar los pines en un arreglo y recorrerlos con un ciclo for.

Solución del ejercicio 2

```
// Barra de proximidad con 5 LEDs y HC-SR04
// Mas cerca el objeto = mas LEDs encendidos

const int trigPin = 9;
const int echoPin = 10;
const int pinesLED[5] = {2, 3, 4, 5, 6}; // Arreglo con los 5 pines

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  for (int i = 0; i < 5; i++) {
    pinMode(pinesLED[i], OUTPUT); // Configurar los 5 pines como salida en un ciclo
  }
  Serial.begin(9600);
}

float medirDistancia() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duracion = pulseIn(echoPin, HIGH);
  return (duracion * 0.034) / 2.0;
}

void loop() {
  float distancia = medirDistancia();
  Serial.print("Distancia: ");
  Serial.print(distancia);
  Serial.println(" cm");

  // Calcular cuantos LEDs encender segun la distancia (0 a 5)
  int encendidos = 0;
  if (distancia <= 100) encendidos = 1;
  if (distancia <= 80) encendidos = 2;
  if (distancia <= 60) encendidos = 3;
  if (distancia <= 40) encendidos = 4;
  if (distancia <= 20) encendidos = 5;
```

```
// Encender los primeros "encendidos" LEDs y apagar el resto
for (int i = 0; i < 5; i++) {
  if (i < encendidos) {
    digitalWrite(pinesLED[i], HIGH);
  } else {
    digitalWrite(pinesLED[i], LOW);
  }
}

delay(200);
}
```

Explicación: la clave está en la serie de if independientes (no else if): cada uno puede subir el contador encendidos si la distancia es menor o igual a su umbral. Como se evalúan en orden de mayor a menor distancia, el valor final de encendidos es el correcto. Luego un solo ciclo for recorre el arreglo de pines y enciende exactamente los primeros N. Este patrón de "arreglo de pines + for" te va a ahorrar muchísimo código en proyectos con muchos LEDs, botones o motores.

Ejercicio 3: radar que barre con un servo

Enunciado: monta el sensor HC-SR04 sobre un servomotor SG90 de modo que el sensor gire. Programa al Arduino para que el servo barra de 0 a 180 grados y de regreso, tomando una medición de distancia en cada posición (cada 15 grados). Muestra en el monitor serie el ángulo y la distancia medida, por ejemplo "Ángulo: 45 - Distancia: 78 cm". El resultado es un mini radar casero que "ve" en abanico.

Materiales extra: servomotor SG90 y su cableado (senial al pin 11, VCC a 5V, GND a tierra). Necesitarás una base para fijar el sensor al brazo del servo: cinta adhesiva doble cara funciona bien para las pruebas.

Objetivo: combinar dos dispositivos (servo y sensor) en un solo programa y entender el concepto de barrido, que es la base de los radares y lidars reales. Practicarás la librería Servo de Arduino.

Solución del ejercicio 3

```
#include <Servo.h> // Librería para controlar servomotores

const int trigPin = 9;
const int echoPin = 10;
const int servoPin = 11;
```

```

Servo miServo;

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  miServo.attach(servoPin); // Conectar el servo al pin 11
  Serial.begin(9600);
  Serial.println("Radar iniciado. Barriendo de 0 a 180 grados...");
}

float medirDistancia() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duracion = pulseIn(echoPin, HIGH, 30000); // Tiempo maximo de espera: 30 ms
  if (duracion == 0) return 400; // Sin eco = muy lejos (fuera de rango)
  return (duracion * 0.034) / 2.0;
}

void loop() {
  // Barrido de ida: de 0 a 180 grados en pasos de 15
  for (int angulo = 0; angulo <= 180; angulo += 15) {
    miServo.write(angulo);
    delay(300); // Dar tiempo al servo para llegar a la posicion
    float d = medirDistancia();
    Serial.print("Angulo: ");
    Serial.print(angulo);
    Serial.print(" - Distancia: ");
    Serial.print(d);
    Serial.println(" cm");
  }
  // Barrido de regreso: de 180 a 0 grados
  for (int angulo = 180; angulo >= 0; angulo -= 15) {
    miServo.write(angulo);
    delay(300);
    float d = medirDistancia();
    Serial.print("Angulo: ");
    Serial.print(angulo);
    Serial.print(" - Distancia: ");
    Serial.print(d);
    Serial.println(" cm");
  }
}

```

Explicación: usamos la librería Servo que ya viene con el IDE de Arduino: `#include <Servo.h>`, crear el objeto Servo, y `attach()` para asociarlo al pin. El programa mueve el servo en pasos de 15 grados, espera 300 ms a que se estabilice en cada posición (si mides mientras el servo aun se mueve, la lectura sale mal) y toma la distancia. Hay dos detalles nuevos: `pulseIn(echoPin, HIGH, 30000)` incluye un tiempo máximo de espera de 30000 microsegundos; si no llega ningún eco en ese tiempo, la función devuelve 0 en lugar de quedarse esperando hasta un segundo completo. Y cuando duración es 0 lo interpretamos como "muy lejos" devolviendo 400. Con los datos de angulo y distancia podrías hasta dibujar un mapa en la computadora, pero eso lo dejaremos para un proyecto avanzado.

Ejercicio 4: mide la velocidad de respuesta del sistema

Enunciado: escribe un programa que mida cuanto tiempo tarda tu sistema completo en detectar un cambio. El programa debe tomar lecturas continuas y, en el momento en que la distancia cambie mas de 10 cm respecto a la lectura anterior, imprimir en el monitor serie cuántos milisegundos pasaron desde la lectura anterior. Repite la prueba moviendo tu mano rápido frente al sensor y anota los tiempos. Prueba después con el promedio móvil del Ejemplo D activado y compara: cuanto mas lento se vuelve el sistema al estabilizar?

Objetivo: entender el compromiso entre velocidad y estabilidad, uno de los conceptos mas importantes en robótica. Un sistema muy estabilizado reacciona lento; un sistema muy rápido es ruidoso. Aprenderás a medir ese compromiso con numeros reales.

Solución del ejercicio 4

```

const int trigPin = 9;
const int echoPin = 10;

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  Serial.begin(9600);
  Serial.println("Prueba de velocidad. Mueve tu mano rapido frente al sensor.");
}

float medirDistancia() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duracion = pulseIn(echoPin, HIGH);
  return (duracion * 0.034) / 2.0;
}

void loop() {
  float anterior = medirDistancia();
  unsigned long t0 = millis(); // Marcar el tiempo actual en milisegundos

  while (true) {
    delay(50); // Pausa entre lecturas
    float actual = medirDistancia();

    // Si el cambio es mayor a 10 cm, reportar el tiempo de reaccion
    if (abs(actual - anterior) > 10) {
      unsigned long reaccion = millis() - t0;
      Serial.print("Cambio detectado: de ");
      Serial.print(anterior);
      Serial.print(" a ");
      Serial.print(actual);
      Serial.print(" cm en ");
      Serial.print(reaccion);
      Serial.println(" ms");
      break; // Salir del while y empezar una nueva prueba
    }
  }

  delay(1000); // Pausa antes de la siguiente prueba
}

```


Explicación: el programa toma una lectura base y anota el tiempo con `millis()`. Luego entra en un ciclo que mide cada 50 ms y compara con la base usando `abs()` (valor absoluto de la diferencia). Cuando detecta un cambio mayor a 10 cm, calcula cuanto tiempo paso y lo reporta. Verás tiempos de reacción de unos 50 a 150 ms con lectura directa. Si agregas el promedio móvil de 5 lecturas, el tiempo de reacción sube a 300 ms o mas, porque el promedio necesita varias lecturas nuevas para "darse cuenta" del cambio. Esa es la lección: la estabilización cuesta velocidad. En un robot que evita obstáculos a alta velocidad, 300 ms pueden ser demasiados; en un medidor de estacionamiento, no importan nada. Tu decides el equilibrio según tu proyecto.

Ejercicio 5: detector de objetos pequeños

Enunciado: pon a prueba los limites del sensor. Coloca a 50 cm del sensor distintos objetos pequeños: una moneda, un lápiz parado, una pelota de tenis, un vaso de plástico, un libro. Para cada objeto, anota si el sensor lo detecta (la lectura baja a ~50 cm) o no (la lectura sigue mostrando la pared del fondo). Luego repite la prueba acercando cada objeto hasta encontrar la distancia máxima a la que cada uno es detectado. Escribe tus resultados en una tabla en tu cuaderno: objeto, distancia máxima de detección, observaciones.

Objetivo: descubrir experimentalmente las limitaciones del sensor que estudiamos en la teoría (objetos pequeños, superficies, ángulos). Un roboticista de verdad no se queda con la hoja de datos: verifica con pruebas reales que puede y que no puede detectar su sensor.

Solución del ejercicio 5

Este ejercicio no tiene un código nuevo: usa el programa del Ejemplo A o el del Ejemplo D. La "solución" son los resultados que debes obtener y su interpretación. Aquí van los resultados típicos para que compares con los tuyos:

- Libro (20 x 25 cm, tapa dura): detectado sin problema hasta mas de 200 cm. Superficie grande y dura: eco fuerte y claro.
- Vaso de plástico: detectado hasta unos 100-150 cm. Superficie curva que dispersa parte del sonido.
- Pelota de tenis: detectada hasta unos 60-80 cm. Pequeña y de superficie suave que absorbe sonido.
- Lápiz parado (delgado): detectado solo hasta 20-30 cm. Muy poco área para reflejar el sonido.

- Moneda: prácticamente indetectable a mas de 10-15 cm. Demasiado pequeña para este sensor.

Si tus numeros son parecidos, tu sensor funciona correctamente y acabas de comprobar en la práctica las limitaciones de la teoría: el tamaño del objeto y el material de su superficie determinan que tan lejos puede estar para ser detectado. La lección de diseño es clara: si tu robot debe detectar obstáculos pequeños o delgados (como patas de sillas o cables en el piso), un solo HC-SR04 apuntando al frente quizá no baste; necesitarás montarlo mas bajo, usar varios sensores o complementar con otro tipo de sensor, como los infrarrojos que veremos en la próxima unidad.

Ejercicio 6: conversor de unidades y umbral configurable

Enunciado: escribe un programa que muestre la distancia en tres unidades a la vez: centímetros, pulgadas y metros. Además, el programa debe permitirte cambiar el umbral de alarma sin modificar el código: al arrancar, pregunta en el monitor serie "A que distancia (cm) quieres la alarma?" y usa el número que escribas como nuevo umbral. Si la distancia medida es menor al umbral, enciende el LED rojo del pin 6; si no, mantenlo apagado.

Objetivo: practicar conversiones de unidades en código y, sobre todo, aprender a leer datos que el usuario escribe en el monitor serie con `Serial.parseFloat()`. Esto convierte a tu programa en algo interactivo y configurable, una habilidad clave para proyectos mas grandes donde no quieres reprogramar el Arduino cada vez que cambias un parámetro.

Solución del ejercicio 6

```
// Conversor de unidades con umbral configurable por el usuario
// Muestra cm, pulgadas y metros. El usuario define la distancia de alarma.

const int trigPin = 9;
const int echoPin = 10;
const int ledRojo = 6;

float umbralAlarma = 30; // Valor por defecto, se cambiara con lo que escriba el
usuario

void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(ledRojo, OUTPUT);
  Serial.begin(9600);
```

```

// Preguntar el umbral al usuario
Serial.println("A que distancia (cm) quieres la alarma?");
Serial.println("Escribe el numero y presiona Enter.");

// Esperar hasta que el usuario escriba algo
while (Serial.available() == 0) {
    // No hacer nada, solo esperar
}
umbralAlarma = Serial.parseFloat(); // Leer el numero escrito

Serial.print("Umbral de alarma configurado en: ");
Serial.print(umbralAlarma);
Serial.println(" cm");
Serial.println("Midiendo... acerca tu mano al sensor.");
}

float medirDistancia() {
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    long duracion = pulseIn(echoPin, HIGH);
    return (duracion * 0.034) / 2.0;
}

void loop() {
    float cm = medirDistancia();

    // Conversiones de unidades
    float pulgadas = cm / 2.54; // 1 pulgada = 2.54 cm
    float metros = cm / 100.0; // 1 metro = 100 cm

    Serial.print(cm);
    Serial.print(" cm | ");
    Serial.print(pulgadas);
    Serial.print(" in | ");
    Serial.print(metros);
    Serial.println(" m");

    // Alarma segun el umbral que configuro el usuario
    if (cm < umbralAlarma) {
        digitalWrite(ledRojo, HIGH);
    } else {
        digitalWrite(ledRojo, LOW);
    }
}

```

```
}  
  
  delay(300);  
}
```

Explicación: hay dos ideas nuevas aquí. La primera es la conversión de unidades, que es aritmética simple: dividir los centímetros entre 2.54 da pulgadas, y entre 100 da metros. La segunda es la lectura del usuario: `Serial.available()` te dice cuántos caracteres han llegado por el puerto serie; el `while` espera hasta que llegue al menos uno. Luego `Serial.parseFloat()` lee esos caracteres y los convierte en número decimal. Así el umbral deja de estar quemado en el código y pasa a ser un dato que el usuario decide en cada arranque. Prueba escribir 10, luego reinicia el Arduino y escribe 150: el comportamiento de la alarma cambia sin tocar una sola línea de código. Esta técnica de "configuración por puerto serie" la usan muchísimos proyectos reales para calibrar sensores sin reprogramar.

Resumen de la unidad

En esta unidad le diste a tu robot su sentido mas importante: la percepción de la distancia. Aprendiste que los sensores son los sentidos del robot y que, para un robot móvil, medir distancias es tan fundamental como la vista lo es para ti al caminar.

Conociste el principio del eco, la misma técnica que usan los murciélagos para volar en la oscuridad: emitir un sonido, escuchar su rebote y calcular la distancia según el tiempo que tardó en regresar. Viste que el sensor HC-SR04 hace exactamente eso con ultrasonido de 40 kilohercios, usando sus dos cilindros como emisor y receptor.

Estudiaste al HC-SR04 a fondo: sus cuatro pines (VCC, Trig, Echo, GND), su rango de 2 a 400 centímetros, su alimentación de 5 voltios y su angulo de detección de unos 15 grados. Aprendiste el ciclo completo de medicion: pulso de 10 microsegundos en Trig, rafaga ultrasonica automática, eco recibido y pulso en Echo cuya duración es el tiempo de vuelo del sonido.

Dominaste la formula de la distancia: $\text{distancia en cm} = (\text{tiempo en microsegundos} \times 0.034) / 2$, y entendiste de donde sale cada número: 0.034 es la velocidad del sonido expresada en centímetros por microsegundo, y la división entre 2 corrige que el sonido viaja de ida y vuelta. Practicaste la formula con ejemplos numéricos completos.

Conociste las limitaciones del sensor: superficies blandas que absorben el sonido, ángulos inclinados que desvían el eco, objetos pequeños difíciles de detectar, el límite inferior de 2 cm y la interferencia entre sensores cercanos. Y aprendiste a compensar el ruido con técnicas de estabilización: descartar valores fuera de rango, promedios, mediana, promedio móvil y pausas entre mediciones.

En la práctica construiste cuatro proyectos: la lectura básica con monitor serie, el semáforo de distancia con 3 LEDs, la alarma de proximidad con buzzer estilo sensor de reversa y la lectura estabilizada con promedio móvil. Además resolviste cinco ejercicios que te llevaron a modificar umbrales, controlar 5 LEDs con arreglos, construir un radar con servo, medir la velocidad de respuesta del sistema y probar experimentalmente los límites del sensor.

También memorizaste los errores más comunes: olvidar el pinMode del Echo como INPUT, medir sin pausas, confundir microsegundos con milisegundos, olvidar dividir entre 2 y alimentar el sensor con 3.3V.

Todo esto es la base de la Unidad 5, donde montaremos este sensor en un chasis con ruedas y construiremos un robot móvil que detecta obstáculos y los evita por sí solo. El sensor ya lo dominas; lo que sigue es ponerle patas.

UNIDAD 5: SENSORES II - LUZ, PRESENCIA Y TEMPERATURA

En la Unidad 4 aprendiste a usar sensores que hablan en binario: botones, PIR, interruptores magnéticos. Un botón está presionado o no está presionado; un sensor PIR detecta movimiento o no lo detecta. Son sensores digitales: solo conocen dos respuestas, SI y NO, HIGH y LOW. Con ellos construiste una alarma, un sistema antirrobo y un contador de pasos. Funcionaron muy bien, pero el mundo real casi nunca es blanco y negro.

Piensa en la luz de tu cuarto. No solo está "encendida" o "apagada": puede haber penumbra, una luz suave de lámpara, el sol del mediodía entrando por la ventana. Tu ojo distingue cientos de niveles de brillo. Piensa en la temperatura: no solo hace "calor" o "frío", hay un rango continuo de grados. Un robot que solo entienda SI o NO es como una persona que solo puede decir "sí" o "no" a todo: puede sobrevivir, pero no puede tener una conversación rica con su entorno.

Para eso existen los sensores analógicos: dispositivos que devuelven un rango continuo de valores en lugar de dos estados. Un sensor de luz puede decirte "hay un 30 por ciento de luz", "un 70 por ciento", "un 95 por ciento". Un sensor de temperatura puede decirte exactamente cuántos grados hay: 24.5, 25.1, 25.3. Con esa información tu robot puede tomar decisiones mucho más inteligentes: no solo "enciende la lámpara", sino "enciende la lámpara al 40 por ciento de brillo porque todavía hay algo de luz del atardecer".

Esta unidad es la segunda dedicada a sensores, y es un paso gigante en tu formación como constructor de robots. Aquí vas a dominar tres familias de sensores que aparecen en casi todos los robots del mundo real: los sensores de luz, los sensores de presencia por infrarrojo y los sensores de temperatura. Entre las tres clases de esta unidad vas a construir una lámpara automática que se enciende sola cuando oscurece, un detector de línea que puede seguir una pista negra (la base de los robots seguidores de línea que compiten en torneos), y un termómetro digital con alarma.

Antes de entrar a cada sensor, necesitas entender una idea fundamental que hace posible todo lo demás: como un Arduino, que es una máquina digital que solo entiende unos y ceros, puede leer valores continuos del mundo analógico. La respuesta es el convertidor analógico-digital, y dominarlo te abre la puerta a cientos de sensores.

Imagínate que quieres medir cuanta agua hay en un vaso. Podrías poner una regla junto al vaso y leer los centímetros. El convertidor analógico-digital (ADC por sus siglas en inglés: Analog to Digital Converter) hace algo parecido con el voltaje. El Arduino Uno tiene un ADC de

10 bits. Eso significa que toma el voltaje de entrada, que va de 0 a 5 voltios, y lo divide en 1024 escalones numerados del 0 al 1023. Cada escalón vale aproximadamente 4.88 milivoltios (5 voltios divididos entre 1024). Cuando lees un pin analógico con la función `analogRead()`, el Arduino te devuelve un número entre 0 y 1023 que representa el voltaje que esta viendo en ese pin en ese momento.

Hagamos los cálculos con calma porque los vas a usar todo el tiempo. Si `analogRead()` te devuelve 0, significa que el pin tiene 0 voltios. Si te devuelve 1023, significa que tiene 5 voltios (el máximo). Si te devuelve 512, el voltaje es aproximadamente 2.5 voltios (la mitad). La formula general es: $\text{voltaje} = (\text{lectura} * 5.0) / 1023.0$. Con una lectura de 700, el voltaje seria $(700 * 5.0) / 1023.0 = 3.42$ voltios. Con una lectura de 300, seria $(300 * 5.0) / 1023.0 = 1.47$ voltios. Memoriza esta formula o guárdala en un papel pegado a tu mesa de trabajo, porque la vas a usar en cada proyecto analógico que hagas en tu vida.

Los pines analógicos del Arduino Uno son los que dicen A0, A1, A2, A3, A4 y A5. Son seis entradas dedicadas al ADC. Ojo con un detalle importante: aunque se llaman pines analógicos, lo que hacen es LEER valores analógicos; por dentro todo sigue siendo digital. El ADC es el traductor entre los dos mundos. Otro detalle: los pines A0 a A5 también se pueden usar como pines digitales normales con `digitalRead()` y `digitalWrite()` si alguna vez te quedas sin pines digitales, pero su superpoder es `analogRead()`.

Ahora, la diferencia entre sensores analógicos y digitales en la práctica. Un sensor digital se conecta a un pin digital y lo lees con `digitalRead()`: te devuelve HIGH o LOW. Un sensor analógico se conecta a un pin analógico (A0-A5) y lo lees con `analogRead()`: te devuelve un número entre 0 y 1023. Esa es la diferencia en código, y es sencilla. La diferencia en lo que puedes hacer con esa información es enorme.

Pero aquí viene una trampa en la que caen casi todos los principiantes: un sensor analógico no se conecta solo. La mayoría de los sensores de luz, por ejemplo, son resistencias variables: su resistencia cambia con la luz, pero el Arduino no puede medir resistencia directamente, solo puede medir voltaje. Necesitas un circuito intermedio que convierta "resistencia que cambia" en "voltaje que cambia". Ese circuito se llama divisor de voltaje, y es probablemente el circuito mas util que vas a aprender en toda tu carrera como maker. Lo veremos a fondo en la explicación detallada, con su formula y un ejemplo numérico completo.

Los sensores que vas a dominar en esta unidad son los siguientes. Primero, la fotorresistencia o LDR (Light Dependent Resistor): una resistencia que baja cuando hay luz y sube cuando hay oscuridad. Con un divisor de voltaje y `analogRead()` puedes medir el nivel de luz de 0 a 1023. Es el sensor mas barato que existe (cuesta centavos) y el mas didactico. Con el vas a construir la lámpara automática.

Segundo, el sensor infrarrojo de reflectancia TCRT5000: un par emisor-receptor de luz infrarroja que mide cuanta luz infrarroja rebota de una superficie. Una superficie blanca refleja mucho; una negra refleja poco. Con eso tu robot puede "ver" una línea negra pintada sobre un piso blanco y seguirla. Este es el sensor de los robots seguidores de línea, una de las categorías mas populares en competencias de robótica. El módulo TCRT5000 trae un potenciómetro para calibrar el umbral entre "veo blanco" y "veo negro", y aprender a calibrarlo es una habilidad que te servirá con docenas de módulos.

Tercero, los sensores de obstáculos por infrarrojo: parecidos al TCRT5000 pero pensados para detectar si hay un objeto enfrente, no para leer el piso. Dan una salida digital (hay obstáculo / no hay obstáculo) y también traen potenciómetro de ajuste de distancia. Son los "ojos" de los robots que evitan chocar.

Cuarto, el sensor de temperatura LM35: un circuito integrado que entrega un voltaje proporcional a la temperatura, exactamente 10 milivoltios por cada grado Celsius. Si el ambiente esta a 25 grados, el LM35 entrega 250 milivoltios (0.25 voltios). Con `analogRead()` y una multiplicación conviertes la lectura a grados. Es analógico, preciso y no necesita librerías.

Quinto, el DHT11: un módulo digital que mide temperatura y humedad. A diferencia del LM35, se comunica con el Arduino por un protocolo digital de un solo cable y necesitas instalar una librería para leerlo. Es menos preciso que otros sensores, pero mide dos cosas a la vez y es muy popular en estaciones meteorológicas caseras.

Al final de esta unidad también vas a entender un concepto de control que separa a los proyectos de juguete de los proyectos profesionales: la histéresis. Es una palabra elegante para una idea simple: cuando tu lámpara automática decide encenderse porque oscurecio, no quieres que parpadee como loca justo en el límite entre luz y oscuridad. La histéresis le da al sistema "memoria" del estado anterior para que las decisiones sean estables. Lo verás con un ejemplo del ventilador que se prende y apaga sin parar, y aprenderás a evitarlo con dos umbrales en lugar de uno.

Lo que vas a construir en esta unidad, en resumen: primero medirás la luz con una LDR y la verás en el monitor serie; después harás que un LED se encienda solo cuando oscurezca (tu primera lámpara automática, con histéresis para que no parpadee); luego calibraras un TCRT5000 para detectar una línea negra sobre papel blanco; y cerrarás con un termómetro digital con LM35 que dispara una alarma con buzzer cuando la temperatura sube demasiado. En la sección de práctica llevaras todo mas lejos: una barra de LEDs que muestra el nivel de luz como un medidor, un seguidor de luz con dos LDR que gira hacia la lámpara mas brillante, un contador de objetos con infrarrojo, un termostato con dos umbrales y un registrador de temperatura que escribe una lectura cada minuto.

Un consejo antes de empezar: ten a la mano tu multímetro si tienes uno. Medir el voltaje real en el punto medio del divisor te va a ayudar a entender que esta pasando mucho mas rápido que solo mirar numeros en el monitor serie. Y si no tienes multímetro, no te preocupes: el monitor serie del Arduino IDE será tu instrumento de medicion, y con los ejemplos de esta unidad aprenderás a calibrar todo solo con el.

Vamos a ello. El mundo analógico te estaba esperando.

Explicación detallada

La fotorresistencia LDR: el sensor mas simple del mundo

LDR significa Light Dependent Resistor, resistencia dependiente de la luz. Es un componente con dos patitas y una carita que parece tener una serpiente dibujada encima: esa superficie ondulada es el material fotosensible, generalmente sulfuro de cadmio. Su comportamiento es fácil de entender: cuando recibe mucha luz, su resistencia baja muchísimo (puede llegar a unos 100 ohmios o menos a plena luz del sol); cuando esta en oscuridad, su resistencia sube enormemente (puede llegar a 1 megaohmio, es decir, un millón de ohmios, o mas). Entre esos dos extremos hay un rango continuo: mas luz, menos resistencia; menos luz, mas resistencia.

Fíjate que la LDR no genera voltaje por si sola. No es como una pila. Es una resistencia variable, y el Arduino no puede leer resistencia directamente: solo puede leer voltaje en sus pines analógicos. Por eso necesitas un circuito que convierta "resistencia que cambia con la luz" en "voltaje que cambia con la luz". Ese circuito es el divisor de voltaje, y vale la pena entenderlo a fondo porque lo vas a usar con termistores, potenciómetros, sensores de flexion y muchos otros componentes.

El divisor de voltaje: la formula que todo maker debe saber

Un divisor de voltaje son dos resistencias en serie conectadas entre 5V y GND (tierra). El punto donde se unen las dos resistencias es la salida: ahí mides un voltaje que esta entre 5V y 0V, y su valor depende de la proporcion entre las dos resistencias. La formula es: $V_{salida} = V_{entrada} * (R2 / (R1 + R2))$, donde R1 es la resistencia conectada a 5V (la de arriba) y R2 es la resistencia conectada a GND (la de abajo).

Vamos a desmenuzarla con un ejemplo numérico concreto, porque aquí es donde la teoría se vuelve herramienta. Supón que $R1 = 10,000$ ohmios (10k) y $R2 = 10,000$ ohmios (10k), con $V_{entrada} = 5V$. Entonces $V_{salida} = 5 * (10000 / (10000 + 10000)) = 5 * (10000 / 20000) = 5 * 0.5 = 2.5V$. Lógico: dos resistencias iguales dividen el voltaje a la mitad.

Ahora cambia R2 a 5,000 ohmios (5k), manteniendo R1 en 10k. $V_{salida} = 5 * (5000 / 15000) = 5 * 0.333 = 1.67V$. La salida bajo porque R2 es mas pequeña en proporcion. Y si R2 sube a 20,000 ohmios: $V_{salida} = 5 * (20000 / 30000) = 5 * 0.667 = 3.33V$. La salida subió. La regla de oro: el voltaje de salida "sigue" a R2; si R2 crece respecto a R1, la salida sube; si R2 se achica, la salida baja.

Ahora aplica esto a la LDR. Arma el divisor así: 5V -> LDR (arriba, R1) -> punto medio (al pin A0) -> resistencia fija de 10k (abajo, R2) -> GND. Con mucha luz, la LDR baja a unos 1k ohmio. Calcula: $V_{salida} = 5 * (10000 / (1000 + 10000)) = 5 * (10000 / 11000) = 5 * 0.909 = 4.55V$. El Arduino leerá $analogRead() = (4.55 / 5) * 1023 = 931$. Con oscuridad total, la LDR sube a 1 megaohmio (1,000,000). $V_{salida} = 5 * (10000 / (1000000 + 10000)) = 5 * (10000 / 1010000) = 5 * 0.0099 = 0.05V$. El Arduino leerá aproximadamente 10. Así que con esta configuración: mucha luz = lectura alta (cerca de 900), oscuridad = lectura baja (cerca de 10). Tiene sentido: la LDR es R1 (arriba), y cuando su resistencia baja, "jala" la salida hacia 5V.

Si inviertes el orden (resistencia fija arriba, LDR abajo), el comportamiento se invierte: mucha luz = lectura baja. Ambas configuraciones son validas; solo tienes que saber cual usaste para interpretar las lecturas. En los ejemplos de esta unidad usaremos LDR arriba y resistencia de 10k abajo, así que recuerda: luz alta = número alto.

Un detalle práctico: el valor de la resistencia fija (10k) no es mágico. Funciona bien porque esta en medio del rango de la LDR. Si usarás una resistencia de 100 ohmios, casi todo el rango util quedaría aplastado; si usarás 10 megaohmios, pasaría lo mismo al revés. La regla práctica es elegir una resistencia fija cercana a la resistencia de la LDR en condiciones "normales" de tu aplicación (por ejemplo, con la luz de tu cuarto). 10k es el valor estándar que verás en el 90 por ciento de los tutoriales, y funciona.

analogRead() y el mapeo de valores

Ya sabes que `analogRead(pin)` devuelve un número entre 0 y 1023. Pero ese número crudo no siempre es lo que quieres mostrar o usar. A veces quieres convertirlo a voltaje, a porcentaje de luz, o a un valor de 0 a 255 para controlar el brillo de un LED con PWM. Para eso existe la función `map()`.

La función `map()` toma un valor en un rango y lo convierte proporcionalmente a otro rango. Su forma es: `map(valor, desdeMin, desdeMax, haciaMin, haciaMax)`. Ejemplo: `map(512, 0, 1023, 0, 255)` devuelve 127 o 128 (la mitad de 255, porque 512 es la mitad de 1023). Ejemplo práctico: si tu LDR te da lecturas entre 50 (oscuridad) y 900 (plena luz) y quieres un porcentaje de 0 a 100, usas `map(lectura, 50, 900, 0, 100)`. Ojo: `map()` no limita el resultado; si la lectura sale del rango (por ejemplo, 950), el resultado también sale del rango (daría mas de 100). Para

evitarlo combina con constrain(): porcentaje = constrain(map(lectura, 50, 900, 0, 100), 0, 100). La función constrain() recorta el valor entre un mínimo y un máximo.

Otro uso típico: controlar el brillo de un LED inversamente a la luz ambiente (a mas oscuridad, mas brillo del LED). Si tu lectura va de 50 a 900, puedes hacer brillo = map(lectura, 50, 900, 255, 0). Fíjate que invertí el rango destino (255 a 0): así, lectura baja (oscuridad) da brillo alto. Luego analogWrite(pinLED, brillo). Este truco de invertir rangos en map() es muy común y vale la pena practicarlo.

Importante: map() trabaja con numeros enteros. Si necesitas decimales (por ejemplo, convertir a voltaje con dos decimales), haz la operación manualmente con float: voltaje = lectura * (5.0 / 1023.0). El ".0" es crucial en Arduino: si escribes 5 / 1023 con enteros, el resultado es 0 porque la división entera trunca. Siempre usa 5.0 cuando quieras decimales.

El TCRT5000: ojos infrarrojos para seguir líneas

El TCRT5000 es un sensor optico de reflectancia que viene en un encapsulado pequeño con dos "ojitos": uno es un LED emisor de luz infrarroja y el otro es un fototransistor receptor. La idea es simple: el emisor lanza luz infrarroja hacia abajo; si hay una superficie clara debajo, la luz rebota y el receptor la capta; si hay una superficie oscura (negra), la luz se absorbe y el receptor casi no capta nada. Así el sensor distingue blanco de negro sin tocar nada.

El infrarrojo es invisible para el ojo humano, lo cual es una ventaja: el sensor no se confunde con la luz visible del cuarto (bueno, casi: la luz solar directa si tiene mucho infrarrojo y puede interferir, cosa que veremos en errores comunes). El TCRT5000 se usa típicamente a una distancia de 2 a 15 milímetros de la superficie. Mas lejos, la senal se vuelve débil y poco confiable.

Casi siempre lo compraras como módulo ya armado: una pequeña tarjeta con el TCRT5000, un LED indicador, un comparador (chip LM393) y un potenciómetro azul de ajuste. Este módulo tiene dos salidas: una analógica (A0 del módulo) que te da el valor crudo de reflectancia, y una digital (D0 del módulo) que te da HIGH o LOW según si la reflectancia supera el umbral ajustado con el potenciómetro. Esa es una gran ventaja pedagogica: puedes empezar en modo digital (fácil) y pasar a modo analógico (mas control) cuando quieras.

Como detecta una línea negra sobre blanco: imagina una pista dibujada con cinta de aislar negra sobre una cartulina blanca. El robot lleva el sensor apuntando al piso. Mientras el sensor esta sobre el blanco, recibe mucho rebote: salida digital HIGH (o LOW, según el módulo; hay que verificarlo). Cuando el sensor pasa sobre la línea negra, el rebote cae: la salida cambia. El Arduino lee ese cambio y sabe "estoy sobre la línea" o "me sali de la línea". Con dos o mas sensores (uno a cada lado de la línea) el robot puede corregir su rumbo: si el sensor izquierdo

pierde la línea, gira a la derecha, y viceversa. Ese es el principio completo de un seguidor de líneas, el robot mas clásico de las competencias.

Calibración del potenciómetro del módulo

El potenciómetro azul del módulo ajusta el umbral del comparador: el punto de corte entre "veo blanco" y "veo negro". Calibrarlo es obligatorio porque cada pista, cada altura de montaje y cada iluminación es distinta. El procedimiento es el siguiente, y te recomiendo hacerlo siempre con el LED indicador del módulo a la vista:

1. Monta el sensor a la altura a la que ira en el robot (típicamente 5 a 10 mm del piso).
2. Coloca el sensor sobre la superficie BLANCA.
3. Gira el potenciómetro lentamente con un desarmador pequeño hasta que el LED indicador cambie de estado (se encienda o se apague, según el módulo). Marca mentalmente ese punto.
4. Ahora coloca el sensor sobre la línea NEGRA y verifica que el LED cambie al estado contrario.
5. Ajusta el potenciómetro a un punto intermedio entre ambos extremos, donde el cambio sea claro y estable en ambas superficies.

Si el LED parpadea o cambia con solo mover un poco el sensor, el umbral esta muy justo: alejalo un poco del punto de transicion. Una buena calibración deja un "margen" generoso a cada lado. Tip profesional: calibra con la iluminación que habra durante el uso real. Si calibras en tu cuarto con luz cálida y el robot competira en un gimnasio con luz blanca intensa, la calibración puede fallar. La luz ambiental afecta, aunque el sensor use infrarrojo.

Si usas la salida analógica del módulo en lugar de la digital, no necesitas el potenciómetro: lees el valor crudo con `analogRead()` y defines el umbral en el código. Esto te da mas flexibilidad (puedes cambiar el umbral sin desarmador, e incluso hacer calibración automática al encender), pero requiere que entiendas bien los valores que lees. En los ejemplos usaremos primero la salida digital (mas simple) y en la práctica jugaras con la analógica.

Sensores de obstáculos por infrarrojo

Son primos del TCRT5000 pero miran al frente en lugar de al piso. El módulo típico (a veces marcado como FC-51 o "IR obstacle avoidance") tiene un par emisor-receptor infrarrojo montado en ángulo, de modo que detecta objetos a entre 2 y 30 centímetros (algunos llegan a 80 cm). También trae potenciómetro para ajustar la distancia de detección y LED indicador. Su salida es digital: normalmente HIGH cuando no hay obstáculo y LOW cuando detecta algo (lógica invertida, típica de estos módulos; siempre verifícalo con el LED).

La diferencia conceptual con el TCRT5000 es importante: el TCRT5000 mide reflectancia de una superficie cercana para distinguir colores; el sensor de obstáculos detecta la presencia de un objeto a distancia media. Un robot evasor de obstáculos usa dos o tres de estos sensores al frente; un seguidor de línea usa TCRT5000 abajo. Son herramientas distintas para problemas distintos, aunque la física (infrarrojo que rebota) sea la misma.

Limitaciones que debes conocer: las superficies negras o muy rugosas absorben el infrarrojo y el sensor puede no verlas aunque estén cerca; las superficies transparentes (vidrio, acrílico) lo dejan pasar; y la luz solar directa satura el receptor. Nunca confíes ciegamente en un solo sensor para seguridad real: en robótica educativa están bien, pero un robot que de verdad no debe chocar usa ultrasonido (que verás en la Unidad 6) o combinaciones de sensores.

El LM35: temperatura en milivoltios

El LM35 es un circuito integrado de tres patitas que parece un transistor pequeño. Su magia es que entrega un voltaje directamente proporcional a la temperatura: exactamente 10 milivoltios por cada grado Celsius. A 0 grados entrega 0V, a 25 grados entrega 250 mV (0.25V), a 100 grados entrega 1000 mV (1.0V). No necesita calibración, no necesita librerías, no necesita componentes externos: lo conectas a 5V, GND y un pin analógico, y listo.

La conversión a grados es una multiplicación directa. Lees el pin con `analogRead()`, conviertes a voltaje y divides entre 0.01 (porque cada grado son 0.01 voltios). En una sola formula: $\text{temperaturaC} = (\text{analogRead}(\text{pin}) * 5.0 / 1023.0) / 0.01$, que se simplifica a $\text{temperaturaC} = \text{analogRead}(\text{pin}) * 0.4883$. Redondeando: cada unidad del ADC equivale a unos 0.49 grados. Hagamos un ejemplo numérico completo: si `analogRead()` devuelve 51, el voltaje es $51 * 5.0 / 1023.0 = 0.249\text{V} = 249\text{ mV}$; dividido entre 10 mV por grado = 24.9 grados. Si devuelve 62, son $62 * 0.4883 = 30.3$ grados. Si devuelve 41, son 20.0 grados. Fácil, rápido y sin librerías.

El rango útil del LM35 con alimentación de 5V va de unos 2 grados hasta 150 grados, aunque para ambiente (0 a 50 grados) es perfecto. Su precisión típica es de mas o menos 0.5 grados, mas que suficiente para proyectos caseros. Un consejo: el LM35 es sensible al ruido eléctrico en cables largos; si lo montas lejos del Arduino, pon un capacitor cerámico de 100 nF entre 5V y GND pegado a sus patitas. Y no lo confundas con el TMP36, que se ve idéntico pero tiene

un offset de 500 mV (a 0 grados entrega 0.5V): la formula es distinta. Verifica que tu componente diga LM35.

El DHT11: temperatura y humedad en un módulo

El DHT11 es un módulo azul con una cajita plástica perforada que contiene el sensor. A diferencia del LM35, es digital: se comunica con el Arduino por un solo cable de datos usando un protocolo de tiempos precisos. Mide temperatura (0 a 50 grados, precisión de mas o menos 2 grados) y humedad relativa (20 a 90 por ciento, precisión de mas o menos 5 por ciento). Es menos preciso que el LM35 en temperatura, pero te da la humedad, que el LM35 no puede medir.

Para usarlo necesitas instalar una librería. La mas común es "DHT sensor library" de Adafruit, que se instala desde el Gestor de Librerías del Arduino IDE (busca "DHT" e instala la de Adafruit; también te pedirá la librería "Adafruit Unified Sensor", instalala cuando te lo pida). El código básico es: incluir la librería, crear el objeto DHT con el pin y el tipo (DHT11), iniciarlo en setup() con dht.begin(), y leer con dht.readTemperature() y dht.readHumidity(). Entre lecturas hay que esperar al menos 2 segundos, porque el sensor es lento: si lees mas rápido, te devuelve valores viejos o errores.

El DHT11 devuelve un código de error si algo falla (por ejemplo, si el cable de datos esta suelto). La buena práctica es verificar con isnan(): si isnan(temperatura), significa "no es un número", o sea, la lectura fallo; en ese caso no uses el valor. Verás esto en el ejemplo paso a paso.

Cuando elegir cada uno: LM35 si solo necesitas temperatura, quieres precisión decente y no quieres librerías; DHT11 (o su hermano mayor DHT22, mas preciso) si necesitas humedad o quieres un módulo fácil de conectar. Para el termómetro con alarma de esta unidad usaremos el LM35, y en la práctica compararas ambos.

Umbrales e histéresis: por que tu ventilador parpadea

Un umbral es un valor límite que dispara una acción: "si la temperatura pasa de 30 grados, enciende el ventilador". Suena simple y lógico. El problema aparece cuando la temperatura real esta justo en el límite, oscilando entre 29.9 y 30.1 por el ruido normal del sensor. Sin protección, el ventilador se enciende, se apaga, se enciende, se apaga, varias veces por segundo. Ese parpadeo frenetico se llama "repiqueteo" o "chattering", y no solo se ve mal: puede dañar reles, quemar motores por arranques repetidos y volver loco a cualquiera que lo escuche.

La solución es la histéresis: en lugar de un solo umbral, usas DOS umbrales separados. Para el ventilador: se enciende cuando la temperatura SUBE de 31 grados, pero no se apaga hasta que BAJA de 29 grados. Entre 29 y 31 hay una "zona muerta" donde el ventilador mantiene su estado anterior: si estaba encendido, sigue encendido; si estaba apagado, sigue apagado. Esa memoria del estado anterior es la esencia de la histéresis.

Vamos a simularlo paso a paso con numeros para que quede clarísimo. Umbrales: encender a 31, apagar a 29. La temperatura sube: 28 (apagado), 29 (apagado, todavía no llega a 31), 30 (apagado), 31.2 (ENCENDIDO). Ahora baja un poco por el aire del ventilador: 30.5 (sigue encendido, porque para apagarse debe bajar de 29), 29.8 (sigue encendido), 28.7 (APAGADO). Sin histéresis, con un solo umbral de 30, la secuencia 29.9, 30.1, 29.8, 30.2 habría causado cuatro cambios de estado; con histéresis, cero cambios inútiles. El sistema se vuelve estable y predecible.

En código, la histéresis se implementa con una variable de estado y dos comparaciones. La estructura típica es:

```
if (temperatura > UMBRAL_ALTO) {  
    ventiladorEncendido = true;  
}  
else if (temperatura < UMBRAL_BAJO) {  
    ventiladorEncendido = false;  
}  
digitalWrite(pinVentilador, ventiladorEncendido);
```

Fíjate en el detalle clave: no hay un else simple, hay un else if con el umbral bajo. Si la temperatura esta entre los dos umbrales, ninguna de las dos condiciones se cumple y la variable ventiladorEncendido conserva su valor anterior. Esa es toda la magia. La separación entre umbrales debe ser mayor que el ruido típico del sensor: si tu LM35 oscila mas o menos 0.5 grados, una separación de 2 grados es segura.

La histéresis no es solo para temperatura. La usarás en la lámpara automática de esta unidad (para que el LED no parpadee al atardecer, cuando la luz cruza el umbral lentamente), en termostatos, en cargadores de batería, en control de nivel de agua. Cada vez que una acción de encendido/apagado dependa de una medición analógica ruidosa, la histéresis es la respuesta profesional. Los termostatos de tu casa la usan: por eso el aire acondicionado no se prende y apaga cada diez segundos.

Errores comunes (y como evitarlos)

Error 1: leer un pin flotante. Si conectas `analogRead()` a un pin que no tiene nada conectado, te devuelve numeros aleatorios que cambian si acercas la mano. Eso no es un sensor fantasma: es un pin flotante captando ruido eléctrico del ambiente. Todo pin analógico debe estar conectado a algo con una referencia clara (un divisor de voltaje, la salida de un sensor). Si ves lecturas erráticas, lo primero que debes revisar es si el circuito esta completo y bien conectado a GND.

Error 2: no calibrar. Cada LDR es distinta, cada TCRT5000 se monta a distinta altura, cada cuarto tiene distinta luz. Los valores "mágicos" de los tutoriales (como "si lectura < 500 es de noche") casi nunca funcionan tal cual en tu montaje. La buena práctica es medir primero: sube un programa que solo imprima `analogRead()` en el monitor serie, observa los valores con luz y con oscuridad (tapa el sensor con la mano), y elige tus umbrales a partir de TUS numeros. Diez minutos de calibración ahorran horas de frustración.

Error 3: el ruido eléctrico. Los cables largos actúan como antenas y meten ruido en las lecturas analógicas; los motores y servos meten picos cuando arrancan. Si tus lecturas "saltan" mucho, hay tres remedios: promediar varias lecturas en el código (lee 10 veces y promedia), poner un capacitor de 100 nF entre el pin analógico y GND (filtra el ruido de alta frecuencia), y separar los cables de señal de los cables de potencia de los motores. El promedio por software es gratis y muy efectivo:

```
long suma = 0;
for (int i = 0; i < 10; i++) {
    suma += analogRead(A0);
    delay(5);
}
int lecturaSuave = suma / 10;
```

Error 4: olvidar el tiempo de estabilización del ADC. El convertidor del Arduino tarda unos 100 microsegundos por lectura, y si cambias rápidamente entre dos pines analógicos distintos, la primera lectura del segundo pin puede estar contaminada por el anterior. La práctica segura: cuando leas varios pines analógicos seguidos, haz una lectura de descarte al cambiar de pin, o deja un pequeño `delay(10)` entre lecturas de pines distintos. En la mayoría de los proyectos simples no notarás el problema, pero si lees LM35 y LDR alternadamente y los valores se "contagian", ya sabes por que.

Error 5: alimentar sensores desde pines digitales. Algunos tutoriales alimentan el TCRT5000 desde un pin digital para "ahorrar" pines de 5V. Mala idea: un pin digital del Arduino solo puede dar unos 20-40 mA de forma segura, y varios módulos juntos pueden excederlo y dañar la

placa. Alimenta los sensores desde el pin 5V (que puede dar mucha mas corriente) y usa los pines digitales solo para señales.

Error 6: confundir lógica normal con lógica invertida. Muchos módulos de sensores (infrarrojos de obstáculos, algunos TCRT5000) dan LOW cuando detectan y HIGH cuando no detectan. Si tu robot hace exactamente lo contrario de lo que esperas, no es que el sensor este roto: es que la lógica esta invertida. Verifícalo siempre con el LED indicador del módulo antes de escribir el código definitivo.

Buenas prácticas de esta unidad

Primera: calibra con el monitor serie antes de programar la lógica. Escribe primero el programa mínimo que imprime lecturas, observa, anota tus valores de día y de noche (o de blanco y negro), y después escribe el programa real con umbrales basados en tus mediciones.

Segunda: usa constantes con nombre para pines y umbrales (`#define` o `const int`). Si calibraste que `UMBRAL_OSCURIDAD` es 300 y luego cambias de cuarto, solo cambias una línea en lugar de cazar numeros mágicos por todo el código.

Tercera: aplica histéresis a todo control de encendido/apagado basado en sensores analógicos. No es opcional elegante: es lo que separa un prototipo que parpadea de un producto que funciona.

Cuarta: documenta tu calibración en un comentario del código ("calibrado el 19-sep: día=820, noche=120, umbral=400"). Tu yo del futuro te lo agradecerá cuando rearmes el proyecto dentro de tres meses.

Quinta: para sensores criticos, promedia lecturas. Diez lecturas con `delay(5)` tardan 50 milisegundos y eliminan la mayor parte del ruido. Tu robot no necesita reaccionar en microsegundos a la luz ambiente.

Con la teoría clara, es hora de ensuciarse las manos. En la siguiente sección construirás cuatro proyectos paso a paso, cada uno con su circuito, su código comentado línea por línea y su verificación.

Ejemplos paso a paso

Ejemplo A: medir la luz con una LDR y verla en el monitor serie

Este es el "hola mundo" de los sensores analógicos. Antes de automatizar nada, vas a ver con tus propios ojos como cambian los numeros cuando cambia la luz. Todo lo demás de esta unidad se construye sobre este experimento.

Material: Arduino Uno, una LDR, una resistencia de 10k ohmios, protoboard y cables.

Circuito (divisor de voltaje):

1. Conecta 5V del Arduino a una patita de la LDR.
2. Conecta la otra patita de la LDR al pin A0 del Arduino. Este es el punto medio del divisor.
3. Desde ese mismo punto (A0), conecta la resistencia de 10k hacia GND.
4. Listo: 5V -> LDR -> A0 -> resistencia 10k -> GND. Recuerda: con esta configuración, mas luz = lectura mas alta.

Código:

```

// Ejemplo A: medir luz con LDR
// Lee el pin A0 cada medio segundo y muestra la lectura
// cruda (0-1023), el voltaje y un porcentaje estimado.

const int PIN_LDR = A0;

void setup() {
  Serial.begin(9600);
  Serial.println("Medidor de luz con LDR");
  Serial.println("Lectura | Voltaje | Nivel");
}

void loop() {
  int lectura = analogRead(PIN_LDR);           // 0 a 1023
  float voltaje = lectura * (5.0 / 1023.0);    // conversion a voltios
  int porcentaje = map(lectura, 0, 1023, 0, 100); // 0 a 100 por ciento

  Serial.print(lectura);
  Serial.print(" | ");
  Serial.print(voltaje);
  Serial.print(" V | ");
  Serial.print(porcentaje);
  Serial.println(" %");

  delay(500);
}

```

Explicación del código: `analogRead(PIN_LDR)` devuelve el número crudo del ADC. La línea del voltaje aplica la formula que aprendiste: `lectura * (5.0 / 1023.0)`; el 5.0 con decimal obliga a Arduino a usar numeros con punto decimal (float). La función `map()` convierte el rango 0-1023 a un porcentaje 0-100 para que sea mas intuitivo. Todo se imprime en una línea cada 500 milisegundos.

Verificación: sube el programa y abre el monitor serie (lupa arriba a la derecha del IDE, a 9600 baudios). Apunta la LDR a la lámpara de tu cuarto: verás numeros altos (700-950). Ahora tapa la LDR con tu mano: los numeros deben caer (50-200). Si los numeros suben al taparla, invertiste el divisor (pusiste la resistencia arriba y la LDR abajo); funciona igual, solo invierte tu interpretación. Anota tus valores de "día" y "noche": los necesitarás para calibrar la lámpara automática del siguiente ejemplo. Esta es la calibración de la que hablamos: diez minutos mirando numeros te ahorran horas.

Ejemplo B: lámpara automática con histéresis

Ahora automatizamos: un LED que se enciende solo cuando oscurece y se apaga cuando amanece. Es el mismo principio de las lámparas de alumbrado publico y de los focos con sensor de las cocheras. Y le pondremos histéresis para que no parpadee al atardecer.

Material: lo del ejemplo A mas un LED y una resistencia de 220 ohmios.

Circuito:

1. Mantiene el divisor de la LDR del ejemplo A (5V -> LDR -> A0 -> 10k -> GND).
2. Conecta el pin 9 del Arduino a la resistencia de 220 ohmios, luego al anodo del LED (patita larga), y el catodo (patita corta) a GND.

Código:

```

// Ejemplo B: lampara automatica con histeresis
// El LED se enciende cuando oscurece y se apaga cuando aclara,
// con dos umbrales para evitar parpadeos en el limite.

const int PIN_LDR = A0;
const int PIN_LED = 9;

// Umbrales calibrados (ajustalos con TUS valores del ejemplo A):
const int UMBRAL_ENCENDER = 250; // si baja de esto, esta oscuro: prender
const int UMBRAL_APAGAR   = 350; // si sube de esto, hay luz: apagar

bool lamparaEncendida = false;

void setup() {
  pinMode(PIN_LED, OUTPUT);
  Serial.begin(9600);
  Serial.println("Lampara automatica lista");
}

void loop() {
  int luz = analogRead(PIN_LDR);

  // Histeresis: dos umbrales, la variable conserva su estado
  // cuando la lectura esta entre ambos.
  if (luz < UMBRAL_ENCENDER) {
    lamparaEncendida = true; // oscurecio: prender
  }
  else if (luz > UMBRAL_APAGAR) {
    lamparaEncendida = false; // aclaro: apagar
  }

  digitalWrite(PIN_LED, lamparaEncendida ? HIGH : LOW);

  Serial.print("Luz: ");
  Serial.print(luz);
  Serial.print(" | Lampara: ");
  Serial.println(lamparaEncendida ? "ENCENDIDA" : "apagada");

  delay(300);
}

```

Explicación del código: la lógica de histéresis es la que viste en la teoría. Si la lectura cae por debajo de 250, se considera de noche y la lámpara se prende. Si sube por encima de 350, se considera de día y se apaga. Entre 250 y 350 no pasa nada: la lámpara mantiene su estado

anterior. El operador ternario (condición ? valorSi : valorNo) es una forma compacta de escribir un if-else para asignar o imprimir; `lamparaEncendida ? HIGH : LOW` significa "si esta encendida, HIGH, si no, LOW".

Verificación: con los umbrales del ejemplo, tapa la LDR con la mano: el LED debe encenderse. Destapala: debe apagarse. Ahora haz la prueba crítica de la histéresis: acerca tu mano LENTAMENTE hasta dejar la lectura justo entre 250 y 350 (mirala en el monitor serie). El LED no debe parpadear: mantiene su estado. Si hubieras usado un solo umbral de 300, en esa zona el LED temblaría sin parar. Ajusta `UMBRAL_ENCENDER` y `UMBRAL_APAGAR` a tus valores reales: una buena regla es ponerlos alrededor del punto medio entre tu "día" y tu "noche", separados por unos 100 puntos.

Variante para practicar: cambia `digitalWrite` por `analogWrite` y haz que el brillo del LED sea inversamente proporcional a la luz (mas oscuridad = mas brillo). Pista: `brillo = map(luz, UMBRAL_APAGAR, UMBRAL_ENCENDER, 0, 255)` con `constrain()`. El pin 9 soporta PWM, así que funciona.

Ejemplo C: detector de línea con TCRT5000

Este es el primer paso hacia un robot seguidor de línea. Vas a usar el módulo TCRT5000 en modo digital: el te dirá HIGH o LOW según este sobre blanco o sobre negro, y tu lo calibraras con el potenciómetro.

Material: Arduino Uno, módulo TCRT5000 (el que trae comparador LM393 y potenciómetro azul), una cartulina blanca y cinta de aislar negra (o un marcador negro grueso).

Circuito:

1. VCC del módulo -> 5V del Arduino.
2. GND del módulo -> GND del Arduino.
3. D0 (salida digital) del módulo -> pin 7 del Arduino.
4. (Opcional para experimentar: A0 del módulo -> A0 del Arduino, para ver valores analógicos crudos.)

Calibración (hazla ANTES de subir el código definitivo):

1. Pega un tramo de cinta negra de unos 2 cm de ancho sobre la cartulina blanca. Esa es tu pista.
2. Sostén el módulo a unos 8 mm de la cartulina, apuntando hacia abajo.

3. Ponlo sobre la parte blanca y gira el potenciómetro con un desarmador pequeño hasta que el LED indicador del módulo cambie de estado. Prueba también sobre la cinta negra: el LED debe cambiar al estado contrario.
4. Deja el potenciómetro en el punto medio, donde la diferencia entre blanco y negro sea mas clara y estable.

Código:

```
// Ejemplo C: detector de linea con TCRT5000 (salida digital)
// Enciende el LED integrado (pin 13) cuando detecta la linea negra.

const int PIN_SENSOR = 7;    // D0 del modulo
const int PIN_LED = 13;      // LED integrado del Arduino

void setup() {
  pinMode(PIN_SENSOR, INPUT);
  pinMode(PIN_LED, OUTPUT);
  Serial.begin(9600);
  Serial.println("Detector de linea listo. Pasa el sensor sobre la pista.");
}

void loop() {
  int estado = digitalRead(PIN_SENSOR);

  // OJO: muchos modulos dan LOW sobre negro y HIGH sobre blanco.
  // Si el tuyo es al reves, invierte la comparacion.
  bool sobreLinea = (estado == LOW);

  digitalWrite(PIN_LED, sobreLinea ? HIGH : LOW);

  Serial.print("Sensor: ");
  Serial.print(estado);
  Serial.print(" | Linea negra: ");
  Serial.println(sobreLinea ? "SI" : "no");

  delay(200);
}
```

Explicación del código: es deliberadamente simple porque el módulo hace el trabajo pesado (el comparador convierte la reflectancia en HIGH/LOW según el umbral del potenciómetro). `digitalRead()` lee el pin 7. La variable `sobreLinea` traduce la lectura a algo legible. El LED del pin 13 te da retroalimentación visual inmediata sin mirar la computadora, útil cuando estés probando sobre la pista real.

Verificación: mueve el sensor sobre la cartulina: sobre el blanco el LED del Arduino debe estar apagado (y el monitor debe decir "no"); sobre la cinta negra debe encenderse ("SI"). Si esta al revés, cambia (estado == LOW) por (estado == HIGH). Si el LED parpadea en el borde de la cinta, tu umbral esta muy justo: recalibra el potenciómetro buscando mas margen. Prueba también variando la altura: a mas de 15 mm la detección se vuelve poco confiable; ese es el límite físico del sensor.

Experimento con la salida analógica: si conectaste A0 del módulo al A0 del Arduino, agrega `Serial.println(analogRead(A0))` y observa los numeros sobre blanco (altos, típicamente 800-1000) y sobre negro (bajos, típicamente 100-300). Con esos numeros podrías definir el umbral en código en lugar de usar el potenciómetro, que es lo que harás en uno de los ejercicios de práctica.

Ejemplo D: termómetro con LM35 y alarma de temperatura alta

Cerramos la sección con un instrumento de medición real: un termómetro digital que muestra la temperatura en el monitor serie y hace sonar un buzzer si pasa de cierto límite. Es la base de termostatos, incubadoras y sistemas de monitoreo.

Material: Arduino Uno, LM35, buzzer activo (el que suena solo con voltaje, de dos patitas), protoboard y cables.

Circuito:

1. Patita izquierda del LM35 (viendolo de frente, con la cara plana hacia ti) -> 5V. La del centro -> A1. La derecha -> GND. Verifica con la hoja de datos de tu LM35 si tienes dudas: el orden puede variar según el encapsulado, aunque lo común es VCC-salida-GND.
2. Buzzer: patita positiva (la mas larga, o marcada con +) -> pin 8 del Arduino; la otra -> GND. Si tu buzzer es pasivo (necesita `tone()`), usa `tone(8, 2000)` en lugar de `digitalWrite`.

Código:

```
// Ejemplo D: termometro con LM35 y alarma de temperatura alta
// Muestra la temperatura cada segundo. Si supera el umbral,
// suena el buzzer con histeresis para que no "tartamudee".

const int PIN_LM35 = A1;
const int PIN_BUZZER = 8;

// El LM35 entrega 10 mV por grado: cada unidad del ADC = 0.4883 grados.
const float MV_POR_GRADO = 10.0;
const float UMBRAL_ALARMA_ON = 30.0; // suena si pasa de 30 C
```



```

const float UMBRAL_ALARMA_OFF = 28.0; // calla si baja de 28 C

bool alarmaActiva = false;

float leerTemperatura() {
    // Promedia 10 lecturas para filtrar el ruido electrico.
    long suma = 0;
    for (int i = 0; i < 10; i++) {
        suma += analogRead(PIN_LM35);
        delay(5);
    }
    float lectura = suma / 10.0;
    float voltaje = lectura * (5.0 / 1023.0); // voltios
    float tempC = (voltaje * 1000.0) / MV_POR_GRADO; // mV entre 10 mV/C
    return tempC;
}

void setup() {
    pinMode(PIN_BUZZER, OUTPUT);
    Serial.begin(9600);
    Serial.println("Termometro LM35 listo");
}

void loop() {
    float temp = leerTemperatura();

    // Histeresis: prende arriba de 30, apaga debajo de 28.
    if (temp > UMBRAL_ALARMA_ON) {
        alarmaActiva = true;
    }
    else if (temp < UMBRAL_ALARMA_OFF) {
        alarmaActiva = false;
    }
    digitalWrite(PIN_BUZZER, alarmaActiva ? HIGH : LOW);

    Serial.print("Temperatura: ");
    Serial.print(temp, 1); // un decimal
    Serial.print(" C");
    if (alarmaActiva) {
        Serial.print(" *** ALARMA: temperatura alta ***");
    }
    Serial.println();

    delay(1000);
}

```

Explicación del código: la función leerTemperatura() encapsula toda la cadena de conversión: promedia 10 lecturas del ADC (filtro de ruido), convierte a voltaje con la formula conocida, pasa a milivoltios multiplicando por 1000 y divide entre 10 (los mV por grado del LM35). Ejemplo numérico de verificación: si el promedio es 51, voltaje = $51 * 5.0/1023.0 = 0.249V$; en mV = 249; entre 10 = 24.9 C. La histéresis de 30/28 evita que el buzzer tartamudee cuando la temperatura ronda los 30 grados. Serial.print(temp, 1) imprime con un decimal.

Verificación: deja el circuito quieto un minuto y anota la temperatura: debe estar cerca de la temperatura ambiente real (24-28 C en un cuarto normal). Ahora calienta el LM35 sujetandolo entre tus dedos unos 20 segundos: verás como la lectura sube grado por grado hasta pasar de 30 y el buzzer suena. Sueltalo y sopla suavemente: bajara y el buzzer callara al cruzar 28. Si la lectura es absurda (por ejemplo, -40 o 150), revisa el cableado: lo mas común es confundir las patitas del LM35 o tener el pin analógico flotante. Si la lectura oscila mas de un grado, acerca los cables y asegúrate de que el LM35 este bien clavado en el protoboard.

Con estos cuatro ejemplos ya tienes el kit completo de la unidad: medir luz, automatizar con luz, detectar líneas y medir temperatura con alarma. En la práctica los combinaraas y los llevaras mas lejos.

Práctica

Los ejemplos anteriores te dieron las piezas; estos ejercicios te obligan a combinarlas y a tomar decisiones de diseño como lo haría un ingeniero. Haz cada ejercicio de verdad en tu protoboard antes de mirar la solución: la solución esta para verificar y para aprender el "como", pero el aprendizaje real ocurre cuando tu calibración no funciona a la primera y tienes que pensar por que.

Ejercicio 1: medidor de luz con barra de LEDs

Enunciado: construye un "vumetro" de luz con 5 LEDs en fila. A mas luz ambiente, mas LEDs encendidos; en oscuridad total, todos apagados. Usa la LDR con su divisor de voltaje en A0 y los LEDs en los pines 3, 4, 5, 6 y 7 (con sus resistencias de 220 ohmios).

Objetivo: practicar el mapeo de un rango analógico a un número discreto de salidas, y calibrar con tus propios valores de día y noche.

Pista: primero mide tus valores reales de día y de noche con el programa del Ejemplo A. Luego usa map() para convertir la lectura a un número de 0 a 5 LEDs, con constrain() para no salirte del rango. Un for enciende los primeros N LEDs y apaga el resto.

Solución del ejercicio 1

```
// Solucion 1: medidor de luz con barra de 5 LEDs
const int PIN_LDR = A0;
const int LEDS[] = {3, 4, 5, 6, 7};
const int NUM_LEDS = 5;

// Calibra estos dos valores con TU LDR y TU cuarto:
const int LECTURA_OSCURIDAD = 80; // lectura típica a oscuras
const int LECTURA_PLENA_LUZ = 850; // lectura típica con buena luz

void setup() {
  for (int i = 0; i < NUM_LEDS; i++) {
    pinMode(LEDS[i], OUTPUT);
  }
  Serial.begin(9600);
}

void loop() {
  int luz = analogRead(PIN_LDR);

  // Convierte la lectura a "cuantos LEDs prender" (0 a 5).
  int nivel = map(luz, LECTURA_OSCURIDAD, LECTURA_PLENA_LUZ, 0, NUM_LEDS);
  nivel = constrain(nivel, 0, NUM_LEDS);

  // Prende los primeros 'nivel' LEDs, apaga el resto.
  for (int i = 0; i < NUM_LEDS; i++) {
    digitalWrite(LEDS[i], (i < nivel) ? HIGH : LOW);
  }

  Serial.print("Luz: ");
  Serial.print(luz);
  Serial.print(" -> LEDs: ");
  Serial.println(nivel);

  delay(200);
}
```

Explicación: el arreglo `LEDS[]` guarda los pines para manejarlos con un `for`, una técnica que te ahorra repetir cinco veces lo mismo. `map()` traduce tu rango calibrado (oscuridad a plena luz) al rango 0-5; `constrain()` evita que un valor fuera de rango encienda un LED fantasma número 6 o de un número negativo. El `for` final compara el índice `i` con el `nivel`: los LEDs con índice menor que el `nivel` se prenden. Si tu barra se comporta al revés (mas LEDs con menos luz), invertiste el divisor o tus constantes de calibración están cruzadas: revisa ambas.

Ejercicio 2: seguidor de luz simple con dos LDR

Enunciado: monta dos LDR separadas unos 10 cm (una a la izquierda, una a la derecha), cada una con su divisor de voltaje (A0 y A1). Dos LEDs indican la dirección: el LED izquierdo (pin 8) se enciende si hay mas luz a la izquierda, el derecho (pin 9) si hay mas luz a la derecha. Si la diferencia es pequeña (menos de 60 puntos), se enciende un tercer LED central (pin 10) indicando "luz de frente". Es el principio de los robots que buscan la luz, sin motores todavía.

Objetivo: comparar dos sensores analógicos, trabajar con diferencias y una zona muerta (prima hermana de la histéresis).

Solución del ejercicio 2

```
// Solucion 2: seguidor de luz con dos LDR (indicacion con LEDs)
const int PIN_LDR_IZQ = A0;
const int PIN_LDR_DER = A1;
const int LED_IZQ = 8;
const int LED_DER = 9;
const int LED_CENTRO = 10;

// Diferencia minima para decidir que la luz viene de un lado.
// Funciona como zona muerta: evita que los LEDs tiemblen.
const int UMBRAL_DIFERENCIA = 60;

void setup() {
  pinMode(LED_IZQ, OUTPUT);
  pinMode(LED_DER, OUTPUT);
  pinMode(LED_CENTRO, OUTPUT);
  Serial.begin(9600);
  Serial.println("Busca la lampara: mueve una luz de un lado a otro.");
}

void loop() {
  int izq = analogRead(PIN_LDR_IZQ);
  int der = analogRead(PIN_LDR_DER);
  int diferencia = izq - der;    // positivo: mas luz a la izquierda

  // Apaga todo primero (estado limpio en cada ciclo).
  digitalWrite(LED_IZQ, LOW);
  digitalWrite(LED_DER, LOW);
  digitalWrite(LED_CENTRO, LOW);

  if (diferencia > UMBRAL_DIFERENCIA) {
    digitalWrite(LED_IZQ, HIGH);    // luz a la izquierda
```

```

}
else if (diferencia < -UMBRAL_DIFERENCIA) {
    digitalWrite(LED_DER, HIGH);    // luz a la derecha
}
else {
    digitalWrite(LED_CENTRO, HIGH); // luz de frente (o pareja)
}

Serial.print("Izq: ");
Serial.print(izq);
Serial.print(" Der: ");
Serial.print(der);
Serial.print(" Dif: ");
Serial.println(diferencia);

delay(200);
}

```

Explicación: la variable diferencia resume todo: su signo dice de que lado hay mas luz y su magnitud dice cuanta mas. La zona muerta de 60 puntos evita que los LEDs cambien freneticamente cuando la luz esta pareja; es el mismo espiritu de la histéresis aplicado a una comparación. Nota que este ejercicio lee dos pines analógicos seguidos (A0 y A1): si notas que los valores se "contagian" (uno sube cuando tapas el otro), agrega una lectura de descarte: lee A0 dos veces y quédate con la segunda. Prueba con la lámpara de tu celular moviendola de izquierda a derecha frente a las LDR: el LED encendido debe "seguir" a la luz. Este es exactamente el algoritmo que usará tu primer robot móvil fototropico en la Unidad 7, solo que en lugar de LEDs movera motores.

Ejercicio 3: contador de objetos con sensor infrarrojo

Enunciado: usa el módulo sensor de obstáculos IR (o un TCRT5000 apuntando a un punto fijo) como "puerta" contadora: cada vez que un objeto pasa frente al sensor, incrementa un contador y muéstralo en el monitor serie. Agrega un LED (pin 13) que parpadee brevemente con cada conteo y un botón (pin 2, con resistencia pull-down) que reinicie el contador a cero. El reto es contar cada objeto UNA sola vez aunque el sensor lo vea durante varios ciclos del loop.

Objetivo: detectar flancos (cambios de estado) en lugar de niveles, la técnica base de todos los contadores y encoders.

Solución del ejercicio 3

```

// Solucion 3: contador de objetos con IR y boton de reinicio
const int PIN_IR = 7;      // salida digital del modulo
const int PIN_LED = 13;
const int PIN_BOTON = 2;

int contador = 0;
int estadoAnterior = HIGH; // sin objeto al inicio (logica tipica)

void setup() {
  pinMode(PIN_IR, INPUT);
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_BOTON, INPUT);
  Serial.begin(9600);
  Serial.println("Contador listo. Pasa objetos frente al sensor.");
  Serial.println("Pulsa el boton para reiniciar.");
}

void loop() {
  int estado = digitalRead(PIN_IR);

  // Deteccion de flanco: solo cuenta en la TRANSICION de
  // "sin objeto" (HIGH) a "objeto detectado" (LOW).
  if (estado == LOW && estadoAnterior == HIGH) {
    contador++;
    Serial.print("Objeto #");
    Serial.println(contador);

    // Parpadeo de confirmacion.
    digitalWrite(PIN_LED, HIGH);
    delay(150);
    digitalWrite(PIN_LED, LOW);
  }
  estadoAnterior = estado;

  // Boton de reinicio (con antirrebote simple).
  if (digitalRead(PIN_BOTON) == HIGH) {
    contador = 0;
    Serial.println("Contador reiniciado a 0.");
    delay(300); // evita reinicios multiples por rebote
  }

  delay(20);
}

```

Explicación: el truco esta en estadoAnterior. Si contaras cada vez que el sensor ve el objeto (cada ciclo del loop), un objeto que tarda medio segundo en pasar se contaria 25 veces. Al contar solo la transicion HIGH->LOW (el flanco de bajada), cada objeto suma exactamente 1, sin importar cuanto tiempo bloquee el sensor. Esta técnica de "detectar el cambio, no el estado" es la misma que usan los encoders de los motores para medir velocidad y distancia, así que memorízala. Ajusta la lógica si tu módulo esta invertido (algunos dan HIGH al detectar): cambia la condición a (estado == HIGH && estadoAnterior == LOW). Calibra la distancia con el potenciómetro del módulo para que detecte tu objeto de prueba (una caja pequeña, tu mano) a unos 10-15 cm.

Ejercicio 4: termostato con dos umbrales

Enunciado: construye un termostato con el LM35 que controle un "ventilador" (simulado con un LED en el pin 9, o un motor DC pequeño con transistor si ya viste la Unidad 3). El ventilador se enciende a 30 C y se apaga a 27 C (histéresis de 3 grados). Además, muestra la temperatura en el monitor serie una vez por segundo con el estado del ventilador. Como reto extra, haz que el LED parpadee rápido mientras la temperatura este subiendo hacia el umbral (pista: compara la lectura actual con la anterior).

Objetivo: aplicar histéresis a un control real de temperatura y practicar la comparación de tendencias.

Solución del ejercicio 4

```
// Solucion 4: termostato con histeresis (LM35 + ventilador/LED)
const int PIN_LM35 = A1;
const int PIN_VENTILADOR = 9;

const float TEMP_ENCENDER = 30.0; // prende el ventilador
const float TEMP_APAGAR = 27.0;  // apaga el ventilador

bool ventiladorOn = false;
float tempAnterior = 0;
bool primeraLectura = true;

float leerTemperatura() {
    long suma = 0;
    for (int i = 0; i < 10; i++) {
        suma += analogRead(PIN_LM35);
        delay(5);
    }
    float voltaje = (suma / 10.0) * (5.0 / 1023.0);
```



```

    return (voltaje * 1000.0) / 10.0; // 10 mV por grado
}

void setup() {
    pinMode(PIN_VENTILADOR, OUTPUT);
    Serial.begin(9600);
    Serial.println("Termostato listo: ON a 30 C, OFF a 27 C.");
}

void loop() {
    float temp = leerTemperatura();

    // Histeresis del termostato.
    if (temp > TEMP_ENCENDER) {
        ventiladorOn = true;
    }
    else if (temp < TEMP_APAGAR) {
        ventiladorOn = false;
    }

    // Extra: parpadeo de aviso si la temperatura viene subiendo
    // y todavia no llega al umbral de encendido.
    bool subiendo = !primeraLectura && (temp > tempAnterior + 0.2);
    tempAnterior = temp;
    primeraLectura = false;

    if (ventiladorOn) {
        digitalWrite(PIN_VENTILADOR, HIGH);
    }
    else if (subiendo && temp > TEMP_ENCENDER - 2.0) {
        // Parpadea cuando esta cerca del umbral y subiendo.
        digitalWrite(PIN_VENTILADOR, (millis() / 250) % 2);
    }
    else {
        digitalWrite(PIN_VENTILADOR, LOW);
    }

    Serial.print("Temp: ");
    Serial.print(temp, 1);
    Serial.print(" C | Ventilador: ");
    Serial.print(ventiladorOn ? "ON " : "OFF");
    if (subiendo) Serial.print(" (subiendo)");
    Serial.println();

    delay(1000);
}

```

```
}
```

Explicación: la histéresis de 3 grados (30 para prender, 27 para apagar) es mas amplía que la del ejemplo D a propósito: un ventilador real tarda en enfriar, y una banda angosta lo prendería y apagaría con frecuencia. La detección de tendencia compara la lectura actual con la anterior exigiendo al menos +0.2 grados para declarar "subiendo", lo que filtra el ruido pequeño. El parpadeo usa `millis() / 250 % 2`: cada 250 ms alterna entre 0 y 1, un truco sin `delay()` que no bloquea el programa. Para probarlo sin esperar al verano, calienta el LM35 con tus dedos como en el ejemplo D y observa la secuencia: aviso parpadeante, encendido a 30, apagado al bajar de 27.

Ejercicio 5: registrar la temperatura cada minuto

Enunciado: escribe un programa que lea la temperatura con el LM35 (o el DHT11 si prefieres practicar librerías) una vez por minuto y la imprima en el monitor serie con una marca de tiempo en formato "minuto N: XX.X C". Debe funcionar con `millis()` (sin `delay()` largos que bloqueen), llevar la cuenta de cuantas lecturas lleva, y calcular al final de cada 10 lecturas el promedio de esas 10. Déjalo corriendo 10 minutos y observa como varía la temperatura de tu cuarto.

Objetivo: practicar temporizacion no bloqueante con `millis()`, acumulacion de datos y calculo de promedios por lotes.

Solución del ejercicio 5

```
// Solucion 5: registrador de temperatura cada minuto (no bloqueante)
const int PIN_LM35 = A1;
const unsigned long INTERVALO = 60000UL; // 1 minuto en milisegundos

unsigned long ultimaLectura = 0;
unsigned long minutos = 0;
float acumulado = 0;
int muestrasLote = 0;

float leerTemperatura() {
    long suma = 0;
    for (int i = 0; i < 10; i++) {
        suma += analogRead(PIN_LM35);
        delay(5);
    }
    float voltaje = (suma / 10.0) * (5.0 / 1023.0);
    return (voltaje * 1000.0) / 10.0;
}
```

```

void setup() {
  Serial.begin(9600);
  Serial.println("Registrador de temperatura iniciado.");
  Serial.println("Una lectura por minuto. Promedio cada 10 lecturas.");
  Serial.println("-----");
}

void loop() {
  unsigned long ahora = millis();

  // Solo actua cuando paso un minuto desde la ultima lectura.
  if (ahora - ultimaLectura >= INTERVALO) {
    ultimaLectura = ahora;
    minutos++;

    float temp = leerTemperatura();
    acumulado += temp;
    muestrasLote++;

    Serial.print("Minuto ");
    Serial.print(minutos);
    Serial.print(": ");
    Serial.print(temp, 1);
    Serial.println(" C");

    // Cada 10 lecturas, reporta el promedio del lote.
    if (muestrasLote >= 10) {
      float promedio = acumulado / muestrasLote;
      Serial.print(">>> Promedio de 10 min: ");
      Serial.print(promedio, 1);
      Serial.println(" C");
      Serial.println("-----");
      acumulado = 0;
      muestrasLote = 0;
    }
  }

  // Aquí podrias hacer otras tareas: el loop no se bloquea.
}

```

Explicación: la técnica de `millis()` es el patrón mas importante de Arduino después del `loop` mismo: en lugar de detener todo con `delay(60000)`, guardas el momento de la última lectura y en cada vuelta preguntas "ya paso un minuto?". La resta `ahora - ultimaLectura` funciona incluso cuando `millis()` se reinicia (desborda) cada 50 días, gracias a la aritmética de `unsigned`

long. El sufijo UL en 60000UL le dice al compilador que el número es unsigned long, evitando un desbordamiento silencioso de int (60000 no cabe en un int de 16 bits con signo, cuyo máximo es 32767; sin el UL el intervalo quedaría mal). El acumulador y el contador por lote permiten el promedio cada 10 lecturas, y se reinician después de reportar. Para probar sin esperar 10 minutos reales, cambia INTERVALO a 5000UL (5 segundos) durante las pruebas y regresalo a 60000UL para la corrida final.

Variante con DHT11: si quieres practicar la librería, reemplaza leerTemperatura() por lecturas con dht.readTemperature(), verifica con isnan() y respeta los 2 segundos mínimos entre lecturas (con un minuto de intervalo vas sobrado). Agrega también dht.readHumidity() y registra "Minuto N: 24.5 C, 55 % HR" para tener una mini estación meteorológica.

Resumen de la unidad

En esta unidad cruzaste la frontera entre el mundo digital y el mundo analógico, y eso te convierte en un constructor de robots mucho más capaz. Aprendiste que el Arduino Uno lee valores continuos gracias a su convertidor analógico-digital de 10 bits: analogRead() traduce un voltaje de 0 a 5V en un número de 0 a 1023, donde cada paso vale unos 4.88 mV. Con la fórmula $\text{voltaje} = \text{lectura} * 5.0 / 1023.0$ puedes convertir cualquier lectura a su voltaje real, la base de todas las mediciones de esta unidad.

Dominaste el divisor de voltaje ($V_{\text{valida}} = V_{\text{entrada}} * R2 / (R1 + R2)$), el circuito que convierte la resistencia variable de una LDR en un voltaje que el Arduino puede leer. Viste con números reales como una LDR de 1k a plena luz da unos 4.55V (lectura ~931) y de 1M en oscuridad da 0.05V (lectura ~10), y aprendiste a usar map() y constrain() para traducir lecturas crudas a porcentajes, brillos de LED y niveles útiles.

Conociste el TCRT5000, el sensor infrarrojo de reflectancia que distingue una línea negra sobre fondo blanco midiendo el rebote de luz infrarroja, y aprendiste a calibrar su potenciómetro buscando el punto medio entre la respuesta al blanco y al negro. Entendiste la diferencia con los sensores de obstáculos IR, que miran al frente y detectan presencia a centímetros de distancia, con sus limitaciones (superficies negras, vidrio, sol directo).

En temperatura manejaste dos enfoques: el LM35 analógico, con su regla de oro de 10 mV por grado y conversión directa ($\text{temperatura} = \text{lectura} * 0.4883$), y el DHT11 digital, que entrega temperatura y humedad por un solo cable usando una librería, con su espera mínima de 2 segundos entre lecturas y su verificación con isnan().

El concepto de control más valioso de la unidad fue la histéresis: usar dos umbrales en lugar de uno para que los sistemas de encendido/apagado no parpadeen en el límite. La aplicaste

en la lámpara automática, en la alarma de temperatura y en el termostato, y viste que la misma idea aparece como "zona muerta" al comparar dos sensores.

También incorporaste el método profesional: calibrar con el monitor serie antes de programar la lógica, usar constantes con nombre, promediar lecturas para filtrar ruido, detectar flancos en lugar de niveles para contar objetos, y temporizar con `millis()` sin bloquear el programa.

Lo que construiste: un medidor de luz con monitor serie, una lámpara automática con histéresis, un detector de línea calibrado, un termómetro con alarma, y en la práctica una barra de LEDs medidora, un seguidor de luz de dos LDR, un contador de objetos IR, un termostato completo y un registrador de temperatura por minutos. En la Unidad 6 sumaras el sentido de la distancia con el sensor ultrasónico HC-SR04 y tu robot empezara a "ver" el espacio que lo rodea.

UNIDAD 6: MOTORES DC Y CONTROL DE VELOCIDAD

En las unidades anteriores de este curso aprendiste a darle ojos y oídos a tu robot. Conectaste sensores que miden distancia, detectan luz, sienten el tacto y escuchan comandos. Todo eso es muy valioso, pero hay un detalle importante: un robot que solo siente y piensa, pero no se mueve, es básicamente una computadora con antenas. Para que un robot de verdad sea un robot, necesita moverse, empujar, girar, avanzar, retroceder y reaccionar con su cuerpo ante lo que sus sensores le dicen. En esta unidad vas a darle músculos a tu robot.

Los actuadores son los músculos de cualquier sistema robótico. Así como tus músculos convierten la energía química de tu cuerpo en movimiento, los actuadores convierten la energía eléctrica en movimiento mecánico. Existen muchos tipos de actuadores: servomotores, motores paso a paso, solenoides, actuadores lineales, bombas y hasta músculos artificiales neumáticos. Cada uno tiene su lugar. Pero si hablamos de robots móviles, de carritos que avanzan, de brazos sencillos que giran y de ventiladores, el rey indiscutible es el motor de corriente directa, el famoso motor DC. Es barato, es fácil de conseguir, es sencillo de entender y con el puedes construir desde un carrito seguidor de líneas hasta un brazo robótico de dos ejes. Por eso esta unidad entera está dedicada a él: a entenderlo, a conectarlo de forma segura y a controlarlo con precisión.

Piensa en todos los aparatos que usan motores DC a tu alrededor. El carrito de juguete de control remoto que corre por la sala, el ventilador que te refresca en verano, el taladro eléctrico, la licuadora, el cepillo de dientes eléctrico, el dron que vuela en el parque. Todos ellos, en el fondo, dependen del mismo principio físico que vas a dominar en esta unidad. Cuando termines, no solo sabrás hacer girar un motorcito con tu Arduino: entenderás el principio que mueve a media industria.

Antes de seguir, hay una advertencia de seguridad que debes grabarte a fuego, porque es el error número uno de los principiantes y puede destruir tu Arduino en cuestión de segundos: **NUNCA** conectes un motor DC directamente a un pin del Arduino. Repítelo conmigo: nunca. Ni para probar rapidito, ni un segundito, ni aunque el motor sea chiquito. Te explico por qué con números. Cada pin digital del Arduino Uno puede entregar como máximo 40 miliamperios de corriente, y en la práctica conviene no pasar de 20 miliamperios para trabajar con margen de seguridad. Un motor DC pequeño, de los que se usan en carritos, consume entre 100 y 300 miliamperios cuando gira libre, y cuando arranca o cuando se atora, su consumo se dispara: la llamada corriente de arranque puede llegar a 1 o 2 amperios, es decir, entre 25 y 50 veces más de lo que un pin del Arduino puede dar. Si conectas el motor directo al pin, el pin intenta

entregar esa corriente, se sobrecalienta y se quema. Y cuando un pin se quema, muchas veces se lleva consigo el microcontrolador completo. Tu placa Arduino, que te costo dinero y tiempo conseguir, queda inservible por un descuido de cinco segundos. Por eso en esta unidad vas a aprender a usar un controlador externo, un módulo que se encarga de la parte pesada: el módulo toma la energía de una fuente separada, como una batería o un eliminador, y el Arduino solo le manda señales de control, señales débiles que no ponen en riesgo a la placa. El Arduino es el cerebro que da órdenes; el controlador es el músculo que ejecuta. Esa separación de responsabilidades es una de las ideas mas importantes de toda la robótica.

Ahora bien, controlar un motor no es solo prenderlo y apagarlo. Un control de verdad tiene dos dimensiones: la dirección y la velocidad. La dirección significa poder hacer que el motor gire hacia adelante o hacia atrás cuando tu quieras. Esto se logra invirtiendo la polaridad, es decir, cambiando cual terminal del motor recibe el positivo y cual recibe el negativo. Para hacer eso de forma electrónica, sin andar cambiando cables a mano, existe un circuito genial llamado puente H, que vas a estudiar a fondo en esta unidad. La velocidad, por su parte, se controla con una técnica llamada PWM, modulación por ancho de pulso, que le permite al Arduino simular un voltaje variable usando solo señales digitales de encendido y apagado muy rápidas. Con PWM puedes hacer que tu carrito avance despacio para no chocar, o a toda velocidad en línea recta, o que acelere suavemente como un coche de verdad en lugar de arrancar de golpe.

Lo que vas a construir en esta unidad es el sistema de control completo de un robot móvil. Empezarás entendiendo como funciona un motor DC por dentro, con imanes y bobinas, en una explicación simple y sin matemáticas complicadas. Después estudiaras el puente H, ese circuito con nombre curioso que es la base de todo control de dirección. Luego conocerás al módulo L298N, el controlador mas popular del mundo maker, aprenderás cada uno de sus pines y como conectarlo sin errores. Dominarás el PWM y la función `analogWrite()`, y aprenderás por que los robots bien hechos arrancan con rampas de aceleracion suaves en lugar de brincar. Verás los errores mas comunes que cometen los principiantes, como olvidar el GND común o alimentar el motor desde el Arduino, para que tu no los cometas. Y en la parte práctica vas a programar cinco ejemplos paso a paso: girar el motor en una dirección, controlar adelante y atrás con botones, regular la velocidad con un potenciómetro, crear rampas de aceleracion suaves y controlar dos motores al mismo tiempo, que es exactamente la base de un carrito robótico. Al final tendrás ejercicios para practicar por tu cuenta, con soluciones completas.

Esta unidad cubre las clases 16, 17 y 18 del curso. La clase 16 se enfoca en el motor DC: su principio de funcionamiento, sus especificaciones como voltaje y RPM, y por que necesita un controlador externo. La clase 17 se dedica al puente H y al módulo L298N: la teoría, las

conexiones completas, la alimentación separada del motor y el truco del GND común. La clase 18 entra de lleno al control de velocidad con PWM: la función `analogWrite()`, las rampas de aceleración y el control de dos motores para mover un chasis. Al terminar esta unidad tendrás entre tus manos una de las habilidades más pedidas en la robótica de hobby y en la industria: mover cosas con precisión usando un microcontrolador.

Un último consejo antes de empezar: ten a la mano tu Arduino Uno, un motor DC pequeño de 3 a 6 voltios, un módulo L298N, una fuente de alimentación separada para el motor (puede ser un portapilas con 4 pilas AA o un eliminador de 6 a 9 voltios), cables jumper, un protoboard, un potenciómetro y un par de botones. No necesitas comprar todo hoy mismo para leer la teoría, pero cuando llegues a los ejemplos, nada se compara con conectar los cables tu mismo, subir el código y ver el motor girar obedeciendo tus órdenes. Esa sensación, la primera vez que un motor gira porque TU lo programaste, es la que engancha a la robótica para siempre. Vamos a ello.

Explicación detallada

Como funciona un motor DC por dentro

Un motor DC es una máquina sorprendentemente simple. Por fuera ves una cajita metálica cilíndrica con dos terminales y un eje que sale por un extremo. Por dentro hay básicamente dos cosas: un imán permanente y una bobina de alambre de cobre enrollado. El imán genera un campo magnético fijo, y la bobina, cuando le pasa corriente eléctrica, se convierte en un electroimán, es decir, en un imán temporal cuyo campo magnético puedes encender y apagar con la corriente.

Aquí viene la magia de la física: cuando acercas dos imanes, los polos opuestos se atraen y los polos iguales se repelen. La bobina está montada en un rotor, una pieza que puede girar libremente sobre el eje del motor. Al pasar corriente por la bobina en una dirección, uno de sus extremos se vuelve polo norte y el otro polo sur. El polo norte de la bobina es repelido por el polo norte del imán fijo y atraído por el polo sur. Ese empujón hace girar al rotor un poquito. Justo cuando el rotor está por llegar a la posición de equilibrio, un componente llamado conmutador invierte automáticamente la dirección de la corriente en la bobina. Al invertirse la corriente, se invierten los polos de la bobina, y el empujón magnético continúa en la misma dirección de giro. Este ciclo se repite cientos o miles de veces por segundo, y el resultado es un giro continuo del eje. Así de simple: imán fijo más bobina con corriente conmutada igual a rotación.

Hay dos consecuencias prácticas importantísimas de este principio. La primera: si inviertes la polaridad de la alimentación, es decir, si cambias el positivo y el negativo de lugar en las terminales del motor, la corriente circula al revés por la bobina, los polos magnéticos se invierten y el motor gira en dirección contraria. Esto es lo que nos permite controlar la dirección de giro electrónicamente: basta con poder intercambiar la polaridad a voluntad. La segunda: mientras más voltaje le apliques al motor, más corriente circula por la bobina, más fuerte es el campo magnético del electroimán y más rápido gira el eje. Por eso el voltaje controla la velocidad, como verás en la siguiente sección.

Los motores DC que usamos en robótica de hobby suelen ser de 3 a 12 voltios. Los más comunes en carritos son los motorreductores amarillos de 3 a 6 voltios, esos que venden en kits de Arduino. Traen una cajita de engranes reductores que baja la velocidad y aumenta la fuerza, porque un motor DC puro gira rapidísimo pero con muy poca fuerza. La reducción de engranes es lo que le da al carrito la fuerza para moverse con el peso de las baterías encima.

Voltaje, velocidad y corriente: las tres especificaciones que mandan

Todo motor DC trae especificaciones que debes saber leer. Las tres más importantes son el voltaje nominal, la velocidad en RPM y la corriente. El voltaje nominal es el voltaje para el que fue diseñado el motor. Si tu motor dice 6V, lo ideal es alimentarlo con 6 voltios. Puedes darle un poco menos y simplemente girará más despacio; puedes darle un poco más y girará más rápido, pero se calentará más y su vida útil se acorta. Como regla práctica, no pases del voltaje nominal por más de un 20 por ciento si quieres que el motor te dure.

La velocidad se mide en RPM, revoluciones por minuto. Un motor DC pequeño sin reducción puede girar a 5000 o 10000 RPM, rapidísimo pero sin fuerza. Un motorreductor de carrito típico gira a 100 o 200 RPM en su eje de salida: mucho más lento, pero con la fuerza suficiente para mover el carrito. Cuando elijas un motor para tu robot, piensa en que necesitas más: velocidad o fuerza. Para un carrito que explora, 100 a 200 RPM está perfecto. Para un ventilador, quieres muchas RPM. Para un brazo robótico que levanta cosas, quieres pocas RPM y mucha fuerza, lo que se llama torque.

La corriente es la especificación más crítica para la seguridad de tu circuito. Todo motor tiene una corriente nominal, la que consume girando libre sin carga, y una corriente de arranque o de rotor bloqueado, la que consume en el instante de arrancar o cuando el eje se atora. La corriente de arranque puede ser de 3 a 10 veces la corriente nominal. Un motorreductor amarillo típico consume unos 150 miliamperios girando libre, pero al arrancar puede pedir 800 miliamperios o más, y si lo atoras con la mano puede pasar de 1 amperio. Por eso tu fuente de alimentación del motor y tu controlador deben soportar esa corriente pico, no solo la nominal. Si tu fuente se queda corta, el motor arrancará débil o la fuente se protegerá y se apagará. Y

por eso, repitiendo la advertencia de la introducción, un pin del Arduino que solo da 40 miliamperios jamás podrá mover un motor: necesitas un controlador externo alimentado por separado.

Hay una relación casi lineal entre el voltaje aplicado y la velocidad del motor: a la mitad del voltaje, mas o menos la mitad de la velocidad. Esta relación es la que explotamos con el PWM para controlar la velocidad desde el Arduino, como verás mas adelante. También existe una relación entre la carga y la corriente: mientras mas trabajo le pidas al motor, mas corriente consume para mantener la velocidad. Si tu carrito sube una rampa, los motores piden mas corriente. Diseña siempre con margen.

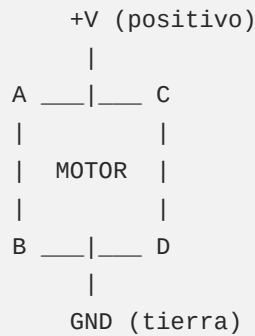
El puente H: el circuito que invierte la polaridad

Ya sabes que para cambiar la dirección de giro de un motor DC basta con invertir la polaridad en sus terminales. Podrías hacerlo a mano cambiando los cables de lugar, pero un robot necesita hacerlo solo, electrónicamente y muchas veces por segundo si hace falta. El circuito que logra esto se llama puente H, y es una de las ideas mas elegantes de la electrónica.

Se llama puente H porque su diagrama tiene forma de la letra H. Imagínalo así: el motor es la barra horizontal del centro de la H. En cada una de las cuatro puntas de la H hay un interruptor, que en la práctica son transistores. Los dos interruptores de arriba se conectan al positivo de la alimentación, y los dos de abajo se conectan a tierra, al negativo. El motor queda en medio, conectado entre el lado izquierdo y el lado derecho del puente.

Para que el motor gire en una dirección, cierras el interruptor de arriba a la izquierda y el de abajo a la derecha. La corriente entra al motor por la izquierda y sale por la derecha. Para girar en dirección contraria, abres esos dos y cierras los otros dos: el de arriba a la derecha y el de abajo a la izquierda. Ahora la corriente entra por la derecha y sale por la izquierda: la polaridad se invirtió y el motor gira al revés. Si abres los cuatro interruptores, el motor queda desconectado y se detiene por inercia. Y hay una combinación prohibida: nunca cierras los dos interruptores del mismo lado al mismo tiempo, porque crearías un cortocircuito directo entre el positivo y la tierra, lo que quemaría los transistores al instante. Los buenos módulos de puente H ya traen protecciones para evitar esto, pero es importante que entiendas por que existe esa regla.

Mira el diagrama conceptual en texto. Llamemos A y B a los interruptores izquierdos (arriba y abajo) y C y D a los derechos (arriba y abajo). El motor esta entre el punto medio izquierdo y el punto medio derecho:



Con A y D cerrados, la corriente va de +V por A, entra al motor por la izquierda, sale por la derecha y baja por D a GND: giro en un sentido. Con C y B cerrados, la corriente entra por la derecha y sale por la izquierda: giro en sentido contrario. Con todos abiertos, motor detenido. Con A y B cerrados juntos: cortocircuito, prohibido. Así de claro es el principio. Todo controlador de motores DC, desde el mas barato hasta los de robots industriales, usa alguna version de este circuito.

El puente H también permite algo muy util: el frenado eléctrico. Si cierras los dos interruptores de abajo (B y D) al mismo tiempo, ambas terminales del motor quedan conectadas a tierra. El motor, que al girar genera electricidad como un pequeño generador, ve sus terminales en cortocircuito y se frena bruscamente, mucho mas rápido que si simplemente lo desconectaras y lo dejarás detenerse por inercia. Esto se llama freno dinamico y es util cuando quieres que tu robot se detenga en un punto preciso.

El módulo L298N: tu puente H listo para usar

Armar un puente H con transistores sueltos es un excelente ejercicio de electrónica, pero para tus proyectos de robótica lo práctico es usar un módulo ya armado. El mas popular del mundo maker es el módulo L298N, una tarjetita roja o azul que trae el chip L298N, que contiene dos puentes H completos, lo que significa que puede controlar dos motores DC independientes, o un motor paso a paso. Es barato, aguanta hasta 2 amperios por canal y trabaja con motores de 5 a 35 voltios. Para los motorreductores de 3 a 6 voltios de tu carrito es mas que suficiente.

Vamos a conocer sus pines uno por uno, porque entender cada pin es la diferencia entre un circuito que funciona a la primera y una tarde de frustración. El módulo tiene tres grupos de conexiones: la alimentación, las salidas a los motores y las entradas de control.

Primero la alimentación. El módulo tiene un bloque de terminales de tornillo con tres posiciones: +12V, GND y +5V. Aquí hay que poner mucha atención. El pin marcado +12V es la entrada de energía para los motores. Ahí conectas tu fuente separada: el portapilas o el eliminador de 6 a 12 voltios. Ojo: aunque diga +12V, acepta desde 5 hasta 35 voltios, así que

tus 6 voltios del portapilas están perfectos. El pin GND es la tierra, y aquí viene un punto crítico que veremos en errores comunes: este GND debe conectarse también al GND del Arduino, para que ambos circuitos compartan la misma referencia. El pin +5V es especial: el módulo trae un regulador interno que puede generar 5 voltios para alimentar lógica. Hay un jumper, un pequeño puente plástico, cerca de ese bloque de terminales. Si el jumper está puesto y alimentas los motores con entre 7 y 12 voltios, el pin +5V se convierte en una SALIDA de 5 voltios que puedes usar para alimentar tu Arduino. Pero si alimentas los motores con menos de 7 voltios, por ejemplo con 6 voltios del portapilas, debes QUITAR ese jumper, porque el regulador no trabaja bien con tan poco voltaje, y en ese caso el pin +5V se convierte en una ENTRADA: debes alimentarlo con 5 voltios desde el Arduino para que la lógica del módulo funcione. Esta es la configuración más común en carritos con 4 pilas AA: jumper fuera, pin +5V del módulo conectado al pin 5V del Arduino, y pin +12V del módulo conectado al positivo del portapilas. Memoriza esto porque es la causa número uno de "conecte todo y no se mueve".

Segundo, las salidas a los motores. Hay dos bloques de terminales: OUT1 y OUT2 para el motor A, OUT3 y OUT4 para el motor B. Conectas las dos terminales de tu motor DC a OUT1 y OUT2, sin importar el orden: si el motor gira al revés de lo que esperas, simplemente intercambias los cables o lo corriges en el código. No hay polaridad correcta o incorrecta aquí, solo dirección.

Tercero, las entradas de control. Son seis pines en una tira: ENA, IN1, IN2, IN3, IN4 y ENB. IN1 e IN2 controlan la dirección del motor A: con IN1 en HIGH e IN2 en LOW el motor gira en un sentido, con IN1 en LOW e IN2 en HIGH gira en el sentido contrario, y con ambos en LOW el motor se detiene. IN3 e IN4 hacen lo mismo para el motor B. ENA y ENB son los pines de habilitación y velocidad: ENA controla al motor A y ENB al motor B. Estos dos pines son los que reciben la señal PWM para controlar la velocidad. El módulo trae unos jumpers puestos en ENA y ENB de fábrica: con el jumper puesto, el motor va siempre a máxima velocidad. Para controlar la velocidad desde el Arduino debes quitar esos jumpers y conectar ENA y ENB a pines PWM del Arduino, que en el Uno son los pines 3, 5, 6, 9, 10 y 11, los que traen el simbolito de virgulilla (~).

Un detalle de seguridad importante del L298N: el chip se calienta cuando maneja corriente, por eso trae un disipador de aluminio encima. Si tus motores consumen mucha corriente de forma continua, toca el disipador con cuidado de vez en cuando: si quema al tacto, necesitas motores más pequeños, mejor ventilación o un controlador más robusto. Nunca es normal que huelas a quemado.

PWM y analogWrite(): velocidad con un interruptor rapidísimo

El Arduino Uno no tiene salidas analógicas de verdad: sus pines digitales solo saben dar 0 voltios o 5 voltios, apagado o encendido. Entonces, como le hacemos para darle al motor "un poco menos de voltaje" y que gire mas despacio? La respuesta es el PWM, siglas en inglés de modulación por ancho de pulso. La idea es brillante por lo simple: en lugar de bajar el voltaje, prendemos y apagamos el motor rapidísimo, cientos de veces por segundo. Si el motor esta encendido la mitad del tiempo y apagado la otra mitad, en promedio recibe la mitad de la energía, y gira aproximadamente a la mitad de la velocidad. El motor no alcanza a detenerse en los instantes de apagado porque la conmutacion es tan rápida que su inercia mecánica lo mantiene girando de forma continua y suave.

Piensa en la analogía del interruptor rapidísimo. Imagina que tienes un foco conectado a un interruptor y lo prendes y apagas con la mano. Si lo dejas prendido 9 segundos y apagado 1 segundo de cada 10, el foco se ve casi a plena luz. Si lo dejas prendido 5 y apagado 5, se ve a media luz. Si lo dejas prendido 1 y apagado 9, apenas brilla. El PWM hace exactamente eso, pero a una velocidad de unas 490 veces por segundo en el Arduino Uno, tan rápido que el motor lo percibe como un voltaje continuo intermedio. A la fracción del tiempo que la señal pasa encendida se le llama ciclo de trabajo, y se expresa en porcentaje: 100 por ciento es siempre encendido, 50 por ciento es mitad y mitad, 0 por ciento es apagado.

En Arduino, el PWM se genera con la función `analogWrite(pin, valor)`. El nombre confunde un poco porque no es realmente analógico, pero así se llama. El pin debe ser uno de los pines PWM, marcados con ~ en la placa: 3, 5, 6, 9, 10 y 11 en el Uno. El valor va de 0 a 255. Con 0 el motor esta apagado, con 255 va a máxima velocidad, y con 127 va aproximadamente a la mitad. La relación es lineal: el valor 64 es un cuarto de velocidad, el 192 es tres cuartos. Para convertir un porcentaje a valor PWM multiplicas el porcentaje por 2.55: por ejemplo, 50 por ciento por 2.55 da 127.5, que redondeamos a 127.

Hay un detalle práctico que debes conocer: los motores reales no empiezan a girar con valores PWM muy bajos. Un motorreductor típico necesita un valor de al menos 60 a 90 para vencer la fricción interna y empezar a moverse; con valores menores solo zumba o vibra sin girar. A esto se le llama zona muerta. Cuando programes controles de velocidad, ten en cuenta que el rango util real de tu motor quizá sea de 80 a 255, no de 0 a 255. Cada motor es distinto, así que vale la pena hacer una pequeña prueba para encontrar el valor mínimo con el que TU motor arranca.

La frecuencia del PWM en el Arduino Uno es de aproximadamente 490 hertz en la mayoría de los pines PWM, y de 980 hertz en los pines 5 y 6. Para controlar motores, 490 hertz funciona bien y es lo estándar. No necesitas cambiar la frecuencia para los proyectos de este curso.

Rampas de aceleracion: por que los robots bien hechos arrancan suave

Cuando enciendes un motor de golpe a máxima velocidad, pasan tres cosas malas. Primera: la corriente de arranque se dispara al máximo, estresando la fuente, el controlador y el motor. Segunda: el tiron mecánico brusco puede hacer que tu carrito patine, se des controle o hasta se voltee si es liviano. Tercera: si el robot lleva sensores delicados o una cámara, el sacudidon mete ruido en las lecturas. Los coches de verdad no arrancan de 0 a 100 en un instante, y tu robot tampoco debería.

La solución son las rampas de aceleracion: en lugar de saltar de 0 a 255 de un golpe, subes el valor PWM poco a poco, por ejemplo de 10 en 10 cada 50 milisegundos, hasta llegar a la velocidad deseada. El motor arranca suave, la corriente sube de forma controlada y el robot se mueve con elegancia. Lo mismo al frenar: bajas el PWM gradualmente en lugar de cortarlo de golpe, y el robot se detiene sin patinar. Programar una rampa es fácil: un ciclo for que incrementa el valor de analogWrite con pequeños retardos entre pasos. En los ejemplos de esta unidad verás exactamente como hacerlo.

Las rampas tienen además un beneficio eléctrico: reducen los picos de corriente que pueden reiniciar tu Arduino. Un problema clásico en carritos es que al arrancar los motores de golpe, la caída de voltaje en las baterías es tan brusca que el Arduino se reinicia solo. Con rampas suaves, ese problema casi desaparece. Es una de esas buenas prácticas que separan un prototipo frágil de un robot confiable.

Errores comunes y buenas prácticas

Vamos a repasar los errores que cometen casi todos los principiantes con el L298N, para que tu los evites desde el día uno. El error número uno es olvidar el GND común. El Arduino y el módulo L298N son dos circuitos con fuentes distintas, y para que las señales de control tengan sentido, ambas tierras deben estar conectadas entre si. Si no conectas el GND del Arduino al GND del módulo, las señales de IN1, IN2 y ENA llegan flotando, sin referencia, y el motor hace cosas raras: no se mueve, tiembla o gira cuando no debe. La regla de oro: siempre une las tierras. Un cable del GND del Arduino al GND del módulo y listo.

El error número dos es alimentar el motor desde el Arduino. Ya lo dijimos en la introducción pero lo repetimos porque es el mas peligroso: el pin de 5V del Arduino puede dar como máximo unos 500 miliamperios si lo alimentas por USB, y mucho menos de forma segura y sostenida. Un motor lo supera fácil al arrancar. El motor SIEMPRE se alimenta de una fuente separada conectada al pin +12V del módulo. El Arduino solo manda señales de control por IN1, IN2 y ENA, que consumen una corriente minúscula. Graba esto: potencia por un lado, control por otro.

El error número tres es el jumper de 5V mal puesto. Recuerda: si alimentas los motores con 7 a 12 voltios, deja el jumper puesto y el módulo te da 5V para el Arduino. Si alimentas con menos de 7 voltios, como las 4 pilas AA que dan 6 voltios, QUITA el jumper y alimenta el pin +5V del módulo desde el pin 5V del Arduino. Con el jumper mal puesto, el módulo no enciende su lógica y nada funciona, y te pasas una hora revisando el código cuando el problema era un pedacito de plástico.

El error número cuatro es no quitar los jumpers de ENA y ENB cuando quieres controlar velocidad. Si dejas esos jumpers puestos, el pin de habilitación queda fijo en HIGH y el motor siempre va a máxima velocidad, ignores lo que ignores con `analogWrite`. Para control de velocidad por PWM, esos jumpers deben estar fuera y ENA y ENB conectados a pines PWM del Arduino.

El error número cinco es conectar ENA o ENB a un pin que no es PWM. Si conectas ENA al pin 4, por ejemplo, `analogWrite` no genera PWM ahí y el motor solo entiende encendido o apagado. Usa siempre los pines marcados con ~: 3, 5, 6, 9, 10 y 11.

Como buenas prácticas generales: primero, verifica dos veces las conexiones de alimentación antes de energizar, con el circuito apagado. Un cable de batería invertido puede dañar el módulo. Segundo, empieza tus pruebas con valores PWM bajos e incrementa poco a poco. Tercero, sujeta bien el motor a la mesa o al chasis antes de hacerlo girar: un motor suelto a máxima velocidad salta y puede tirar cables. Cuarto, si algo huele a quemado o se calienta demasiado, apaga todo de inmediato y revisa antes de volver a encender. Quinto, documenta tus conexiones con un dibujito en tu cuaderno: que pin del Arduino va a que pin del módulo. Te salvara horas cuando vuelvas al proyecto días después. Y sexto, nunca metas los dedos ni el cabello cerca de engranes o ejes girando: los motorreductores tienen fuerza suficiente para lastimar.

Ejemplos paso a paso

En esta sección vas a construir cinco circuitos reales, del mas simple al mas completo. En cada ejemplo encontrarás la lista de materiales, las conexiones paso a paso descritas en texto, el código comentado y una explicación de como funciona. Todos los programas compilan en Arduino Uno. Lee con atención las notas de seguridad de cada ejemplo: son las que cuidan tu placa y tus dedos.

Antes de empezar con el ejemplo A, prepara tu área de trabajo. Desconecta todo de la corriente mientras haces las conexiones. Ten a la mano tu Arduino Uno, el módulo L298N, un motor DC de 3 a 6 voltios, un portapilas con 4 pilas AA (6 voltios en total) o un eliminador de 6

a 9 voltios, cables jumper macho-macho y macho-hembra, y tu protoboard. Recuerda la regla de oro de esta unidad: el motor se alimenta de la fuente separada, nunca del Arduino.

Ejemplo A: girar el motor en una dirección con el L298N

Este es el "hola mundo" de los motores. El objetivo es simple: que el motor gire en una dirección a velocidad controlada. Con este ejemplo verificas que tu módulo funciona, que la alimentación está bien y que entiendes los pines IN1, IN2 y ENA.

Materiales: Arduino Uno, módulo L298N, un motor DC de 3 a 6 voltios, portapilas con 4 pilas AA, cables jumper.

Conexiones paso a paso, con todo apagado. Primero la alimentación del módulo. Conecta el cable rojo del portapilas al terminal de tornillo marcado +12V del L298N, y el cable negro del portapilas al terminal GND del mismo bloque. Como alimentas con 6 voltios, que es menos de 7, QUITA el jumper pequeño que está junto al bloque de terminales, el jumper del regulador de 5V. Ahora el pin +5V del módulo es una entrada: conéctalo con un jumper al pin 5V del Arduino. Después une las tierras: conecta el terminal GND del módulo a un pin GND del Arduino. Este cable del GND común es obligatorio, sin el nada funciona bien.

Ahora las salidas al motor. Conecta las dos terminales de tu motor DC a OUT1 y OUT2 del módulo, apretando bien los tornillos. No importa el orden por ahora.

Ahora el control. Quita el jumper de ENA, el que viene puesto de fábrica. Conecta el pin ENA del módulo al pin 9 del Arduino, que es un pin PWM marcado con ~. Conecta IN1 del módulo al pin 8 del Arduino, e IN2 del módulo al pin 7 del Arduino. Listo. Verifica visualmente: portapilas al +12V y GND del módulo, jumper del regulador fuera, 5V del módulo al 5V del Arduino, GND del módulo al GND del Arduino, motor a OUT1 y OUT2, ENA al pin 9, IN1 al pin 8, IN2 al pin 7.

Sube el siguiente código y observa. El motor debe girar en una dirección a media velocidad aproximadamente.


```

// Ejemplo A: motor DC en una direccion con L298N
// Seguridad: el motor se alimenta del portapilas (fuente separada),
// nunca de un pin del Arduino. Une siempre el GND del modulo
// con el GND del Arduino.

const int PIN_ENA = 9;    // PWM: controla la velocidad del motor A
const int PIN_IN1 = 8;    // direccion del motor A
const int PIN_IN2 = 7;    // direccion del motor A

void setup() {
  pinMode(PIN_ENA, OUTPUT);
  pinMode(PIN_IN1, OUTPUT);
  pinMode(PIN_IN2, OUTPUT);
}

void loop() {
  // IN1 en HIGH e IN2 en LOW: el motor gira en un sentido.
  digitalWrite(PIN_IN1, HIGH);
  digitalWrite(PIN_IN2, LOW);
  // Velocidad media: 150 de 255 (aprox 59 por ciento).
  analogWrite(PIN_ENA, 150);
  // El motor sigue girando; no hacemos nada mas en el loop.
}

```

Como funciona: en el setup declaramos los tres pines como salidas. En el loop ponemos IN1 en HIGH e IN2 en LOW, lo que cierra el puente H en una de sus dos configuraciones validas y hace que la corriente circule por el motor en un sentido. Con analogWrite mandamos un PWM de valor 150 al pin ENA, así el motor gira a velocidad media. Si el motor gira al revés de lo que esperabas, no te preocupes: intercambia los valores de IN1 e IN2 en el código, o intercambia físicamente los cables del motor en OUT1 y OUT2.

Prueba por tu cuenta: cambia el 150 por 255 y verás la máxima velocidad; cambialo por 80 y verás como apenas gira, cerca de su zona muerta. Si con 80 no se mueve, tu motor necesita mas valor mínimo: prueba con 100. Este experimento te enseña el rango util real de tu motor.

Nota de seguridad: si el motor no gira y el módulo se calienta rápido, apaga todo de inmediato. Revisa que no hayas invertido la alimentación y que el jumper del regulador este fuera con 6 voltios. Nunca dejes un circuito trabado energizado mientras lo revisas.

Ejemplo B: control de dirección adelante y atrás con dos botones

Ahora vamos a controlar la dirección con dos botones: uno para adelante y otro para atrás. Cuando no presionas ninguno, el motor se detiene. Este ejemplo introduce la lectura de botones con resistencias pull-up internas, una técnica que ya debes dominar de unidades anteriores, aplicada al control de un motor.

Materiales: todo lo del ejemplo A, mas dos botones pulsadores y cables jumper.

Conexiones: mantén todo lo del ejemplo A exactamente igual. Agrega los botones. Conecta una patita del botón 1 al pin 2 del Arduino y la otra patita a GND del Arduino. Haz lo mismo con el botón 2: una patita al pin 3 del Arduino y la otra a GND. Vamos a usar la resistencia pull-up interna del Arduino con `pinMode(pin, INPUT_PULLUP)`, así no necesitas resistencias externas: el pin lee HIGH cuando el botón esta suelto y LOW cuando lo presionas.

El código:

```

// Ejemplo B: direccion adelante/atras con dos botones
// Boton 1 (pin 2): adelante. Boton 2 (pin 3): atras.
// Sin boton presionado: el motor se detiene.
// Seguridad: fuente separada para el motor, GND comun obligatorio.

const int PIN_ENA = 9;
const int PIN_IN1 = 8;
const int PIN_IN2 = 7;
const int BTN_ADELANTE = 2;
const int BTN_ATRAS = 3;
const int VELOCIDAD = 180; // velocidad fija para este ejemplo

void setup() {
  pinMode(PIN_ENA, OUTPUT);
  pinMode(PIN_IN1, OUTPUT);
  pinMode(PIN_IN2, OUTPUT);
  pinMode(BTN_ADELANTE, INPUT_PULLUP);
  pinMode(BTN_ATRAS, INPUT_PULLUP);
}

void loop() {
  bool adelante = (digitalRead(BTN_ADELANTE) == LOW); // presionado
  bool atras = (digitalRead(BTN_ATRAS) == LOW);

  if (adelante && !atras) {
    // Gira en un sentido.
    digitalWrite(PIN_IN1, HIGH);
    digitalWrite(PIN_IN2, LOW);
    analogWrite(PIN_ENA, VELOCIDAD);
  }
  else if (atras && !adelante) {
    // Gira en sentido contrario: se invierte la polaridad.
    digitalWrite(PIN_IN1, LOW);
    digitalWrite(PIN_IN2, HIGH);
    analogWrite(PIN_ENA, VELOCIDAD);
  }
  else {
    // Ningun boton, o ambos a la vez: detener por seguridad.
    digitalWrite(PIN_IN1, LOW);
    digitalWrite(PIN_IN2, LOW);
    analogWrite(PIN_ENA, 0);
  }
  delay(20); // pequeno antirrebote
}

```

Como funciona: leemos los dos botones en cada vuelta del loop. Como usamos INPUT_PULLUP, el botón presionado se lee como LOW. Si solo esta presionado el de adelante, configuramos el puente H en un sentido y aplicamos velocidad. Si solo esta presionado el de atrás, invertimos IN1 e IN2 y el motor gira al revés: ahí esta el puente H invirtiendo la polaridad en acción. Si no hay botón presionado, o si por accidente se presionan los dos al mismo tiempo, detenemos el motor poniendo IN1 e IN2 en LOW y el PWM en 0. Esa condición de seguridad evita comportamientos extraños.

Observa algo importante: el cambio de dirección es instantáneo en este ejemplo. Si el motor va a toda velocidad hacia adelante y de golpe lo pones en reversa, el tiron mecánico es fuerte y el pico de corriente es grande. En un carrito real esto puede patinar las llantas. Mas adelante, con las rampas de aceleracion, aprenderás a hacer estas transiciones suaves. Por ahora, para tus pruebas, suelta el botón, espera a que el motor se detenga y luego presiona el otro.

Nota de seguridad: los botones manejan solo señales de control de 5 voltios del Arduino, nunca la corriente del motor. Jamás conectes un botón en serie con la alimentación del motor para prenderlo y apagarlo a mano: los botones pequeños no están hechos para esa corriente y se pueden dañar o calentar.

Ejemplo C: control de velocidad con potenciómetro

Este es uno de los ejemplos mas satisfactorios: giras una perilla y el motor responde al instante, despacio o rápido según tu mano. Vamos a leer un potenciómetro con analogRead, convertir su valor con la función map y aplicarlo al motor con analogWrite. Es la cadena completa de control analógico: sensor, procesamiento y actuador.

Materiales: todo lo del ejemplo A, mas un potenciómetro de 10 kiloohmios.

Conexiones: mantén el circuito del motor del ejemplo A. Agrega el potenciómetro: conecta una patita extrema del potenciómetro al pin 5V del Arduino, la otra patita extrema a GND del Arduino, y la patita del centro, el cursor, al pin analógico A0 del Arduino. Así, al girar la perilla, el pin A0 lee un voltaje entre 0 y 5 voltios, que el Arduino convierte a un número entre 0 y 1023.

El código:

```

// Ejemplo C: velocidad del motor con potenciómetro
// El potenciómetro en A0 da valores de 0 a 1023.
// Los convertimos a PWM de 0 a 255 con map().
// Seguridad: fuente separada para el motor, GND comun obligatorio.

const int PIN_ENA = 9;
const int PIN_IN1 = 8;
const int PIN_IN2 = 7;
const int PIN_POT = A0;

void setup() {
  pinMode(PIN_ENA, OUTPUT);
  pinMode(PIN_IN1, OUTPUT);
  pinMode(PIN_IN2, OUTPUT);
  // Fijamos una direccion de giro para este ejemplo.
  digitalWrite(PIN_IN1, HIGH);
  digitalWrite(PIN_IN2, LOW);
}

void loop() {
  int lectura = analogRead(PIN_POT);      // 0 a 1023
  int velocidad = map(lectura, 0, 1023, 0, 255); // 0 a 255
  analogWrite(PIN_ENA, velocidad);
  delay(20); // pequena pausa para estabilizar la lectura
}

```

Como funciona: `analogRead` lee la posición del potenciómetro como un número entre 0 y 1023. La función `map` convierte ese rango al rango de PWM de 0 a 255 de forma proporcional. `analogWrite` aplica el resultado al pin ENA y el motor gira a la velocidad que marca la perilla. La dirección queda fija en el setup porque este ejemplo se enfoca en la velocidad.

Prueba esto: gira la perilla muy despacio desde cero y encuentra el punto exacto donde el motor empieza a moverse. Ese valor de PWM es el mínimo útil de tu motor, su salida de la zona muerta. Anótalo: lo vas a necesitar cuando programes rampas y controles finos. También notarás que entre ese mínimo y el máximo la velocidad sube de forma continua y suave. Eso es el PWM trabajando.

Nota de seguridad: el potenciómetro solo maneja la señal de control del Arduino, una corriente minúscula. Nunca intentes controlar la velocidad del motor poniendo un potenciómetro en serie con su alimentación: el potenciómetro se calentaría y se quemaría porque no está diseñado para la corriente de un motor. La velocidad siempre se controla con PWM a través del controlador.

Ejemplo D: rampa de aceleracion suave

Aquí aplicamos una de las buenas prácticas mas importantes de la unidad: arrancar y frenar suavemente. El programa acelera el motor desde cero hasta la máxima velocidad en pasos pequeños, lo mantiene girando un momento, y luego lo desacelera suavemente hasta detenerlo. El ciclo se repite. Verás y escucharás la diferencia: el motor sube de revoluciones como un coche de verdad.

Materiales: los mismos del ejemplo A, sin componentes extra.

Conexiones: identicas al ejemplo A.

El código:

```

// Ejemplo D: rampa de aceleracion y desaceleracion suave
// El motor sube de 0 a 255 en pasos, se mantiene, y baja suave.
// Asi se evitan los picos de corriente del arranque brusco.
// Seguridad: fuente separada para el motor, GND comun obligatorio.

const int PIN_ENA = 9;
const int PIN_IN1 = 8;
const int PIN_IN2 = 7;
const int PASO = 5;           // cuanto sube el PWM en cada paso
const int PAUSA_MS = 30;     // pausa entre pasos (milisegundos)

void setup() {
  pinMode(PIN_ENA, OUTPUT);
  pinMode(PIN_IN1, OUTPUT);
  pinMode(PIN_IN2, OUTPUT);
  digitalWrite(PIN_IN1, HIGH); // direccion fija
  digitalWrite(PIN_IN2, LOW);
}

void loop() {
  // Rampa de subida: de 0 a 255.
  for (int v = 0; v <= 255; v += PASO) {
    analogWrite(PIN_ENA, v);
    delay(PAUSA_MS);
  }
  delay(2000); // girando a maxima velocidad por 2 segundos

  // Rampa de bajada: de 255 a 0.
  for (int v = 255; v >= 0; v -= PASO) {
    analogWrite(PIN_ENA, v);
    delay(PAUSA_MS);
  }
  delay(2000); // detenido por 2 segundos
}

```

Como funciona: el primer ciclo for sube el valor PWM de 0 a 255 en pasos de 5, con una pausa de 30 milisegundos entre cada paso. La rampa completa de subida tarda 51 pasos por 30 milisegundos, es decir, alrededor de 1.5 segundos. El motor acelera de forma progresiva y la corriente sube de manera controlada, sin el pico brutal del arranque directo. Después de 2 segundos a máxima velocidad, el segundo for baja el PWM de 255 a 0 con la misma suavidad, y el motor se detiene sin sacudidas.

Puedes ajustar la suavidad cambiando PASO y PAUSA_MS. Un PASO mas pequeño o una PAUSA_MS mas grande dan una rampa mas lenta y suave. Para un carrito pesado conviene una rampa lenta; para un ventilador da igual. Haz la prueba con PASO de 1 y PAUSA_MS de 15 y verás una aceleracion larguísima y elegante.

Un truco profesional: en lugar de empezar la rampa desde 0, puedes empezarla desde el valor mínimo util de tu motor, ese que encontraste en el ejemplo C. Así evitas la zona muerta donde el motor solo zumba sin girar, y la rampa se siente aun mas lineal. Por ejemplo, si tu motor arranca en 80, tu for puede ir de 80 a 255.

Ejemplo E: control de DOS motores, la base de un carrito

Este es el ejemplo que une todo: vamos a controlar dos motores independientes con el L298N, usando sus dos puentes H. Esto es exactamente lo que necesita un carrito robótico: un motor para la rueda izquierda y otro para la rueda derecha. Con este código puedes hacer que el carrito avance, retroceda, gire a la izquierda, gire a la derecha y se detenga.

Materiales: Arduino Uno, módulo L298N, DOS motores DC de 3 a 6 voltios, portapilas con 4 pilas AA o eliminador de 6 a 9 voltios, cables jumper.

Conexiones paso a paso. La alimentación es igual que en el ejemplo A: portapilas al +12V y GND del módulo, jumper del regulador fuera con 6 voltios, pin +5V del módulo al 5V del Arduino, y GND del módulo al GND del Arduino. No olvides el GND común.

Motor izquierdo: conéctalo a OUT1 y OUT2. Quita el jumper de ENA. Conecta ENA al pin 9 del Arduino, IN1 al pin 8, IN2 al pin 7.

Motor derecho: conéctalo a OUT3 y OUT4. Quita el jumper de ENB. Conecta ENB al pin 10 del Arduino, que también es PWM, IN3 al pin 6 e IN4 al pin 5. Ojo: los pines 5 y 6 son PWM, pero aquí los usamos como digitales normales para dirección, y eso esta bien.

Verifica: cada motor tiene su par de pines de dirección y su pin PWM de velocidad. El GND común une al Arduino con el módulo. Todo con la energía apagada mientras conectas.

El código crea funciones para cada movimiento, que es como se programan los carritos de verdad:

```
// Ejemplo E: control de dos motores (base de un carrito robot)
// Motor A (izquierdo): ENA=9, IN1=8, IN2=7
// Motor B (derecho): ENB=10, IN3=6, IN4=5
// Seguridad: fuente separada para los motores, GND comun obligatorio.
// Sujeta los motores o el chasis antes de probar: a maxima
// velocidad pueden saltar de la mesa.
```



```

const int ENA = 9, IN1 = 8, IN2 = 7;    // motor izquierdo
const int ENB = 10, IN3 = 6, IN4 = 5;   // motor derecho
const int VEL = 200;                    // velocidad de cruceo

void setup() {
    pinMode(ENA, OUTPUT); pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
    pinMode(ENB, OUTPUT); pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
    detener();
}

// --- Funciones de movimiento ---
void adelante() {
    digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
    digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
    analogWrite(ENA, VEL); analogWrite(ENB, VEL);
}

void atras() {
    digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
    digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
    analogWrite(ENA, VEL); analogWrite(ENB, VEL);
}

void girarIzquierda() {
    // El motor derecho avanza y el izquierdo retrocede: giro en su lugar.
    digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
    digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
    analogWrite(ENA, VEL); analogWrite(ENB, VEL);
}

void girarDerecha() {
    digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
    analogWrite(ENA, VEL); analogWrite(ENB, VEL);
}

void detener() {
    digitalWrite(IN1, LOW); digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW); digitalWrite(IN4, LOW);
    analogWrite(ENA, 0); analogWrite(ENB, 0);
}

void loop() {
    adelante(); delay(2000);
    detener(); delay(500);
    girarIzquierda(); delay(1000);
    detener(); delay(500);
    girarDerecha(); delay(1000);
}

```

```
detener(); delay(500);  
atras(); delay(2000);  
detener(); delay(2000);  
}
```

Como funciona: cada función configura los dos puentes H para un movimiento. Para avanzar, ambos motores giran en el mismo sentido. Para retroceder, ambos giran al revés. Para girar a la izquierda en su lugar, el motor derecho avanza mientras el izquierdo retrocede, y el carrito rota sobre su propio eje; para la derecha es al revés. La función detener pone todo en LOW y el PWM en 0. El loop ejecuta una secuencia de demostración: avanza 2 segundos, se detiene, gira a la izquierda, se detiene, gira a la derecha, se detiene, retrocede y se detiene.

Un detalle que notarás al probar: es posible que un motor gire al revés de lo esperado y el carrito gire cuando debería avanzar. Esto pasa porque los dos motores quedaron montados en orientaciones opuestas o los cables quedaron cruzados. La solución es fácil: invierte los valores HIGH y LOW de IN3 e IN4 en el código, o intercambia los cables de ese motor en OUT3 y OUT4. Haz la prueba y ajusta hasta que adelante sea realmente adelante.

Nota de seguridad final de la sección: con dos motores la corriente total se duplica. Si cada motor pide 800 miliamperios al arrancar, tu fuente debe poder dar 1.6 amperios en el pico. Un portapilas con pilas AA comunes puede quedarse corto y el Arduino se reinicia; si te pasa eso, usa pilas recargables de buena capacidad o un eliminador con suficiente amperaje. Y siempre prueba primero con los motores al aire o el carrito levantado, para verificar las direcciones antes de soltarlo al piso.

Práctica

La teoría sin práctica se olvida en una semana. Estos cinco ejercicios están diseñados para que afirmes todo lo aprendido en la unidad: dirección, velocidad, rampas y control por comandos. Hazlos en orden, porque cada uno usa algo del anterior. Cada ejercicio trae su enunciado, su objetivo y su solución completa con código y explicación. Intenta resolverlo tu primero antes de ver la solución: la struggle, el esfuerzo de pensar, es donde ocurre el aprendizaje de verdad.

Ejercicio 1: secuencia adelante, pausa y atrás. Enunciado: programa un motor conectado al L298N para que gire hacia adelante durante 3 segundos a velocidad media, se detenga por completo durante 1 segundo, gire hacia atrás durante 3 segundos a la misma velocidad, se detenga 1 segundo, y repita el ciclo para siempre. Objetivo: dominar el control de dirección con IN1 e IN2 y las pausas con detener. Usa las mismas conexiones del ejemplo A.

Solución del ejercicio 1

```
// Solucion ejercicio 1: secuencia adelante-pausa-atras
// Conexiones iguales al ejemplo A: ENA=9, IN1=8, IN2=7.
// Seguridad: fuente separada para el motor, GND comun obligatorio.

const int PIN_ENA = 9;
const int PIN_IN1 = 8;
const int PIN_IN2 = 7;
const int VELOCIDAD = 150;

void setup() {
  pinMode(PIN_ENA, OUTPUT);
  pinMode(PIN_IN1, OUTPUT);
  pinMode(PIN_IN2, OUTPUT);
}

void adelante() {
  digitalWrite(PIN_IN1, HIGH);
  digitalWrite(PIN_IN2, LOW);
  analogWrite(PIN_ENA, VELOCIDAD);
}

void atras() {
  digitalWrite(PIN_IN1, LOW);
  digitalWrite(PIN_IN2, HIGH);
  analogWrite(PIN_ENA, VELOCIDAD);
}

void detener() {
  digitalWrite(PIN_IN1, LOW);
  digitalWrite(PIN_IN2, LOW);
  analogWrite(PIN_ENA, 0);
}

void loop() {
  adelante(); delay(3000);
  detener(); delay(1000);
  atras(); delay(3000);
  detener(); delay(1000);
}
```

Explicación: creamos tres funciones claras, adelante, atrás y detener, igual que en el ejemplo E pero para un solo motor. El loop las llama en secuencia con los tiempos pedidos. Fíjate que detener pone IN1 e IN2 en LOW y el PWM en 0, así el motor queda libre y se detiene por inercia. Si quisieras un frenado mas brusco, podrías poner IN1 e IN2 en HIGH al mismo

tiempo, que cortocircuita el motor y lo frena eléctricamente, pero para este ejercicio el detenido por inercia esta bien.

Ejercicio 2: velocidad con botones de mas y menos. Enunciado: con dos botones, uno de SUBIR y otro de BAJAR, controla la velocidad del motor en pasos de 25. El motor gira siempre en la misma dirección. La velocidad debe quedarse entre 0 y 255 sin pasarse. Cada vez que presionas SUBIR, la velocidad aumenta 25; cada vez que presionas BAJAR, disminuye 25. Muestra la velocidad actual por el monitor serie. Objetivo: practicar variables de estado, limites con constrain y lectura de botones con detección de flanco para que cada presión cuente una sola vez.

Solución del ejercicio 2

```
// Solucion ejercicio 2: velocidad con botones + y -
// Boton SUBIR en pin 2, boton BAJAR en pin 3 (con INPUT_PULLUP).
// Motor igual al ejemplo A: ENA=9, IN1=8, IN2=7.
// Seguridad: fuente separada para el motor, GND comun obligatorio.

const int PIN_ENA = 9;
const int PIN_IN1 = 8;
const int PIN_IN2 = 7;
const int BTN_MAS = 2;
const int BTN_MENOS = 3;
const int PASO = 25;

int velocidad = 0;
int estadoMasAntes = HIGH;
int estadoMenosAntes = HIGH;

void setup() {
  pinMode(PIN_ENA, OUTPUT);
  pinMode(PIN_IN1, OUTPUT);
  pinMode(PIN_IN2, OUTPUT);
  pinMode(BTN_MAS, INPUT_PULLUP);
  pinMode(BTN_MENOS, INPUT_PULLUP);
  digitalWrite(PIN_IN1, HIGH); // direccion fija
  digitalWrite(PIN_IN2, LOW);
  Serial.begin(9600);
  Serial.println("Control de velocidad con botones. + / -");
}

void loop() {
  int estadoMas = digitalRead(BTN_MAS);
  int estadoMenos = digitalRead(BTN_MENOS);
```

```

// Detectamos el flanco de bajada: el momento en que se presiona.
if (estadoMasAntes == HIGH && estadoMas == LOW) {
    velocidad += PASO;
    Serial.print("Velocidad: ");
    Serial.println(velocidad);
    delay(150); // antirrebote sencillo
}
if (estadoMenosAntes == HIGH && estadoMenos == LOW) {
    velocidad -= PASO;
    Serial.print("Velocidad: ");
    Serial.println(velocidad);
    delay(150); // antirrebote sencillo
}
estadoMasAntes = estadoMas;
estadoMenosAntes = estadoMenos;

velocidad = constrain(velocidad, 0, 255); // limites seguros
analogWrite(PIN_ENA, velocidad);
}

```

Explicación: la clave de este ejercicio es la detección de flanco. Si solo preguntáramos si el botón está presionado, mientras lo mantengas oprimido la velocidad subiría sin control en cada vuelta del loop. Al comparar el estado actual con el estado anterior, solo contamos el instante en que el botón pasa de suelto a presionado, así cada presión suma exactamente un paso. La función constrain limita la variable entre 0 y 255 para que nunca se salga del rango válido de analogWrite. El monitor serie te muestra el valor actual, útil para depurar.

Ejercicio 3: arranque suave y frenado suave con un botón. Enunciado: con un solo botón en el pin 2, el motor debe arrancar con rampa suave desde 0 hasta 200, mantenerse ahí mientras el botón siga presionado, y al soltar el botón debe frenar con rampa suave hasta 0. Si se vuelve a presionar, arranca suave otra vez. Objetivo: combinar rampas de aceleración con lectura de botones, y entender que la rampa debe ejecutarse una sola vez por evento, no en cada vuelta del loop.

Solución del ejercicio 3

```

// Solucion ejercicio 3: arranque y frenado suaves con un boton
// Boton en pin 2 (INPUT_PULLUP). Motor igual al ejemplo A.
// Seguridad: fuente separada para el motor, GND comun obligatorio.

const int PIN_ENA = 9;
const int PIN_IN1 = 8;
const int PIN_IN2 = 7;

```

```

const int BTN = 2;
const int VEL_MAX = 200;
const int PASO = 4;
const int PAUSA_MS = 25;

int velocidadActual = 0;
bool estabaPresionado = false;

void setup() {
  pinMode(PIN_ENA, OUTPUT);
  pinMode(PIN_IN1, OUTPUT);
  pinMode(PIN_IN2, OUTPUT);
  pinMode(BTN, INPUT_PULLUP);
  digitalWrite(PIN_IN1, HIGH); // direccion fija
  digitalWrite(PIN_IN2, LOW);
  analogWrite(PIN_ENA, 0);
}

void rampaHasta(int objetivo) {
  // Sube o baja suavemente hasta el valor objetivo.
  while (velocidadActual != objetivo) {
    if (velocidadActual < objetivo) velocidadActual += PASO;
    else velocidadActual -= PASO;
    // Evita pasarse del objetivo en el ultimo paso.
    if (abs(velocidadActual - objetivo) < PASO) velocidadActual = objetivo;
    analogWrite(PIN_ENA, velocidadActual);
    delay(PAUSA_MS);
  }
}

void loop() {
  bool presionado = (digitalRead(BTN) == LOW);

  if (presionado && !estabaPresionado) {
    // Acaban de presionar: rampa de arranque.
    rampaHasta(VEL_MAX);
  }
  if (!presionado && estabaPresionado) {
    // Acaban de soltar: rampa de frenado.
    rampaHasta(0);
  }
  estabaPresionado = presionado;
}

```

Explicación: usamos otra vez la detección de flanco, pero ahora para disparar una rampa completa una sola vez. Cuando se detecta que el botón paso de suelto a presionado, llamamos a rampaHasta con la velocidad máxima, y la función sube el PWM paso a paso hasta llegar. Cuando se detecta que se soltó, llamamos a rampaHasta con 0 y baja suavemente. La variable velocidadActual guarda en que punto va la rampa, y estabaPresionado recuerda el estado anterior del botón. Fíjate que mientras el botón se mantiene presionado no pasa nada nuevo: el motor simplemente sigue a velocidad constante. Este patrón de "hacer algo una sola vez cuando cambia el estado" es fundamental en robótica y lo usarás en sensores, comunicaciones y máquinas de estado.

Ejercicio 4: control por comandos del monitor serie. Enunciado: controla el motor escribiendo comandos en el monitor serie. Los comandos son: "adelante", "atrás", "stop", y "vel N" donde N es un número entre 0 y 255, por ejemplo "vel 180". El programa debe entender los comandos aunque vengan en mayúsculas o minúsculas y debe ignorar espacios extra. Al recibir un comando valido, responde por el monitor serie confirmando la acción. Objetivo: practicar la lectura de texto por el puerto serie, comparación de cadenas y conversión de texto a número, habilidades clave para controlar robots a distancia.

Solución del ejercicio 4

```
// Solucion ejercicio 4: control del motor por comandos serie
// Comandos: adelante, atras, stop, vel N (N de 0 a 255).
// Motor igual al ejemplo A: ENA=9, IN1=8, IN2=7.
// Seguridad: fuente separada para el motor, GND comun obligatorio.

const int PIN_ENA = 9;
const int PIN_IN1 = 8;
const int PIN_IN2 = 7;

int velocidad = 150;

void setup() {
  pinMode(PIN_ENA, OUTPUT);
  pinMode(PIN_IN1, OUTPUT);
  pinMode(PIN_IN2, OUTPUT);
  Serial.begin(9600);
  Serial.println("Comandos: adelante, atras, stop, vel N (0-255)");
  detener();
}

void adelante() {
  digitalWrite(PIN_IN1, HIGH);
```

```

    digitalWrite(PIN_IN2, LOW);
    analogWrite(PIN_ENA, velocidad);
    Serial.print("Avanzando a velocidad ");
    Serial.println(velocidad);
}

void atras() {
    digitalWrite(PIN_IN1, LOW);
    digitalWrite(PIN_IN2, HIGH);
    analogWrite(PIN_ENA, velocidad);
    Serial.print("Retrocediendo a velocidad ");
    Serial.println(velocidad);
}

void detener() {
    digitalWrite(PIN_IN1, LOW);
    digitalWrite(PIN_IN2, LOW);
    analogWrite(PIN_ENA, 0);
    Serial.println("Motor detenido");
}

void loop() {
    if (Serial.available() > 0) {
        String cmd = Serial.readStringUntil('\n');
        cmd.trim();           // quita espacios y saltos de linea extra
        cmd.toLowerCase();    // todo a minusculas para comparar facil

        if (cmd == "adelante") {
            adelante();
        }
        else if (cmd == "atras") {
            atras();
        }
        else if (cmd == "stop") {
            detener();
        }
        else if (cmd.startsWith("vel ")) {
            int v = cmd.substring(4).toInt(); // numero despues de "vel "
            v = constrain(v, 0, 255);
            velocidad = v;
            // Reaplica la velocidad al estado actual del motor.
            analogWrite(PIN_ENA, velocidad);
            Serial.print("Velocidad ajustada a ");
            Serial.println(velocidad);
        }
        else {
            Serial.println("Comando no reconocido");
        }
    }
}

```



```

    }
  }
}

```

Explicación: Serial.readStringUntil lee todo lo que escribiste hasta el salto de línea. Con trim quitamos espacios y con toLowerCase normalizamos a minúsculas, así "ADELANTE", "Adelante" y "adelante " funcionan igual. Los comandos simples se comparan directo. Para "vel N" usamos startsWith para detectar el prefijo y substring para extraer el número, que convertimos con toInt y limitamos con constrain. Cada acción confirma por el monitor serie lo que hizo, lo cual es esencial cuando controlas un robot a distancia: siempre quieres saber que el robot entendió tu orden. Este mismo esquema de comandos lo puedes ampliar después para un carrito con dos motores, agregando comandos como "izquierda" o "derecha".

Ejercicio 5: mini ventilador con velocidad variable y apagado automático. Enunciado: construye un mini ventilador usando el motor DC con una pequeña hélice de plástico o cartón pegada al eje. El ventilador tiene 3 velocidades seleccionables con un botón: al presionar el botón avanza velocidad 1 (suave), velocidad 2 (media), velocidad 3 (fuerte) y luego apagado, en ciclo. Además, si el ventilador lleva mas de 30 segundos encendido sin que se presione el botón, debe apagarse solo y la secuencia reinicia desde apagado. Un LED en el pin 13 indica si el ventilador esta encendido. Objetivo: integrar todo lo de la unidad, máquina de estados con un botón, temporizador con millis sin bloquear, y salida adicional con LED.

Solución del ejercicio 5

```

// Solucion ejercicio 5: mini ventilador de 3 velocidades
// Boton en pin 2 (INPUT_PULLUP). LED indicador en pin 13.
// Motor igual al ejemplo A: ENA=9, IN1=8, IN2=7.
// La helice va pegada al eje del motor. CUIDADO: no acerques
// los dedos a la helice cuando este girando.
// Seguridad: fuente separada para el motor, GND comun obligatorio.

const int PIN_ENA = 9;
const int PIN_IN1 = 8;
const int PIN_IN2 = 7;
const int BTN = 2;
const int LED = 13;

// Velocidades del ventilador: apagado, suave, media, fuerte.
const int VEL[4] = {0, 90, 160, 230};
const unsigned long TIEMPO_AUTO_OFF = 30000; // 30 segundos

int nivel = 0; // 0=apagado, 1, 2, 3
int estadoBtnAntes = HIGH;

```

```

unsigned long ultimoCambio = 0;

void setup() {
  pinMode(PIN_ENA, OUTPUT);
  pinMode(PIN_IN1, OUTPUT);
  pinMode(PIN_IN2, OUTPUT);
  pinMode(BTN, INPUT_PULLUP);
  pinMode(LED, OUTPUT);
  digitalWrite(PIN_IN1, HIGH); // direccion fija del ventilador
  digitalWrite(PIN_IN2, LOW);
  aplicarNivel();
}

void aplicarNivel() {
  analogWrite(PIN_ENA, VELs[nivel]);
  digitalWrite(LED, nivel > 0 ? HIGH : LOW);
  ultimoCambio = millis(); // reinicia el temporizador
}

void loop() {
  int estadoBtn = digitalRead(BTN);
  if (estadoBtnAntes == HIGH && estadoBtn == LOW) {
    nivel = (nivel + 1) % 4; // avanza 0-1-2-3-0 en ciclo
    aplicarNivel();
    delay(200); // antirrebote
  }
  estadoBtnAntes = estadoBtn;

  // Apagado automatico tras 30 s sin presionar el boton.
  if (nivel > 0 && millis() - ultimoCambio >= TIEMPO_AUTO_OFF) {
    nivel = 0;
    aplicarNivel();
  }
}

```

Explicación: este ejercicio junta varias técnicas. El arreglo VELS guarda las cuatro velocidades, incluyendo el 0 de apagado. La variable nivel indica en que velocidad estamos, y la expresión $(\text{nivel} + 1) \% 4$ hace que avance en ciclo: de 3 regresa a 0. La función `aplicarNivel` centraliza todo lo que pasa al cambiar de velocidad: escribe el PWM, prende o apaga el LED y reinicia el temporizador. El apagado automático usa `millis`, que cuenta los milisegundos desde que encendió el Arduino, sin bloquear el programa con `delay`: comparamos cuanto tiempo paso desde el último cambio y si supera 30 segundos apagamos. Este patrón de temporizador no bloqueante es oro puro en robótica, porque te permite hacer varias cosas a la vez. Y la advertencia de la hélice va en serio: aunque sea de cartón, una hélice girando rápido puede dar un buen golpe en un dedo. Prueba primero a velocidad 1.

Resumen de la unidad

En esta unidad le diste músculos a tu robot. Aprendiste que los actuadores convierten la energía eléctrica en movimiento, y que el motor DC es el actuador estrella de los robots móviles por ser barato, simple y fácil de controlar. Entendiste su funcionamiento interno: un imán permanente y una bobina que se convierte en electroimán, con un conmutador que invierte la corriente para mantener el giro continuo. Descubriste las dos consecuencias prácticas de ese principio: invertir la polaridad invierte la dirección de giro, y variar el voltaje varía la velocidad.

Aprendiste a leer las especificaciones de un motor: el voltaje nominal, la velocidad en RPM y, la más crítica para la seguridad, la corriente, incluyendo la corriente de arranque que puede ser varias veces la nominal. Grabaste la regla de oro de la seguridad eléctrica: NUNCA conectar un motor directo a un pin del Arduino, porque el motor pide mucha más corriente de la que el pin puede dar, y eso quema la placa. El Arduino es el cerebro que da órdenes; un controlador externo alimentado por separado es el músculo que ejecuta.

Dominaste el puente H, el circuito en forma de H que invierte la polaridad con cuatro interruptores para controlar la dirección de giro electrónicamente, incluyendo el concepto de frenado eléctrico. Conociste a fondo el módulo L298N: sus dos puentes H para dos motores, los pines ENA, IN1 e IN2 para dirección y velocidad, la alimentación separada del motor por el terminal +12V, el jumper de 5V que se quita cuando alimentas con menos de 7 voltios, y la obligación del GND común entre el módulo y el Arduino.

Entendiste el PWM, la modulación por ancho de pulso, con la analogía del interruptor rapidísimo: encender y apagar cientos de veces por segundo para simular un voltaje intermedio. Aprendiste la función `analogWrite` con sus valores de 0 a 255, la zona muerta de los motores reales y la importancia de las rampas de aceleración para arrancar y frenar

suavemente, cuidando la mecánica, la electrónica y las baterías. Repasaste los errores mas comunes, como olvidar el GND común, alimentar el motor desde el Arduino o dejar mal los jumpers, y las buenas prácticas para evitarlos.

En la práctica construiste cinco circuitos: giro en una dirección, control adelante y atrás con botones, velocidad con potenciómetro usando `analogRead` y `map`, rampas de aceleracion suaves, y el control de dos motores que es la base de un carrito robótico, con funciones de movimiento claras. Resolviste ejercicios de secuencias, botones de velocidad, arranque suave con un botón, comandos por monitor serie y un mini ventilador con apagado automático usando `millis` sin bloquear.

Con esto completaste las clases 16, 17 y 18 del curso. Ya puedes mover cosas con precisión desde tu Arduino de forma segura. En la siguiente unidad usarás todo esto para construir tu primer robot móvil completo: un chasis con dos motores que se mueve, gira y reacciona a su entorno. Los músculos ya están listos; ahora les toca caminar.

UNIDAD 7: SERVOMOTORES Y MOVIMIENTO PRECISO

Introducción

Imagínate que tienes dos motores sobre tu mesa. El primero es un motor DC como los que conoces de unidades anteriores: le das energía y gira, gira, gira, sin parar. Puedes cambiarle la velocidad con PWM, puedes invertir su giro con el puente H, pero nunca puedes decirle "gira exactamente hasta los 90 grados y quédate quieto ahí". El motor DC no entiende de posiciones. Es como un caballo desbocado: rápido, fuerte, pero imposible de estacionar con precisión.

El segundo motor es distinto. Le dices "ve a 90 grados" y va a 90 grados y se queda ahí, quieto, sosteniendo la posición aunque algo intente moverlo. Le dices "ve a 45 grados" y obedece. Es un servo, y es el músculo de precisión de la robótica.

Esta diferencia lo cambia todo. Un brazo robótico no puede funcionar con motores DC ordinarios, porque cada articulación necesita estar en un ángulo exacto: el hombro a 30 grados, el codo a 120 grados, la garra abierta o cerrada. Un carrito robot puede usar motores DC en las ruedas para avanzar, pero para girar el volante o apuntar una cámara necesita un servo. La cabeza de un robot que sigue con la mirada a las personas usa servos. Las puertas automáticas, las máquinas expendedoras, los timelapses de cámara con movimiento programado, los aviones de aeromodelismo con sus alerones... todo eso se mueve con servomotores.

En esta unidad vas a dominar el servomotor de arriba abajo. Vas a abrir uno (en sentido figurado, porque no queremos que lo desarmes y pierdas los engranes microscópicos) para entender que tiene adentro: un motor DC pequeño, un tren de engranes que multiplica su fuerza, un potenciómetro que le dice al circuito en qué posición está, y una tarjeta de control que compara "donde estoy" con "donde me pidieron estar" y corrige el error todo el tiempo. Ese circuito cerrado de control es la magia: el servo no solo recibe órdenes, también verifica que las cumplió.

También vas a entender la señal que lo controla, que es una de las ideas más elegantes de la electrónica: un pulso eléctrico que se repite cada 20 milisegundos, y donde el ANCHO del pulso codifica la posición deseada. Un pulso de 1 milisegundo significa "ve a 0 grados". Un pulso de 2 milisegundos significa "ve a 180 grados". Y un pulso de 1.5 milisegundos significa "quédate en el centro, 90 grados". El Arduino genera esa señal por ti usando la librería

Servo.h, así que no tienes que contar microsegundos a mano... aunque aprenderás a hacerlo cuando necesites precisión extra.

Trabajarás con el servo SG90, el servomotor mas popular del mundo para proyectos educativos. Es pequeño, barato, azul, y tiene tres cables: uno rojo para alimentación, uno marron o negro para tierra, y uno naranja o amarillo para la senal. Con ese servo y un Arduino vas a construir cosas reales: un barrido automático como el de los radares de las películas, un control de posición con potenciómetro donde giras una perilla y el servo te obedece al instante, movimientos suaves programados paso a paso, y al final de la unidad un radar funcional que combina el servo con el sensor ultrasónico HC-SR04 para medir distancias en cada angulo y reportarlas por el monitor serial.

Hay algo importante que debes aprender desde ahora y que te va a ahorrar muchos dolores de cabeza: la alimentación. Un servo se mueve con fuerza, y esa fuerza cuesta corriente eléctrica. El puerto USB de tu computadora, que alimenta al Arduino, no siempre alcanza para mover varios servos a la vez, sobre todo cuando están haciendo esfuerzo. Vas a aprender a alimentar los servos con una fuente externa (como un paquete de baterías o un eliminador) compartiendo la tierra (GND) con el Arduino. Esta técnica del "GND común" es fundamental en robótica y la usarás en todos tus proyectos futuros.

Otra cosa que vas a dominar es la calibración. Los servos baratos no son perfectos: algunos no llegan exactamente a 180 grados, otros vibran un poco cuando sostienen posición, otros responden mejor si les hablas en microsegundos en lugar de grados. Aprenderás a encontrar los limites reales de TU servo específico y a ajustar tu código para trabajar dentro de ellos. Esa es la diferencia entre un proyecto que "mas o menos funciona" y uno que funciona bien.

Para cuando termines esta unidad, habrás cubierto las clases 19, 20 y 21 del curso. En la clase 19 entenderás que es un servomotor por dentro y como funciona la senal de pulsos. En la clase 20 dominarás la librería Servo.h: como conectar el servo, como moverlo a posiciones, como hacer barridos, como controlarlo con un potenciómetro y como lograr movimientos suaves. En la clase 21 te meteras con la precisión y la potencia: calibración, limites mecanicos, alimentación externa con GND común, y el proyecto estrella: un radar con sensor ultrasónico que barre el ambiente y te dice donde hay objetos.

Así que conecta tu Arduino, saca tu servo SG90 de su bolsita, y prepárate. Al final de esta unidad vas a tener un radar funcionando sobre tu mesa, y vas a entender exactamente por que cada parte hace lo que hace. Eso es robótica de verdad: no copiar código a ciegas, sino saber por que cada línea existe.

Explicación detallada

Anatomía del servo: que hay adentro de la cajita azul

Si pudieras abrir un servo SG90 sin romperlo (no lo hagas, los engranes son diminutos y los resortes internos son traicioneros), encontrarías cuatro componentes principales trabajando en equipo. Entender este equipo es entender por que el servo puede hacer lo que un motor DC no puede: detenerse en una posición exacta.

El primer componente es un motor DC pequeño. Si, adentro del servo hay un motor DC como los que ya conoces: gira cuando le das energía, y mientras mas energía, mas rápido gira. Por si solo, este motor no sabe detenerse en ningún ángulo. Es la materia prima del movimiento, nada mas.

El segundo componente es el tren de engranes. El motor DC pequeño gira muy rápido pero con muy poca fuerza. Los engranes son una serie de rueditas dentadas de plástico que reducen la velocidad y multiplican la fuerza (el torque). Es el mismo principio de las bicicletas: cuando pones un cambio bajo, pedaleas rápido pero la rueda gira despacio y con mucha fuerza para subir una cuesta. En el SG90, la reducción es tan grande que el eje de salida gira despacio pero con un torque de alrededor de 1.8 kilogramos-centimetro a 4.8 voltios. Eso significa que puede sostener un peso de 1.8 kilogramos colgando a 1 centimetro del eje. Para su tamaño, es muchísima fuerza. El último engrane del tren esta conectado al eje de salida, que es el que ves afuera con sus estrias donde se montan los "cuernos" o brazos de plástico.

El tercer componente es el potenciómetro interno. Un potenciómetro es una resistencia variable: su valor cambia según la posición de su eje. Adentro del servo, el eje del potenciómetro esta acoplado al eje de salida, así que cuando el eje del servo gira, el potenciómetro también gira y su resistencia cambia. El circuito de control lee ese valor y así sabe EN QUE POSICIÓN ESTA el servo en cada momento. Es como si el servo tuviera ojos propios: siempre sabe donde esta su brazo. Sin este componente, el servo seria un motor DC mas, ciego y sin control.

El cuarto componente es el circuito de control, una pequeña tarjeta electrónica con un chip. Este chip hace un trabajo continuo y rapidísimo: compara la posición deseada (la que le llega por el cable de señal desde el Arduino) con la posición actual (la que le reporta el potenciómetro interno). Si hay diferencia, manda energía al motor DC en la dirección correcta para corregirla. Si la posición actual ya es igual a la deseada, deja de mandar energía (o manda solo la necesaria para sostener). Este ciclo de "medir, comparar, corregir" se repite cientos de veces por segundo. En ingeniería se llama "control de lazo cerrado", y es la razón por la que el servo se queda quieto en la posición que le pides e incluso resiste cuando

intentas moverlo con la mano: el circuito detecta que algo lo movió y ordena al motor que regrese.

Fíjate en la belleza del diseño: el motor DC aporta el movimiento, los engranes aportan la fuerza, el potenciómetro aporta la información, y el circuito aporta la inteligencia. Ninguno sirve solo. Juntos forman un sistema que obedece órdenes de posición con precisión de un grado.

La señal de control: el toque en el hombro periodico

Ahora viene la parte mas interesante: como le dices al servo a donde ir. La comunicación entre el Arduino y el servo se hace por un solo cable, el cable de señal, usando pulsos electricos. El protocolo es sencillo pero hay que entenderlo bien porque explica muchos comportamientos extraño que verás en la práctica.

La regla es esta: cada 20 milisegundos, el Arduino manda un pulso (un momento breve en que el cable pasa de 0 voltios a 5 voltios y regresa a 0). Lo que importa no es que el pulso exista, sino CUANTO DURA. El ancho del pulso codifica la posición deseada:

- Pulso de 1.0 milisegundo (1000 microsegundos) = posición 0 grados (extremo izquierdo).
- Pulso de 1.5 milisegundos (1500 microsegundos) = posición 90 grados (centro).
- Pulso de 2.0 milisegundos (2000 microsegundos) = posición 180 grados (extremo derecho).

Entre esos valores, la relación es lineal: un pulso de 1.25 milisegundos apunta a 45 grados, uno de 1.75 milisegundos apunta a 135 grados, y así. El servo lee el ancho de cada pulso, lo compara con la lectura de su potenciómetro interno, y mueve el motor para corregir cualquier diferencia.

Piénsalo con una analogía. Imagínate que estas dormido en un sillón y cada 20 segundos alguien te da un toquecito en el hombro y te susurra una instrucción: "sientate derecho", "recargate a la izquierda", "recargate a la derecha". Tu no necesitas que te estén empujando todo el tiempo; con ese recordatorio periodico basta para que mantengas la postura pedida. El servo funciona igual: no necesita una señal continua, le basta ese "toque en el hombro" eléctrico cada 20 milisegundos para saber donde debe estar. Si los pulsos dejan de llegar, muchos servos se quedan en la última posición (otros se apagan y quedan flojos, depende del modelo).

Por que 20 milisegundos? Es un estándar historico que viene de los radios de aeromodelismo de hace décadas, y se quedó porque funciona bien: es lo bastante rápido para que el servo responda con agilidad, y lo bastante lento para que el circuito tenga tiempo de medir y corregir entre pulso y pulso. La frecuencia correspondiente es 50 Hz (50 pulsos por segundo).

Un detalle práctico importante: el Arduino puede generar estos pulsos por software usando la librería Servo.h, sin que tu tengas que contar microsegundos. La librería usa temporizadores internos del microcontrolador para producir pulsos precisos en segundo plano, mientras tu programa hace otras cosas. En el Arduino Uno puedes controlar hasta 12 servos simultáneamente con esta librería (en el Mega hasta 48). Hay un costo: la librería desactiva el PWM en los pines 9 y 10 del Uno, así que si usas servos, evita analogWrite en esos dos pines.

El servo SG90: especificaciones que si importan

El SG90 es el servo de referencia en educación por una razón: cuesta poco, se consigue en todas partes y es suficiente para la mayoría de proyectos de aprendizaje. Pero "barato" no significa "sin limites". Conoce sus especificaciones para no pedirle imposibles:

- Voltaje de operación: 4.8 a 6 voltios. Funciona con 5V del Arduino para pruebas con UN servo sin carga pesada, pero para trabajo serio necesita su propia fuente.
- Torque: aproximadamente 1.8 kg-cm a 4.8V y 2.2 kg-cm a 6V. Suficiente para mover un sensor, una aguja indicadora, una pequeña garra o la dirección de un carrito ligero. Insuficiente para un brazo robótico grande o para cargar peso considerable.
- Velocidad: alrededor de 0.12 segundos por cada 60 grados a 4.8V. Es decir, tarda poco mas de un tercio de segundo en barrer los 180 grados completos. No es un servo rápido ni lento: es promedio.
- Rango de giro: 180 grados nominales (de 0 a 180). En la práctica, muchas unidades llegan de 0 a 180 reales, pero algunas se quedan en 170 o 175. Por eso existe la calibración, que veremos mas adelante.
- Corriente: en reposo consume poquísimo (unos 5 a 10 miliamperios), moviéndose sin carga unos 100 a 250 miliamperios, y con carga o atorado puede llegar a 600 u 800 miliamperios. Ese pico de corriente es el que causa problemas cuando alimentas varios servos desde el Arduino.
- Peso: 9 gramos. De ahí su apodo: "micro servo de 9g".
- Cables: rojo = alimentación positiva (VCC, 4.8-6V), marron o negro = tierra (GND), naranja o amarillo = senal (se conecta a un pin digital del Arduino).

Hay una version muy parecida llamada MG90S, con engranes metálicos en lugar de plástico. Es mas resistente al desgaste y soporta un poco mas de torque, pero por lo demás se controla exactamente igual. Si algún día rompes los engranes de plástico de tu SG90 por forzarlo (le pasa a todo el mundo una vez), el MG90S es el reemplazo natural.

La librería Servo.h: tus cuatro comandos esenciales

Arduino incluye de fábrica la librería Servo.h, así que no necesitas instalar nada. Con cuatro funciones cubres el 95 por ciento de lo que harás con servos en tu vida. Vamos una por una.

Primero, creas un objeto Servo. Cada servo físico que conectes necesita su propio objeto en el código:

```
Servo miServo;
```

La función `attach(pin)` conecta ese objeto a un pin digital del Arduino. Es donde le dices "el servo físico esta conectado al pin 9". A partir de ese momento, la librería empieza a mandar los pulsos de 50 Hz por ese pin, en segundo plano, sin que tu tengas que hacer nada mas:

```
miServo.attach(9);
```

Hay una forma extendida de `attach` que te permite definir los limites del pulso en microsegundos: `attach(pin, min, max)`. Por ejemplo, `miServo.attach(9, 1000, 2000)` dice que el pulso mínimo es 1000 microsegundos y el máximo 2000. Esto sirve para calibrar servos que no llegan a los 180 grados nominales, como veremos en la sección de calibración. Si no pasas esos valores, la librería usa 544 y 2400 por defecto, un rango mas amplio que el estándar.

La función `write(angulo)` es la que usarás todos los días. Le pasas un angulo entre 0 y 180 y el servo se mueve ahí:

```
miServo.write(90);    // ve al centro  miServo.write(0);      // ve al extremo izquierdo
miServo.write(180);  // ve al extremo derecho
```

Por dentro, `write` convierte el angulo a un ancho de pulso y la librería se encarga de generarlo. Tu no ves los microsegundos, solo piensas en grados. Es la forma mas intuitiva de trabajar.

La función `writeMicroseconds(us)` es la version de precisión. En lugar de grados, le pasas directamente el ancho del pulso en microsegundos, típicamente entre 1000 y 2000:

```
miServo.writeMicroseconds(1500);    // centro, equivalente a write(90)
miServo.writeMicroseconds(1000);    // extremo 0 grados
miServo.writeMicroseconds(2000);    // extremo 180 grados
```

Por que querrías usar microsegundos si ya tienes grados? Porque te da mas resolución. Con `write`, el paso mínimo es 1 grado. Con `writeMicroseconds`, puedes moverte en pasos de un microsegundo, que equivale a fracciones de grado. Para un radar que necesita apuntar con precisión fina, o para calibrar los limites exactos de tu servo, los microsegundos son la herramienta correcta. Además, algunos servos responden de forma mas lineal si los controlas directamente en microsegundos.

La función `read()` te dice la última posición que le ordenaste al servo (el último valor que pasaste a `write()`). Ojo: no mide la posición real del eje, solo recuerda lo que le pediste. Si el servo esta atorado contra un obstáculo y no pudo llegar, `read()` igual te dirá el valor que le ordenaste. Es útil para llevar la cuenta en tu programa, no como sensor de posición real.

Existe también `attached()`, que devuelve verdadero si el objeto Servo esta conectado a un pin, y `detach()`, que deja de mandar pulsos al servo. `detach()` es útil cuando quieres que el servo quede libre (por ejemplo, para moverlo con la mano sin forzar el circuito) o para ahorrar energía. Pero cuidado: al hacer `detach()`, el servo deja de sostener su posición y puede moverse si algo lo empuja.

Alimentación separada: por que el USB no siempre alcanza

Este es el tema que mas proyectos arruina y que menos se explica. Vamos a fondo.

El Arduino Uno, alimentado por USB desde tu computadora, puede entregar unos 500 miliamperios en total por su pin de 5V (y eso compartido con todo lo demás que este conectado). Un solo SG90 moviendose sin carga consume entre 100 y 250 miliamperios, así que con UN servo y sin esfuerzo, el USB aguanta. El problema empieza cuando agregas mas servos, o cuando el servo hace fuerza.

Cuando un servo arranca su movimiento, o cuando empuja contra resistencia, su consumo se dispara a 600 u 800 miliamperios por un instante. Si tienes dos o tres servos moviendose a la vez, el pico total puede superar lo que el USB entrega. Que pasa entonces? El voltaje cae, el Arduino se reinicia solo, los servos vibran o se mueven erraticos, el programa "se vuelve loco". Es el síntoma clásico: "mi proyecto funciona con un servo pero cuando conecto el segundo todo falla". No es tu código. Es la fuente de poder.

La solución profesional es alimentar los servos con una fuente externa y compartir la tierra con el Arduino. Así se hace:

1. Consigue una fuente de 5 a 6 voltios capaz de entregar al menos 1 amperio por cada dos servos (mejor si sobra). Opciones: un paquete de 4 baterías AA recargables (dan unos 4.8V), un eliminador de 5V a 2A, o una fuente regulada de laboratorio.
2. Conecta el positivo de la fuente externa al cable rojo de los servos.
3. Conecta el negativo de la fuente externa al cable marron/negro de los servos Y TAMBIÉN al pin GND del Arduino. Este es el famoso "GND común": todas las tierras conectadas entre si.
4. Conecta solo el cable de señal (naranja/amarillo) de cada servo a su pin digital del Arduino.

5. El Arduino se alimenta por su lado (USB o su propio eliminador).

Por que el GND común es obligatorio? Porque la señal de control es un voltaje RELATIVO a tierra. Cuando el Arduino pone 5V en el pin de señal, el circuito del servo necesita una referencia de "0V" para saber que esos 5V son realmente 5V y no ruido. Si las tierras no están conectadas, el servo no tiene referencia y la señal es ininteligible: el servo tiembla, se mueve solo o no responde. Regla de oro de la robótica: todos los GND del sistema siempre conectados entre si. Sin excepción.

Un error frecuente es conectar la fuente externa al pin VIN del Arduino y de ahí alimentar los servos. No lo hagas: el regulador de voltaje del Arduino se calienta y su capacidad de corriente es limitada. Los servos van directo a la fuente externa, el Arduino va por su lado, y solo comparten GND y las líneas de señal.

Para proyectos con muchos servos (un brazo robótico de 4 o 5 servos, por ejemplo), agrega un capacitor electrolitico grande (470 o 1000 microfaradios) entre positivo y tierra de la fuente de los servos, lo mas cerca posible de ellos. El capacitor actúa como un "tanque" de energía que entrega los picos de corriente instantaneos sin que el voltaje caiga. Es un truco barato que elimina vibraciones misteriosas.

Limites mecanicos y calibración: cada servo es un individuo

En teoría, el SG90 gira de 0 a 180 grados. En la práctica, cada unidad es ligeramente distinta por tolerancias de fabricación: el potenciómetro interno no esta montado exactamente igual en todos, los topes mecanicos varían un poco, y el circuito de control tiene sus propios margenes. Resultado: tu servo particular quizá llegue de 5 a 175 grados reales, o de 0 a 170, o (raro pero pasa) intente ir mas allá de 180 y se atore contra el tope mecánico haciendo un zumbido feo.

Calibrar significa descubrir los limites REALES de tu servo y ajustar el código para trabajar dentro de ellos. Así se hace, paso a paso:

1. Monta un brazo o cuerno en el eje del servo para ver claramente hacia donde apunta. Marca una referencia visual (un transportador impreso o marcas en papel sirven).
2. Escribe un programa que mueva el servo con writeMicroseconds en pasos pequeños, empezando en 1500 (centro) y bajando de 50 en 50: 1450, 1400, 1350... Observa en que valor el servo deja de moverse aunque sigas bajando el número. Ese es tu mínimo real. Supongamos que es 1100 microsegundos.
3. Haz lo mismo hacia arriba: 1550, 1600, 1650... hasta que deje de avanzar. Supongamos que tu máximo real es 1950 microsegundos.

4. Ahora usa `attach(pin, 1100, 1950)` en tu código. Con esto le dices a la librería: "cuando yo pida `write(0)`, manda 1100 microsegundos; cuando pida `write(180)`, manda 1950". El mapeo de grados a microsegundos se ajusta a TU servo real, y `write(90)` quedara en el centro verdadero.

Este procedimiento toma diez minutos y transforma un servo "mas o menos" en un servo preciso. Hazlo con cada servo importante de tus proyectos, porque dos SG90 comprados juntos pueden tener limites distintos.

Los limites mecanicos también importan en el diseño: nunca programes movimientos que empujen el mecanismo contra un tope físico. Si tu proyecto tiene una puerta que el servo abre y cierra, ajusta los ángulos para que la puerta se detenga ANTES de golpear el marco, no después. Un servo atorado contra un tope consume corriente máxima continuamente, se calienta, y con el tiempo daña los engranes de plástico o quema el motor. Si escuchas un zumbido agudo continuo viniendo del servo mientras sostiene posición, algo anda mal: probablemente esta luchando contra un tope o contra una carga excesiva.

Jitter: el temblor misterioso y como eliminarlo

Jitter es el nombre técnico del temblor o vibración pequeña que a veces hace el servo cuando debería estar quieto. El brazo oscila uno o dos grados de un lado a otro, rapidísimo, haciendo un zumbido. Es molesto y en proyectos de precisión arruina las mediciones. Estas son sus causas y soluciones, de la mas común a la mas rara:

Causa 1: fuente de poder débil. Si el voltaje cae cada vez que el servo intenta corregir su posición, el circuito de control entra en un ciclo de sobre-corrección: corrige, el voltaje cae, la corrección falla, vuelve a corregir... La solución es la alimentación separada con GND común que ya vimos, mas el capacitor de respaldo.

Causa 2: órdenes de posición que cambian constantemente. Si tu programa le manda `write(90)`, `write(91)`, `write(90)`, `write(91)` en un ciclo rápido (por ejemplo, porque lees un potenciómetro ruidoso y cada lectura da un valor distinto), el servo intenta seguir esos cambios y tiembla. La solución es filtrar: solo manda una nueva posición si cambio mas de 2 o 3 grados respecto a la anterior, o promedia varias lecturas del potenciómetro antes de decidir.

Causa 3: interferencia en el cable de senal. Si el cable de senal va junto a cables de potencia o es muy largo, puede captar ruido eléctrico que el servo interpreta como cambios de posición. La solución es usar cables cortos, no enrollar el cable de senal junto al de alimentación, y en instalaciones largas usar cable blindado o trenzar senal con tierra.

Causa 4: el servo esta en su límite mecánico. Si le pides 180 pero su máximo real es 175, el circuito empuja contra el tope sin llegar nunca, y eso se ve como vibración. La solución es la calibración con `attach(pin, min, max)` usando tus valores reales.

Un truco final contra el jitter en proyectos donde el servo solo necesita moverse de vez en cuando: usa `detach()` después de llegar a la posición. Sin pulsos de control, el circuito deja de intentar corregir y el temblor desaparece. El servo queda libre (sin fuerza de sostenimiento), así que solo sirve cuando nada empuja el mecanismo. Para un indicador de temperatura con aguja, por ejemplo, funciona perfecto: mueves la aguja, haces `detach`, y la aguja se queda quieta sin vibrar.

Errores comunes (y como evitarlos)

Error 1: forzar el servo con la mano mientras esta energizado. Cuando el servo esta recibiendo pulsos, su circuito de control esta luchando por mantener la posición. Si tu lo giras a la fuerza con la mano, el motor trabaja en contra tuya consumiendo corriente máxima, los engranes de plástico sufren un estres brutal y puedes barrer los dientes del engrane principal. Si necesitas mover el eje con la mano, primero haz `detach()` en el código o desconecta la alimentación. Y nunca, nunca gires el eje mas allá de sus topes mecanicos: sentiras una resistencia dura al final del recorrido, ahí debes detenerte.

Error 2: alimentar varios servos desde el pin 5V del Arduino. Con un servo pequeño y sin carga puedes salirte con la tuya en pruebas rápidas. Con dos o mas, o con carga, el regulador del Arduino se sobrecarga, el voltaje cae y todo el sistema se vuelve inestable: reinicios, movimientos erraticos, lecturas de sensores que se vuelven locas. La regla es simple: un servo para pruebas rápidas puede ir al 5V del Arduino; dos o mas servos, o cualquier servo haciendo fuerza real, van a fuente externa con GND común. Sin excepciones.

Error 3: olvidar el GND común con fuente externa. Ya lo explicamos, pero lo repetimos porque es el error número uno en foros de ayuda: "conecte mi servo a una batería externa y ahora no obedece". Conectaste el positivo y la señal, pero olvidaste unir la tierra de la batería con la tierra del Arduino. El servo recibe una señal sin referencia y no entiende nada. Siempre: GND de la fuente externa unido al GND del Arduino.

Error 4: usar los pines 9 y 10 para PWM mientras usas la librería Servo. La librería Servo.h en el Arduino Uno toma control del temporizador que genera el PWM en los pines 9 y 10. Si en tu programa usas `analogWrite(9, valor)` o `analogWrite(10, valor)` al mismo tiempo que servos, el PWM de esos pines dejará de funcionar correctamente. Usa otros pines PWM (3, 5, 6, 11) para tus LEDs o motores DC cuando trabajes con servos.

Error 5: mandar al servo mas allá de sus límites con write. Si tu servo calibrado llega de 1100 a 1950 microsegundos y tu le pides write(0) con un attach por defecto (544 a 2400), el servo intentara ir a 544 microsegundos, chocara contra su tope mecánico interno y se quedara zumbando, consumiendo corriente máxima y calentandose. Calibra y respeta los límites reales.

Error 6: delays bloqueantes gigantes entre movimientos. Si pones delay(5000) después de cada write, tu programa queda congelado cinco segundos sin poder leer botones ni sensores. Para secuencias simples esta bien, pero en cuanto tu proyecto necesite hacer dos cosas a la vez (mover el servo Y leer un sensor), necesitarás la técnica de millis() no bloqueante que ya conoces de unidades anteriores. La aplicaremos en el proyecto del radar.

Error 7: no montar bien el cuerno del servo. Los cuernos (los bracitos de plástico) se fijan al eje estriado con un tornillo pequeño que viene en la bolsita del servo. Si no aprietas ese tornillo, el cuerno puede girar libre sobre el eje y tu proyecto "pierde" la posición: el servo gira pero el mecanismo no. Aprieta el tornillo, pero sin excederte: si lo fuerzas demasiado puedes barrer la rosca de plástico.

Error 8: confundir los cables. Rojo a positivo, marron/negro a GND, naranja/amarillo a señal. Si inviertes positivo y tierra, puedes dañar el servo instantaneamente. Antes de energizar, verifica dos veces con la vista: sigue cada cable desde el servo hasta su destino. Los servos no traen protección contra polaridad inversa en su mayoría.

Con la teoría clara, es hora de ensuciarnos las manos. En la siguiente sección vas a conectar tu primer servo y a moverlo con código real, paso a paso, explicando cada línea.

Ejemplos paso a paso

Antes del primer ejemplo, las conexiones base que usaremos en toda la sección. Son solo tres cables:

- Cable rojo del servo -> 5V del Arduino (para UN servo en pruebas; con mas servos usa fuente externa con GND común como aprendiste).
- Cable marron o negro del servo -> GND del Arduino.
- Cable naranja o amarillo del servo -> pin digital 9 del Arduino.

Monta un cuerno en el eje del servo para ver el movimiento. Listo. Empezamos.

Ejemplo A: barrido básico de 0 a 180 (el clásico Sweep)

Este es el "Hola mundo" de los servos: el brazo barre de un extremo al otro, una y otra vez, como un limpiaparabrisas. Es el ejemplo Sweep que viene con el IDE de Arduino, aquí explicado línea por línea para que no sea magia negra.

```
#include <Servo.h>    // Incluimos la libreria que genera los pulsos de control

Servo miServo;        // Creamos el objeto que representa a nuestro servo fisico
int angulo = 0;        // Variable para llevar la cuenta del angulo actual

void setup() {
  miServo.attach(9); // El servo esta en el pin 9: empieza a mandar pulsos de 50 Hz
}

void loop() {
  // Primera mitad: barrer de 0 a 180 grados, de uno en uno
  for (angulo = 0; angulo <= 180; angulo += 1) {
    miServo.write(angulo); // Ordena ir al angulo actual
    delay(15);             // Espera 15 ms para dar tiempo a moverse 1 grado
  }
  // Segunda mitad: barrer de regreso, de 180 a 0
  for (angulo = 180; angulo >= 0; angulo -= 1) {
    miServo.write(angulo); // Ordena ir al angulo actual
    delay(15);             // Espera 15 ms entre cada paso
  }
}
```

Que esta pasando aquí? En setup, attach(9) le dice a la librería que genere la señal de pulsos en el pin 9. Desde ese momento, aunque tu programa no haga nada mas, el servo recibe su "toque en el hombro" cada 20 milisegundos y sostiene la última posición ordenada.

En loop, el primer for sube el angulo de 0 a 180 en pasos de 1 grado. Cada iteración manda write(angulo) y espera 15 milisegundos. Ese delay es clave: el SG90 tarda unos 0.12 segundos en girar 60 grados, o sea unos 2 milisegundos por grado. Esperar 15 ms por grado le da tiempo de sobra para llegar antes de recibir la siguiente orden, así el movimiento se ve fluido en lugar de nervioso. El segundo for hace lo mismo pero de regreso.

Experimenta cambiando los valores: si pones delay(5), el barrido será mas rápido pero quizás menos suave; si pones angulo += 5 en lugar de += 1, el movimiento será a saltos visibles. Juega con estos numeros para sentir como responde el servo. Esa experimentacion vale mas que mil páginas de teoría.

Ejemplo B: mover a posiciones con botones

Un barrido automático es bonito, pero un robot necesita ir a posiciones ESPECÍFICAS cuando se lo ordenan. En este ejemplo, tres botones llevan el servo a tres posiciones: 0, 90 y 180 grados. Es la base de cosas como "presiona el botón y la garra se abre".

Conexiones adicionales: tres botones (pushbuttons). Cada botón va entre un pin digital (2, 3 y 4) y GND. Usaremos las resistencias pull-up internas del Arduino, así no necesitas resistencias externas: el pin lee HIGH cuando el botón esta suelto y LOW cuando lo presionas.

```
#include <Servo.h>

Servo miServo;
// Pines de los botones (usamos pull-up interno: presionado = LOW)
const int BTN_IZQ = 2;    // lleva el servo a 0 grados
const int BTN_CEN = 3;    // lleva el servo a 90 grados
const int BTN_DER = 4;    // lleva el servo a 180 grados

void setup() {
  miServo.attach(9);
  pinMode(BTN_IZQ, INPUT_PULLUP);
  pinMode(BTN_CEN, INPUT_PULLUP);
  pinMode(BTN_DER, INPUT_PULLUP);
  miServo.write(90); // empieza en el centro
}

void loop() {
  // Leemos cada boton. Con pull-up, LOW significa "presionado".
  if (digitalRead(BTN_IZQ) == LOW) {
    miServo.write(0);
    delay(300); // antirrebote sencillo: ignora pulsaciones por 300 ms
  }
  if (digitalRead(BTN_CEN) == LOW) {
    miServo.write(90);
    delay(300);
  }
  if (digitalRead(BTN_DER) == LOW) {
    miServo.write(180);
    delay(300);
  }
}
```

Fíjate en el detalle del antirrebote: los botones mecanicos "rebotan" al presionarse, generando varios LOW rapidísimos en cuestion de milisegundos. El delay(300) después de cada

detección evita que un solo presionado se lea como varios. Es una solución sencilla y suficiente para este caso. En proyectos mas exigentes usarías antirrebote con `millis()`, que ya dominas de unidades anteriores.

Nota que `write()` mueve el servo a la máxima velocidad que puede: el brazo "salta" a la posición. A veces quieres eso (una garra que se cierra rápido), y a veces quieres movimientos suaves y elegantes. Para eso esta el ejemplo D.

Ejemplo C: control con potenciómetro (el servo te obedece al instante)

Este ejemplo es mágico la primera vez que lo pruebas: giras la perilla de un potenciómetro con los dedos y el servo copia tu movimiento en tiempo real. Es control de posición directo, la base de los brazos robóticos teledirigidos y de las "manos" que imitan tu mano.

Conexiones: un potenciómetro de 10k ohmios. Sus patas de los extremos van a 5V y GND, la pata del centro (el cursor) va al pin analógico A0. El servo sigue en el pin 9 con sus tres cables como antes.

```
#include <Servo.h>

Servo miServo;

const int PIN_POT = A0;    // el cursor del potenciómetro
int lectura = 0;           // lo que lee el ADC: 0 a 1023
int angulo = 0;            // el angulo calculado: 0 a 180
int anguloAnterior = -1;   // para no mandar ordenes repetidas (anti-jitter)

void setup() {
    miServo.attach(9);
}

void loop() {
    lectura = analogRead(PIN_POT);           // lee la perilla: 0 (izquierda) a 1023
                                              (derecha)
    angulo = map(lectura, 0, 1023, 0, 180); // convierte el rango a grados de servo

    // Solo ordena si el angulo cambio al menos 2 grados: evita el temblor
    if (abs(angulo - anguloAnterior) >= 2) {
        miServo.write(angulo);
        anguloAnterior = angulo;
    }
    delay(20); // pequena pausa para estabilizar
}
```

La estrella aquí es la función `map()`. El potenciómetro te da valores de 0 a 1023 (la resolución del convertidor analógico-digital del Arduino), pero el servo entiende de 0 a 180 grados. `map(lectura, 0, 1023, 0, 180)` hace la conversión proporcional: si la perilla esta a la mitad (lectura ~512), el angulo será ~90. Es una regla de tres automatizada y la usarás constantemente en robótica para traducir entre mundos de valores distintos.

El truco del `anguloAnterior` es la aplicación directa de lo que aprendiste sobre jitter: el potenciómetro real siempre tiene un poco de ruido eléctrico, así que la lectura brinca entre valores vecinos aunque no toques la perilla. Sin el filtro, el servo recibiría `write(90)`, `write(91)`, `write(90)`... todo el tiempo y temblaría. Exigiendo un cambio mínimo de 2 grados, el servo solo se mueve cuando TU mueves la perilla de verdad. Prueba quitar ese `if` y verás el jitter en acción: es la mejor lección práctica sobre el tema.

Ejemplo D: movimientos suaves con pasos pequeños

El ejemplo B movia el servo a saltos bruscos. Muchas aplicaciones necesitan elegancia: la cabeza de un robot que gira despacio para "mirar", una puerta que se abre con suavidad, una cámara que hace un paneo cinematografico. La técnica es sencilla: en lugar de ordenar el destino final de golpe, avanzas grado por grado (o fracción de grado) con pequeñas pausas. Es como subir una escalera peldaio por peldaio en vez de saltar al techo.

```

#include <Servo.h>

Servo miServo;
int posicionActual = 90; // llevamos la cuenta de donde esta el servo

// Mueve el servo DESPACIO desde donde este hasta el destino.
// pasos: de cuanto en cuanto avanza (1 = super suave, 5 = mas rapido)
// esperaMs: pausa entre pasos (mas ms = mas lento)
void moverSuave(int destino, int pasos, int esperaMs) {
    if (destino > posicionActual) {
        for (int a = posicionActual; a <= destino; a += pasos) {
            miServo.write(a);
            delay(esperaMs);
        }
    } else {
        for (int a = posicionActual; a >= destino; a -= pasos) {
            miServo.write(a);
            delay(esperaMs);
        }
    }
    posicionActual = destino; // actualizamos donde quedo
}

void setup() {
    miServo.attach(9);
    miServo.write(posicionActual); // empieza en el centro
    delay(500);
}

void loop() {
    moverSuave(0, 1, 20); // ve a 0 grados, super despacio y suave
    delay(1000);           // pausa de 1 segundo en el extremo
    moverSuave(180, 1, 20); // ve a 180 grados, igual de suave
    delay(1000);
    moverSuave(90, 2, 10); // regresa al centro, un poco mas rapido
    delay(2000);
}

```

La función moverSuave() es reutilizable: la copias a cualquier proyecto y tienes movimientos elegantes con una sola línea. Los dos parámetros te dan control total sobre la "coreografía": pasos pequeños con esperas largas dan movimientos lentísimos y ceremoniosos; pasos grandes con esperas cortas dan movimientos rápidos pero aun controlados.

Un detalle importante: la variable `posicionActual` lleva la cuenta de donde está el servo. Sin ella, la función no sabría desde donde empezar a moverse. Es un ejemplo del patrón "el programa recuerda el estado del mundo físico", fundamental en robótica: tu código debe saber donde están las cosas, no solo ordenar movimientos a ciegas.

Si necesitas aun mas suavidad que 1 grado por paso, combina esta técnica con `writeMicroseconds`: en lugar de pasos de 1 grado, avanza de 10 en 10 microsegundos. El movimiento se vuelve practicamente continuo, como el de los robots caros. Lo verás aplicado en los ejercicios.

Ejemplo E: radar con servo y HC-SR04 (el proyecto estrella)

Ahora juntamos todo: el servo barre el ambiente de 0 a 180 grados, y en cada angulo el sensor ultrasónico HC-SR04 mide la distancia al objeto mas cercano. El resultado se reporta por el monitor serial como pares "angulo: distancia". Es un radar de verdad, como los de las películas, y es la base de los robots que mapean su entorno.

Conexiones: el servo como siempre (senal al pin 9, rojo a 5V, marron a GND). El HC-SR04: VCC a 5V, GND a GND, TRIG al pin 7, ECO al pin 6. Monta el sensor sobre el cuerno del servo (con cinta adhesiva o un pequeño soporte) de modo que el sensor apunte hacia adelante y gire junto con el brazo del servo. Los "ojos" del sensor (los dos cilindros) deben mirar al frente.

```
#include <Servo.h>

Servo radar;
const int PIN_TRIG = 7;
const int PIN_ECO = 6;
const int PIN_SERVO = 9;

// Mide la distancia en centimetros con el HC-SR04
long medirDistancia() {
    digitalWrite(PIN_TRIG, LOW);
    delayMicroseconds(2);
    digitalWrite(PIN_TRIG, HIGH);    // pulso de disparo de 10 microsegundos
    delayMicroseconds(10);
    digitalWrite(PIN_TRIG, LOW);
    long duracion = pulseIn(PIN_ECO, HIGH);    // tiempo de ida y vuelta del eco
    long distancia = duracion / 58;           // conversion a centimetros
    return distancia;
}

void setup() {
```

```

Serial.begin(9600);
radar.attach(PIN_SERVO);
pinMode(PIN_TRIG, OUTPUT);
pinMode(PIN_ECO, INPUT);
radar.write(90);
delay(1000);
Serial.println("RADAR INICIADO: barriendo de 0 a 180 grados");
Serial.println("Formato -> angulo: distancia_cm");
}

void loop() {
  // Barrido de ida: 0 a 180, midiendo en cada paso de 5 grados
  for (int ang = 0; ang <= 180; ang += 5) {
    radar.write(ang);
    delay(60); // espera a que el servo llegue y se estabilice
    long d = medirDistancia();
    Serial.print(ang);
    Serial.print(": ");
    Serial.print(d);
    Serial.println(" cm");
  }
  // Barrido de regreso: 180 a 0
  for (int ang = 180; ang >= 0; ang -= 5) {
    radar.write(ang);
    delay(60);
    long d = medirDistancia();
    Serial.print(ang);
    Serial.print(": ");
    Serial.print(d);
    Serial.println(" cm");
  }
  Serial.println("--- barrido completo ---");
  delay(500);
}

```

Hay tres decisiones de diseño que debes entender en este código:

Primera: medimos en pasos de 5 grados, no de 1. El HC-SR04 tiene un "cono" de detección de unos 15 grados de ancho, así que medir cada grado seria redundante: verías el mismo objeto cinco veces. Cada 5 grados obtienes 37 mediciones por barrido, suficiente resolución sin datos repetidos ni barridos eternos.

Segunda: el delay(60) después de cada write. El servo necesita tiempo para llegar al nuevo angulo Y para dejar de vibrar antes de medir. Si mides mientras el sensor aun se esta moviendo, la distancia sale mal porque el eco se toma en una posición intermedia. Esos 60

milisegundos son el tiempo de "asentamiento". En tu radar físico puedes ajustarlo: si ves lecturas inestables, subelo a 80 o 100.

Tercera: el formato de salida "ángulo: distancia". Esta pensado para que puedas copiar los datos a una hoja de cálculo o a un programa que dibuje el mapa. De hecho, en el último ejercicio de la sección de práctica vas a convertir estos datos en un mapa visual del cuarto usando el graficador serial. Pero no te adelantes: primero haz funcionar el radar básico, pon objetos a distintas distancias (un libro a 30 cm, una caja a 80 cm) y observa como los números cambian según el ángulo. Cuando veas que el número baja justo cuando el sensor apunta al libro, habrás construido tu primer sistema de percepción espacial. Eso, en robótica, es un momento que se celebra.

Con estos cinco ejemplos tienes el repertorio completo del servo: barridos, posiciones discretas, control continuo con potenciómetro, movimientos suaves programados y percepción con radar. La siguiente sección te pone a prueba con ejercicios que combinan estas técnicas en proyectos reales.

Práctica

Ejercicio 1: puerta automática con botón

Enunciado: construye una puerta automática en miniatura. Un servo controla la puerta (un cartoncito pegado al cuerno del servo). Al presionar un botón, la puerta se ABRE suavemente hasta 90 grados, espera 3 segundos, y se CIERRA suavemente hasta 0 grados. Mientras la puerta se mueve o esta abierta, un LED debe estar encendido como "luz de cortesía".

Objetivo: practicar movimientos suaves en ambas direcciones, coordinar el servo con otros componentes (LED y temporización), y usar una función reutilizable de movimiento. La puerta no debe "azotar": la apertura y el cierre deben verse controlados.

Solución del ejercicio 1

```

#include <Servo.h>

Servo puerta;
const int PIN_SERVO = 9;
const int PIN_BOTON = 2;
const int PIN_LED = 13;
int posicionActual = 0;

// Mueve la puerta suavemente hasta el destino, grado por grado
void moverPuerta(int destino) {
    if (destino > posicionActual) {
        for (int a = posicionActual; a <= destino; a++) {
            puerta.write(a);
            delay(20);    // 20 ms por grado: apertura elegante
        }
    } else {
        for (int a = posicionActual; a >= destino; a--) {
            puerta.write(a);
            delay(20);
        }
    }
    posicionActual = destino;
}

void setup() {
    puerta.attach(PIN_SERVO);
    pinMode(PIN_BOTON, INPUT_PULLUP);
    pinMode(PIN_LED, OUTPUT);
    puerta.write(0); // la puerta empieza cerrada
    delay(500);
}

void loop() {
    // Espera a que presionen el boton (con pull-up, LOW = presionado)
    if (digitalRead(PIN_BOTON) == LOW) {
        digitalWrite(PIN_LED, HIGH); // enciende la luz de cortesia
        moverPuerta(90);             // abre suavemente
        delay(3000);                  // puerta abierta 3 segundos
        moverPuerta(0);               // cierra suavemente
        digitalWrite(PIN_LED, LOW);  // apaga la luz
        delay(500);                   // antirrebote: evita re-disparos
    }
}

```


Explicación: la función `moverPuerta()` es la version simplificada de `moverSuave()` del ejemplo D, fija a pasos de 1 grado y 20 ms de espera. La puerta tarda 90 grados x 20 ms = 1.8 segundos en abrirse: se ve mecánica y elegante, no brusca. El LED se enciende antes de empezar a abrir y se apaga después de cerrar, simulando la luz de cortesía de las puertas automáticas reales. El `delay(500)` final evita que el rebote del botón dispare la secuencia dos veces. Si quieres una puerta mas dramática, baja el delay a 10 ms para abrir rápido... pero entonces ya no seria "automática elegante", seria puerta de película de acción.

Ejercicio 2: indicador de temperatura con aguja-servo

Enunciado: usa un sensor de temperatura LM35 (o un TMP36) y un servo con una aguja de cartón para construir un termómetro analógico como los de antes. El servo debe apuntar a 0 grados cuando la temperatura sea 15 C o menos, y a 180 grados cuando sea 35 C o mas, con escala proporcional en medio. La aguja no debe temblar: solo se mueve cuando la temperatura cambia de verdad.

Objetivo: practicar la función `map()` entre rangos no obvios (temperatura a grados), leer sensores analógicos, y aplicar filtrado anti-jitter en un instrumento de medicion. Bonus: dibuja una caratula de papel con marcas de 15 a 35 grados y pegala detrás de la aguja.

Solución del ejercicio 2

```

#include <Servo.h>

Servo aguja;
const int PIN_SERVO = 9;
const int PIN_TEMP = A0;    // LM35: salida analogica proporcional a la temperatura
int anguloAnterior = -1;

float leerTemperatura() {
    int lectura = analogRead(PIN_TEMP);    // 0 a 1023
    float voltaje = lectura * (5.0 / 1023.0); // convierte a voltios
    float tempC = voltaje * 100.0;          // LM35: 10 mV por grado -> V * 100
    return tempC;
}

void setup() {
    aguja.attach(PIN_SERVO);
    Serial.begin(9600);
}

void loop() {
    float temp = leerTemperatura();

    // Convierte temperatura (15-35 C) a angulo (0-180 grados).
    // constrain() evita que valores fuera de rango rompan la escala.
    int angulo = map((int)temp, 15, 35, 0, 180);
    angulo = constrain(angulo, 0, 180);

    // Solo mueve la aguja si el cambio es de al menos 2 grados: sin temblor
    if (abs(angulo - anguloAnterior) >= 2) {
        aguja.write(angulo);
        anguloAnterior = angulo;
        Serial.print("Temp: ");
        Serial.print(temp);
        Serial.print(" C -> aguja: ");
        Serial.println(angulo);
    }
    delay(500); // el LM35 no necesita leerse mas rapido
}

```

Explicación: el LM35 entrega 10 milivoltios por cada grado Celsius, así que convertir la lectura analógica a temperatura es cuestión de pasar a voltios y multiplicar por 100. Luego map() traduce el rango útil (15 a 35 grados, temperaturas ambiente razonables) al rango del servo (0 a 180). La función constrain() es el guardaespaldas: si la temperatura sale del rango (un día de 40 C o una mañana de 10 C), el angulo se queda clavado en 0 o 180 en lugar de pedirle al

servo un valor imposible. El filtro de 2 grados elimina el temblor: el LM35 tiene un poco de ruido y sin el filtro la aguja vibraría todo el tiempo. Para calentar el sensor y ver la aguja moverse, aprieta el LM35 entre tus dedos unos segundos: tu calor corporal (unos 36 C) llevara la aguja casi al máximo. Es el termómetro mas satisfactorio que vas a construir.

Ejercicio 3: barrido con velocidad variable

Enunciado: modifica el barrido clásico para que la velocidad del barrido se controle con un potenciómetro: girando la perilla, el barrido va de lentísimo a rapidísimo. Además, el servo debe moverse de forma suave en todo el rango de velocidades (sin saltos), y la velocidad actual debe mostrarse en el monitor serial como "lento", "medio" o "rápido".

Objetivo: combinar lectura analógica con temporización de movimientos, entender que la velocidad de un servo se controla con el tamaño de los pasos y los tiempos de espera, y practicar la traducción de un valor continuo a categorías de texto.

Solución del ejercicio 3

```

#include <Servo.h>

Servo miServo;
const int PIN_SERVO = 9;
const int PIN_POT = A0;
int angulo = 0;
int direccion = 1; // 1 = subiendo hacia 180, -1 = bajando hacia 0

void setup() {
    miServo.attach(PIN_SERVO);
    Serial.begin(9600);
}

void loop() {
    int lectura = analogRead(PIN_POT); // 0 a 1023

    // Traduce la perilla a pausa entre pasos: 2 ms (rapidismo) a 50 ms (lentisimo).
    // Ojo: el map esta invertido, mas perilla = menos espera = mas velocidad.
    int espera = map(lectura, 0, 1023, 50, 2);

    // Muestra la categoria de velocidad cada 50 ciclos (para no saturar el serial)
    static int contador = 0;
    if (++contador >= 50) {
        contador = 0;
        Serial.print("Velocidad: ");
        if (espera > 33) Serial.println("lento");
        else if (espera > 15) Serial.println("medio");
        else Serial.println("rapido");
    }

    miServo.write(angulo);
    delay(espera);

    angulo += direccion; // avanza 1 grado en la direccion actual
    if (angulo >= 180) { direccion = -1; angulo = 180; } // rebota en el tope
    if (angulo <= 0) { direccion = 1; angulo = 0; } // rebota en el otro
}

```

Explicación: en lugar de dos ciclos for (ida y vuelta), este código usa una sola variable de dirección que se invierte en los extremos: es el patrón "rebotar", mas elegante y fácil de extender. La velocidad se controla con la pausa entre pasos de 1 grado: 50 ms por grado da un barrido completo en 9 segundos (lento y majestuoso), 2 ms por grado lo hace en 0.36 segundos (casi lo máximo que el SG90 puede dar). El map invertido (de 50 a 2 en lugar de 2 a

50) hace que girar la perilla a la derecha acelere, que es lo intuitivo. La categoría de velocidad se imprime cada 50 ciclos para no inundar el monitor serial con texto. Prueba llevar la perilla al máximo y observa: el servo llega a su límite físico de velocidad y ya no puede ir mas rápido aunque bajes mas la espera. Ese es el techo real del SG90, unos 0.12 segundos por 60 grados, y ningún código lo puede superar.

Ejercicio 4: dos servos sincronizados (aplauso robótico)

Enunciado: conecta DOS servos (pines 9 y 10) con un brazo cada uno, uno frente al otro, como dos manos a punto de aplaudir. Programa una secuencia coordinada: ambos brazos se abren a la vez (servo A a 0, servo B a 180), esperan medio segundo, y se juntan de golpe (servo A a 90, servo B a 90) simulando un aplauso. Repite el aplauso 3 veces, luego los brazos hacen una "reverencia" (ambos bajan suavemente a una posición de descanso). Un botón inicia la secuencia.

Objetivo: practicar el control de múltiples servos con una sola librería, coordinar movimientos simultáneos (los dos servos se mueven AL MISMO TIEMPO, no uno después del otro), y notar el aumento de consumo de corriente: este es el ejercicio donde la fuente externa con GND común deja de ser teoría y se vuelve necesaria.

Solución del ejercicio 4

```
#include <Servo.h>

Servo brazoA;    // mano izquierda, pin 9
Servo brazoB;    // mano derecha, pin 10
const int PIN_BOTON = 2;

void setup() {
  brazoA.attach(9);
  brazoB.attach(10);
  pinMode(PIN_BOTON, INPUT_PULLUP);
  // Posicion de descanso inicial: brazos abajo y separados
  brazoA.write(0);
  brazoB.write(180);
  delay(800);
}

// Mueve AMBOS servos a la vez, paso a paso, para movimientos sincronizados
void moverAmbos(int destinoA, int destinoB, int esperaMs) {
  int actualA = brazoA.read(); // recuerda donde esta cada uno
  int actualB = brazoB.read();
```

```

// Calcula cuantos pasos necesita el que mas tiene que moverse
int pasos = max(abs(destinoA - actualA), abs(destinoB - actualB));
for (int i = 1; i <= pasos; i++) {
    // Interpola la posicion de cada servo: avanzan juntos, al mismo ritmo
    int posA = map(i, 0, pasos, actualA, destinoA);
    int posB = map(i, 0, pasos, actualB, destinoB);
    brazoA.write(posA);
    brazoB.write(posB);
    delay(esperaMs);
}
}

void aplaudir() {
    for (int n = 0; n < 3; n++) {
        moverAmbos(0, 180, 8);    // abrir: rapido pero controlado
        delay(300);
        moverAmbos(90, 90, 4);    // juntar: golpe de aplauso
        delay(400);
    }
}

void loop() {
    if (digitalRead(PIN_BOTON) == LOW) {
        aplaudir();                // tres aplausos sincronizados
        moverAmbos(30, 150, 25);   // reverencia: bajan despacio y elegantes
        delay(1500);
        moverAmbos(0, 180, 25);    // regresan a descanso
        delay(500);               // antirrebote
    }
}

```

Explicación: el corazón es moverAmbos(), que resuelve el problema de sincronizar dos servos que tienen que recorrer distancias distintas. Si el brazo A debe moverse 90 grados y el B solo 20, y los movieras grado por grado al mismo ritmo, el B terminaría antes y se quedaría esperando. La solución es calcular cuántos pasos necesita el que mas recorre, e interpolar la posición del otro con map() para que ambos lleguen a su destino en el mismo número de pasos: se mueven juntos, al mismo ritmo, y llegan al mismo tiempo. Es la misma matematica que usan los brazos robóticos reales para coordinar articulaciones. La función read() sirve aquí para saber donde esta cada servo antes de empezar a interpolar. Y un aviso práctico: con dos servos aplaudiendo a toda velocidad, el pico de corriente es considerable. Si los alimentas del USB del Arduino, verás reinicios o movimientos débiles justo en el golpe del aplauso. Este es el momento de conectar tu fuente externa con GND común y comprobar la diferencia: el aplauso se vuelve firme y contundente.

Ejercicio 5: cabeza que sigue objetos cercanos (el robot te mira)

Enunciado: monta el HC-SR04 sobre el servo como en el ejemplo del radar, pero ahora el comportamiento es reactivo: el servo barre continuamente de 30 a 150 grados en pasos de 10 grados, midiendo la distancia en cada posición. Cuando detecta un objeto a menos de 40 cm, DEJA de barrer y apunta fijo hacia ese objeto (la "cabeza" te esta mirando). Si el objeto se mueve, la cabeza lo sigue: cada 200 ms re-mide y ajusta. Si el objeto desaparece (distancia mayor a 50 cm), la cabeza retoma el barrido de búsqueda. Un LED parpadea rápido mientras "sigue" un objeto.

Objetivo: combinar barrido, medicion y lógica de estados (buscar vs seguir) en un solo programa; practicar histéresis (umbrales distintos para entrar y salir del modo seguimiento, para que no oscile); y construir tu primer comportamiento robótico autónomo reconocible: un robot que te mira cuando te acercas.

Solución del ejercicio 5

```
#include <Servo.h>

Servo cabeza;
const int PIN_SERVO = 9;
const int PIN_TRIG = 7;
const int PIN_ECO = 6;
const int PIN_LED = 13;

const int UMBRAL_CERCA = 40; // menos de 40 cm: hay alguien, modo SEGUIR
const int UMBRAL_LEJOS = 50; // mas de 50 cm: se fue, vuelve a BUSCAR
// (La diferencia de 10 cm es la histeresis: evita que oscile entre modos)

bool siguiendo = false;
int anguloCabeza = 90;

long medirDistancia() {
    digitalWrite(PIN_TRIG, LOW);
    delayMicroseconds(2);
    digitalWrite(PIN_TRIG, HIGH);
    delayMicroseconds(10);
    digitalWrite(PIN_TRIG, LOW);
    return pulseIn(PIN_ECO, HIGH) / 58; // centimetros
}

void setup() {
    cabeza.attach(PIN_SERVO);
    pinMode(PIN_TRIG, OUTPUT);
```

```

pinMode(PIN_ECO, INPUT);
pinMode(PIN_LED, OUTPUT);
cabeza.write(anguloCabeza);
delay(800);
Serial.begin(9600);
Serial.println("Cabeza buscadora lista. Acercate y te mirare.");
}

void loop() {
  if (!siguiendo) {
    // MODO BUSCAR: barrido de 30 a 150 grados, midiendo en cada paso
    digitalWrite(PIN_LED, LOW);
    for (int a = 30; a <= 150 && !siguiendo; a += 10) {
      cabeza.write(a);
      anguloCabeza = a;
      delay(80); // deja que el servo llegue y se asiente
      long d = medirDistancia();
      Serial.print("Buscando en ");
      Serial.print(a);
      Serial.print(" grados: ");
      Serial.print(d);
      Serial.println(" cm");
      if (d < UMBRAL_CERCA) {
        siguiendo = true; // objeto cercano: cambia a modo SEGUIR
        Serial.println("Objeto detectado: siguiendo.");
      }
    }
    // Barrido de regreso
    for (int a = 150; a >= 30 && !siguiendo; a -= 10) {
      cabeza.write(a);
      anguloCabeza = a;
      delay(80);
      long d = medirDistancia();
      if (d < UMBRAL_CERCA) {
        siguiendo = true;
        Serial.println("Objeto detectado: siguiendo.");
      }
    }
  } else {
    // MODO SEGUIR: re-mide y ajusta la cabeza hacia el objeto
    digitalWrite(PIN_LED, HIGH);
    long d = medirDistancia();
    Serial.print("Siguiendo a ");
    Serial.print(d);
    Serial.println(" cm");
  }
}

```



```

if (d > UMBRAL_LEJOS) {
  siguiendo = false; // el objeto se fue: vuelve a buscar
  digitalWrite(PIN_LED, LOW);
  Serial.println("Objeto perdido: buscando de nuevo.");
} else {
  // Pequena exploracion: prueba 10 grados a cada lado y quedate
  // con el angulo donde el objeto se ve mas cerca (mas centrado)
  long dIzq, dCen, dDer;
  cabeza.write(anguloCabeza - 10); delay(120); dIzq = medirDistancia();
  cabeza.write(anguloCabeza); delay(120); dCen = medirDistancia();
  cabeza.write(anguloCabeza + 10); delay(120); dDer = medirDistancia();
  // Elige el angulo con la menor distancia (el objeto mas de frente)
  if (dIzq < dCen && dIzq < dDer) anguloCabeza -= 10;
  else if (dDer < dCen && dDer < dIzq) anguloCabeza += 10;
  anguloCabeza = constrain(anguloCabeza, 30, 150);
  cabeza.write(anguloCabeza);
  // Parpadeo del LED: señal de "te estoy viendo"
  digitalWrite(PIN_LED, LOW); delay(100); digitalWrite(PIN_LED, HIGH);
}
delay(200);
}
}

```

Explicación: el programa tiene dos estados claros, BUSCAR y SEGUIR, controlados por la variable booleana siguiendo. En BUSCAR, el servo barre y mide; si algo aparece a menos de 40 cm, cambia de estado. En SEGUIR, la cabeza hace una mini-exploración: mide a la izquierda, al centro y a la derecha de su posición actual, y se mueve hacia donde el objeto se ve mas cerca, que es donde esta mas de frente al sensor. Así la cabeza "persigue" tu mano si la mueves despacio de un lado a otro. La histéresis (entrar a 40 cm, salir a 50 cm) evita el parpadeo de estados: sin ella, un objeto justo a 40 cm haría que el robot entrará y saliera del modo seguir varias veces por segundo, comportandose como nervioso. El constrain() final impide que la cabeza intente salir del rango 30-150 aunque el objeto se mueva mas allá. Cuando lo pruebes, mueve tu mano LENTO frente al sensor: si la mueves rápido, el ciclo de medicion (unos 120 ms por lectura) no alcanza a seguirte y te "pierde". Esa limitación de velocidad de reacción es real en todos los robots: percibir toma tiempo, y el diseño debe respetarlo.

Resumen de la unidad

El servomotor es el actuador de precisión de la robótica: a diferencia del motor DC, que gira sin parar, el servo va a una posición angular exacta y la sostiene. Por dentro combina cuatro elementos: un motor DC pequeño que genera el movimiento, un tren de engranes que

multiplica la fuerza, un potenciómetro interno que informa la posición actual, y un circuito de control que compara la posición deseada con la real y corrige el error cientos de veces por segundo. Ese lazo cerrado de control es lo que le permite obedecer órdenes de posición y resistir perturbaciones.

La comunicación con el servo usa pulsos cada 20 milisegundos donde el ancho codifica la posición: 1 ms significa 0 grados, 1.5 ms el centro (90 grados) y 2 ms los 180 grados. El Arduino genera esta señal con la librería Servo.h, cuyas funciones esenciales son attach(pin) para conectar el servo, write(angulo) para órdenes en grados, writeMicroseconds(us) para precisión fina, read() para recordar la última orden, y detach() para liberar el servo. En el Arduino Uno puedes manejar hasta 12 servos, pero recuerda que la librería desactiva el PWM de los pines 9 y 10.

El servo SG90, tu compañero en esta unidad, trabaja de 4.8 a 6 voltios, entrega alrededor de 1.8 kg-cm de torque y gira 180 grados nominales. Su talon de Aquiles es la corriente: en esfuerzo puede consumir hasta 800 miliamperios, así que dos o mas servos (o uno solo haciendo fuerza) necesitan fuente externa con GND común, nunca el pin 5V del Arduino. El GND común no es opcional: sin tierras unidas, la señal de control no tiene referencia y el servo no entiende nada.

Cada servo es un individuo: calibra sus limites reales con writeMicroseconds y fijalos con attach(pin, min, max) para no empujarlo contra sus topes mecanicos. El jitter (temblor en reposo) se combate con buena alimentación, filtrando órdenes repetidas, cables de señal cortos y respetando los limites calibrados. Y nunca fuerces el eje con la mano mientras esta energizado: los engranes de plástico no perdonan.

En la práctica aprendiste el repertorio completo: barridos automáticos, posicionamiento con botones, control continuo con potenciómetro usando map(), movimientos suaves interpolados paso a paso, y el proyecto integrador: un radar con HC-SR04 que barre el ambiente y reporta distancias por angulo. Los ejercicios te llevaron mas lejos: puerta automática, termómetro de aguja, barrido de velocidad variable, dos servos sincronizados con interpolacion, y una cabeza robótica que detecta y sigue objetos cercanos con lógica de estados e histéresis.

Lo que sigue en tu camino: combina servos con todo lo que ya sabes. Un brazo robótico es varios servos coordinados con la técnica de interpolacion del ejercicio 4. Un robot móvil que esquiva obstáculos es el radar de esta unidad montado sobre ruedas. Y cuando necesites giros continuos con control de velocidad, volverás a los motores DC de unidades anteriores, ahora sabiendo exactamente cuando elegir cada uno: servo para posición, motor DC para rotación continua. Esa decisión, posición contra velocidad, es una de las mas importantes en el diseño de robots, y ya la dominas.

UNIDAD 8: COMPORTAMIENTOS AUTÓNOMOS

Hasta ahora, todo lo que has construido en este curso responde a tus órdenes. Apretaste un botón y el robot se movió. Moviste una palanca y el motor giro. Escribiste un número en el monitor serial y el servo se colocó en esa posición. Eso es control manual: tu piensas, tu decides y el robot obedece. En esta unidad vas a dar el paso más emocionante de todo el curso: vas a programar robots que deciden solos.

Un comportamiento autónomo es una acción que el robot ejecuta por su cuenta, sin que nadie lo esté controlando en ese momento. Tu le das el programa, lo pones en la pista y el robot empieza a tomar decisiones: lee sus sensores, interpreta lo que está pasando en el mundo real y mueve sus motores en consecuencia. Si se sale de la línea, se corrige. Si aparece un obstáculo en su camino, lo rodea. Nadie le dice qué hacer en cada momento; el programa le dice cómo decidir.

Piensa en la diferencia con un auto a control remoto. En el auto a control remoto, el humano es el cerebro: ve, piensa y manda órdenes con las manos. En un auto autónomo, el cerebro es un programa. Tu vas a ser el autor de ese programa. Esa es la magia de la robótica: no estás construyendo un juguete que se mueve, estás construyendo un agente que se comporta.

El concepto central de esta unidad es un ciclo que se repite una y otra vez, miles de veces por segundo: percibir, decidir y actuar. Primero el robot percibe: lee sus sensores y convierte el mundo físico en números que el programa puede entender. Un sensor infrarrojo dice "veo negro" o "veo blanco". Un sensor ultrasónico dice "el obstáculo está a 24 centímetros". Después el robot decide: con esos números, el programa aplica reglas y elige qué hacer. "Si el sensor izquierdo ve negro, entonces gira a la derecha". Finalmente el robot actúa: enciende los motores de la forma adecuada para ejecutar la decisión. Y luego el ciclo vuelve a empezar: percibe de nuevo, decide de nuevo, actúa de nuevo. Percibir, decidir, actuar. Ese es el latido de todo robot autónomo, desde una aspiradora doméstica hasta un explorador espacial.

Los dos robots que vas a construir en esta unidad son los clásicos de la robótica educativa, y no es casualidad: cada uno enseña una habilidad de decisión distinta. El seguidor de línea es el robot que sigue una pista negra pintada sobre el piso. Su habilidad es la corrección continua: nunca está perfectamente centrado, siempre se está corrigiendo un poco a la izquierda o un poco a la derecha. Te va a enseñar como el programa reacciona a cambios pequeños y constantes en el ambiente. El evasor de obstáculos es el robot que avanza por un cuarto lleno de objetos y nunca choca. Su habilidad es la respuesta a eventos: la mayor parte del tiempo avanza tranquilo, pero cuando el sensor detecta algo cerca, interrumpe lo que estaba haciendo y ejecuta una secuencia de escape. Te va a enseñar como el programa maneja situaciones que ocurren de repente.

Estos dos robots comparten la misma base física: un chasis con dos ruedas motrices, cada una con su motor de corriente directa, mas una rueda loca que sirve de apoyo. Ese chasis lo vas a conocer a fondo porque también lo vas a usar en las unidades siguientes. Los motores se controlan con el módulo L298N, que ya conoces de unidades anteriores: el Arduino no puede dar suficiente corriente para mover motores directamente, así que el L298N funciona como un interruptor de potencia que recibe señales pequeñas del Arduino y deja pasar la corriente grande de la batería hacia los motores. Para decidir como se mueve el chasis, el programa combina los dos motores: si los dos giran hacia adelante, el robot avanza; si los dos giran hacia atrás, retrocede; si uno gira y el otro se detiene, el robot gira hacia un lado.

Esta unidad cubre las clases 22, 23 y 24 del curso. La clase 22 se concentra en la lógica de decisión: como convertir la lectura de un sensor en una acción usando condicionales if y else, como organizar comportamientos mas complejos con máquinas de estados (una forma elegante de decir "el robot sabe en que situación está y actúa en consecuencia"), y como escribir funciones de movimiento reutilizables para no repetir código. La clase 23 es el seguidor de línea: la teoría de como detecta la línea, el montaje del chasis con los dos motores y el L298N, y el código completo con dos sensores infrarrojos y la version mejorada con tres. La clase 24 es el evasor de obstáculos: el sensor ultrasónico HC-SR04 apuntando al frente, la secuencia de escape paso a paso, y la version avanzada con un servo que mueve el sensor para "mirar" a los lados antes de decidir por donde ir.

Antes de empezar, asegúrate de tener claros algunos conceptos de unidades anteriores: que es una entrada digital (HIGH o LOW), como funciona un condicional if con sus ramas else, que es una función y como recibe parámetros, y como se conecta el módulo L298N. Si alguno de esos temas te suena borroso, repasa las unidades 4, 5 y 6 antes de continuar. Los comportamientos autónomos se apoyan en todo lo anterior; esta unidad es donde todas las piezas se juntan y el robot cobra vida.

Una advertencia honesta antes de entrar en materia: el robot autónomo perfecto no existe, y el tuyo tampoco lo será al primer intento. Se va a salir de la línea. Va a chocar contra la pata de la mesa. Va a girar para el lado equivocado y tu vas a pensar que el programa está mal. La mayoría de las veces el programa estará bien y el problema será físico: un sensor mal calibrado, una pila baja, una pista con reflejos, una velocidad demasiado alta. Depurar un robot autónomo es mitad programación y mitad detective físico. Esta unidad te enseña las dos mitades. Cuando termines, no solo vas a tener dos robots funcionando: vas a tener el criterio para diagnosticar por qué un robot se comporta mal y cómo arreglarlo. Esa es la habilidad que separa a quien copia código de quien hace robótica de verdad.

Vamos por partes. Primero la teoría de la decisión, luego el seguidor, luego el evasor, y al final la práctica donde tu pones las manos. Enciende tu Arduino y prepara la cinta de aislar negra: la vas a necesitar.

Explicación detallada

Lógica de decisión: del sensor a la acción

Todo comportamiento autónomo se reduce a una pregunta que el programa se hace muchas veces por segundo: "¿con lo que estoy viendo ahora, qué debo hacer?". La respuesta se escribe con condicionales, y el condicional más importante en robótica es el `if` con `else`. Un sensor te entrega un dato, el `if` evalúa una condición sobre ese dato, y según el resultado el programa ejecuta una acción u otra.

Imagina el caso más simple posible: un robot con un solo sensor infrarrojo abajo y la consigna de no salirse de una zona blanca. El sensor devuelve `LOW` cuando ve blanco y `HIGH` cuando ve negro. La lógica es: "si el sensor ve negro, entonces me estoy saliendo, así que giro; si no, sigo avanzando". En código se ve así:

```
int lectura = digitalRead(SENSOR);

if (lectura == HIGH) {
  girarDerecha();
} else {
  avanzar();
}
```

Eso es todo. Esa estructura tan pequeña ya es un comportamiento autónomo completo: el robot percibe (lee el sensor), decide (el `if`) y actúa (llama a la función de movimiento). El truco

esta en que el loop de Arduino repite este código cientos de veces por segundo, así que el robot reacciona casi al instante cuando algo cambia.

Cuando el robot tiene dos sensores, la lógica crece con if y else anidados o encadenados. Un seguidor de línea con sensor izquierdo y sensor derecho tiene cuatro combinaciones posibles: los dos ven blanco (voy bien, avanzó), el izquierdo ve negro (me estoy saliendo por la izquierda, corrijo a la derecha), el derecho ve negro (corrijo a la izquierda), los dos ven negro (llegue a un cruce o me perdí, decido algo especial). El código encadena las condiciones:

```
if (izq == NEGRO && der == NEGRO) {  
    // caso especial: cruce o fin de línea  
} else if (izq == NEGRO) {  
    // corregir hacia la derecha  
} else if (der == NEGRO) {  
    // corregir hacia la izquierda  
} else {  
    // todo en orden: avanzar  
}
```

El orden de los if importa muchísimo. El programa evalúa de arriba hacia abajo y ejecuta la primera condición que sea verdadera. Por eso el caso mas específico (los dos sensores en negro) va primero: si lo pusieras al final, nunca se ejecutaría porque los casos individuales lo atraparían antes. Esta es una regla de oro de la programación de comportamientos: primero los casos especiales, después los casos generales. Cuando tu robot haga algo raro, lo primero que debes revisar es el orden de tus condiciones.

Los condicionales anidados aparecen cuando una decisión depende de otra. En el evasor de obstáculos, primero preguntas "hay algo cerca?" y solo si la respuesta es si, preguntas la segunda cuestión: "ya mire a los lados, por donde hay mas espacio?". Eso se escribe con un if dentro de otro if:

```
if (distancia < UMBRAL) {  
    detenerse();  
    // ahora decido hacia donde girar  
    if (espacioIzquierda > espacioDerecha) {  
        girarIzquierda();  
    } else {  
        girarDerecha();  
    }  
} else {  
    avanzar();  
}
```

Fíjate en algo importante: las decisiones del robot no usan numeros mágicos sueltos por el código. En vez de escribir "if (distancia < 20)", defines al inicio "const int UMBRAL = 20;" y usas el nombre. Cuando tengas que ajustar el comportamiento en la pista real (y vas a tener que hacerlo), cambiar un solo valor al inicio del programa es rapidísimo; cazar veinte numeros regados por el código es una pesadilla. Los programadores de robots llaman a estos valores "parámetros de comportamiento" y calibrarlos es una parte normal del trabajo, no un fracaso.

Máquinas de estados: el robot sabe en que situación esta

Los if anidados funcionan bien para comportamientos simples, pero hay un punto donde el código se vuelve un espagueti: cuando la acción correcta depende no solo de lo que el sensor ve ahora, sino de lo que el robot estaba haciendo antes. El evasor de obstáculos es el ejemplo perfecto. Su secuencia de escape tiene pasos: primero se detiene, luego retrocede un poco, luego gira, y finalmente reanuda la marcha. Si intentas escribir eso con puros if anidados, el programa se confunde porque "hay un obstáculo cerca" es verdad durante toda la secuencia, y cada vuelta del loop volvería a empezar desde el primer paso.

La solución elegante se llama máquina de estados. La idea es sencilla: defines una variable que guarda "en que situación estoy" (el estado actual), y en cada vuelta del loop el programa mira el estado, ejecuta la acción correspondiente, y decide si debe cambiar a otro estado. Los estados del evasor serían: AVANZAR, DETENERSE, RETROCEDER, GIRAR. El robot siempre esta en exactamente uno de esos estados.

En código, una máquina de estados se escribe con un switch o con if encadenados sobre la variable de estado:

```

enum Estado { AVANZAR, DETENERSE, RETROCEDER, GIRAR };
Estado estadoActual = AVANZAR;

void loop() {
    int distancia = medirDistancia();

    switch (estadoActual) {
        case AVANZAR:
            avanzar();
            if (distancia < UMBRAL) {
                estadoActual = DETENERSE;
            }
            break;

        case DETENERSE:
            alto();
            estadoActual = RETROCEDER;
            tiempoInicio = millis();
            break;

        case RETROCEDER:
            atras();
            if (millis() - tiempoInicio > 500) {
                estadoActual = GIRAR;
                tiempoInicio = millis();
            }
            break;

        case GIRAR:
            girarDerecha();
            if (millis() - tiempoInicio > 700) {
                estadoActual = AVANZAR;
            }
            break;
    }
}

```

Observa dos detalles técnicos importantes. Primero, los tiempos se manejan con `millis()` en lugar de `delay()`. La función `delay()` congela todo el programa: durante un `delay` el robot no lee sensores, no decide nada, queda ciego. En un robot autónomo eso es peligroso y torpe. Con `millis()` guardas el momento en que empezó el estado y en cada vuelta preguntas cuanto tiempo paso; el loop sigue corriendo y el robot sigue percibiendo. Este es uno de los errores comunes mas graves de los principiantes: `delays` que ciegan al robot. Segundo, cada estado

tiene clara su acción y su condición de salida: que tiene que pasar para cambiar al siguiente estado. Esa claridad es la gran ventaja de la máquina de estados: puedes dibujar en papel los estados como cajitas y las transiciones como flechas, y el dibujo se traduce casi directo a código.

El seguidor de línea también se puede pensar como máquina de estados, aunque mas simple: CENTRADO, CORRIGIENDO_IZQ, CORRIGIENDO_DER. Pero con dos o tres sensores los if encadenados suelen bastar. La regla práctica es: si tu comportamiento tiene secuencia (paso 1, luego paso 2, luego paso 3), usa máquina de estados; si solo reacciona al momento presente, los if encadenados son suficientes.

Dibujar la máquina de estados en papel antes de programar no es tarea escolar aburrida: es la forma mas rápida de descubrir que te falta un caso. Cuando dibujas las cajitas te das cuenta de preguntas como "y si mientras retrocede aparece otro obstáculo atrás?" o "que pasa si la línea se acaba de golpe?". Esos casos olvidados son los que hacen que el robot se comporte raro en la pista. Diez minutos de dibujo te ahorran una hora de depuración.

El seguidor de línea: teoría y montaje mental

El seguidor de línea es el "hola mundo" de la robótica autonoma. El desafio: hay una línea negra pintada sobre una superficie blanca y el robot debe recorrerla de principio a fin sin salirse. Parece fácil hasta que ves a tu robot zigzagueando como borracho en su primer intento. Ese zigzag, sin embargo, es exactamente el comportamiento correcto: el robot nunca va perfectamente centrado, va corrigiendose todo el tiempo.

El sensor que hace posible esto es el TCRT5000, un módulo infrarrojo con un emisor y un receptor. El emisor lanza luz infrarroja hacia el piso y el receptor mide cuanta luz rebota. La superficie blanca refleja mucha luz infrarroja y la superficie negra absorbe casi toda. El módulo trae un comparador con un potenciómetro: tu giras un tornillito y ajustas el punto exacto donde el módulo decide "esto es blanco" o "esto es negro". La salida es digital: HIGH o LOW según como lo calibres. Lo habitual es calibrarlo para que entregue LOW sobre blanco y HIGH sobre negro, pero lo importante no es cual valor corresponde a cual, sino que sepas cual es cual en TU montaje y lo definas con constantes en el código.

La colocacion de los sensores es crítica. Van montados al frente del chasis, apuntando hacia abajo, a unos 5 a 10 milímetros del piso. Si quedan muy altos, la senal se debilita y el sensor no distingue bien; si quedan muy bajos, rozan con la pista. Con dos sensores, se colocan uno a cada lado de la línea: cuando el robot va centrado, los dos sensores quedan sobre la zona blanca, uno a la izquierda de la línea y otro a la derecha. En cuanto el robot se desvia un poco, uno de los sensores pisa la línea negra y el programa corrige girando hacia el lado contrario.

Espera, piensa esto con calma porque es el corazón del seguidor: si el sensor IZQUIERDO detecta negro, significa que la línea esta a la izquierda del robot, así que el robot debe girar a la IZQUIERDA para volver a centrarse. Muchos principiantes lo programan al revés y el robot se sale de la pista cada vez mas rápido. La regla mnemotecnica es: "sigue a la línea, no huyas de ella". El sensor que ve negro te esta diciendo donde esta la línea; gira hacia ese lado.

Esta lógica se llama control bang-bang porque la corrección es todo o nada: o avanzas recto o giras. No hay termino medio. El resultado es el zigzag característico: el robot avanza, se desvia, corrige bruscamente, se pasa al otro lado, corrige de nuevo. Funciona, y para pistas con curvas suaves es suficiente. Mas adelante en la sección de ejemplos verás la version con tres sensores, donde el sensor central detecta cuando vas perfectamente centrado y las correcciones son mas suaves.

El diseño de la pista también es parte del robot, aunque no lo parezca. La forma mas barata y efectiva es cinta de aislar negra sobre una superficie blanca: cartulina blanca grande, piso de ceramica clara o una lamina de unicel. La línea debe tener entre 1.5 y 2 centímetros de ancho si tus sensores están separados unos 3 centímetros entre si; la proporcion importa porque si la línea es mas ancha que la separación de los sensores, los dos sensores pueden quedar sobre negro al mismo tiempo y el robot no sabe hacia donde corregir. Evita las curvas muy cerradas en tu primera pista: un angulo de 90 grados es un reto avanzado, empieza con curvas amplias. Y un consejo de oro: construye la pista en el mismo lugar y sobre la misma superficie donde vas a hacer las pruebas finales, porque la calibración depende de la luz del ambiente y del material del piso.

La calibración sobre la pista real es un paso que no te puedes saltar. El procedimiento es: coloca el robot sobre la zona blanca de TU pista, gira el potenciómetro del módulo hasta que el LED indicador apenas se apague (o se encienda, según tu lógica), luego pasa el sensor sobre la línea negra y verifica que el LED cambie de estado. Repite con cada sensor. Hazlo con la iluminación que habra durante las pruebas: si calibras de noche con luz artificial y pruebas de día junto a la ventana, los valores cambian y el robot se vuelve loco. La luz solar directa es el enemigo número uno del TCRT5000 porque el sol emite muchísimo infrarrojo y satura el receptor. Si tu robot funciona perfecto en interiores y falla en el patio, ya sabes por que.

El evasor de obstáculos: teoría del escape

El evasor de obstáculos juega un juego distinto: en vez de seguir algo, debe evitar todo. Lleva el sensor ultrasónico HC-SR04 apuntando al frente, que mide la distancia al objeto mas cercano emitiendo un pulso de sonido de alta frecuencia y midiendo cuanto tarda el eco en regresar. Ya lo usaste en la unidad de sensores: sabes que mide de 2 centímetros a unos 4 metros, que su angulo de detección es de unos 15 grados y que las superficies blandas o inclinadas a veces no devuelven bien el eco.

El comportamiento básico usa un umbral de distancia, por ejemplo 20 centímetros. Mientras la distancia medida sea mayor al umbral, el robot avanza tranquilo. En el momento en que la distancia cae por debajo del umbral, el robot ejecuta su secuencia de escape: se detiene, retrocede un poco para ganar espacio, gira hacia un lado, y reanuda la marcha. Esa secuencia es la máquina de estados que viste antes: AVANZAR, DETENERSE, RETROCEDER, GIRAR. Cada estado dura un tiempo fijo que tu calibras probando: medio segundo de reversa y unos 700 milisegundos de giro suelen ser un buen punto de partida para un chasis pequeño.

La version básica tiene una limitación obvia: gira siempre hacia el mismo lado. Si el obstáculo esta justo en una esquina o si el robot entra a un callejon sin salida, girar siempre a la derecha puede dejarlo atrapado dando vueltas. La version avanzada resuelve esto con un servo explorador: montas el HC-SR04 sobre un pequeño servomotor que puede apuntar al frente, a la izquierda o a la derecha. Cuando detecta un obstáculo, el robot se detiene, el servo "mira" a la izquierda y mide la distancia, luego "mira" a la derecha y mide de nuevo, y el programa elige girar hacia el lado con mas espacio libre. Es un comportamiento notablemente mas inteligente y es el mismo principio que usan las aspiradoras robot comerciales cuando exploran una habitación: ante la duda, mide antes de moverte.

Un detalle físico importante: el HC-SR04 debe ir montado con cierta altura sobre el piso, unos 10 a 15 centímetros, para que no detecte el propio piso como obstáculo cuando el chasis se inclina. Y el servo explorador necesita pausas pequeñas después de cada movimiento: si mides la distancia mientras el servo aun se esta moviendo, la lectura sale mal. En el código verás un delay corto justo después de mover el servo; este es uno de los pocos delays aceptables porque ocurre una sola vez durante la decisión, no dentro del ciclo principal de movimiento.

Depuración de comportamientos: el método del detective

Cuando un robot autónomo se comporta mal, el instinto del principiante es cambiar el código a ciegas. No lo hagas. Existe un método ordenado y te va a ahorrar horas: prueba cada sensor por separado antes de integrar, y cuando algo falle, aísla el problema por capas.

La capa uno es la percepción: el robot esta viendo bien el mundo? Escribe un programa de prueba minuscuro que solo lea el sensor y lo imprima en el monitor serial, sin mover ningún motor. Para el TCRT5000, pasa la mano o un papel negro bajo el sensor y mira si el valor cambia de forma clara y estable. Para el HC-SR04, pon un libro a distancias conocidas (10, 20, 30 centímetros, medidos con regla) y verifica que las lecturas sean razonables y estables. Si las lecturas saltan como locas o no cambian, el problema es el sensor, su calibración o su conexión, y ningún cambio en la lógica lo va a arreglar. Esta prueba te toma dos minutos y descarta la mitad de las causas posibles.

La capa dos es la decisión: con lecturas buenas, el programa elige bien? Agrega impresiones seriales en los puntos de decisión: "sensor izq = negro, decido girar izquierda". Observa el monitor mientras mueves el robot con la mano sobre la pista. Si la decisión impresa es correcta pero el robot hace otra cosa, el problema esta en la capa tres. Si la decisión impresa ya es incorrecta, revisa tus condiciones y su orden.

La capa tres es la acción: los motores hacen lo que el programa ordena? Desconecta la lógica de sensores y prueba cada función de movimiento por separado: llama a avanzar() y verifica que las dos ruedas giren hacia adelante; llama a girarIzquierda() y verifica el giro real. El error clásico aquí es tener un motor conectado al revés: el programa dice "avanza" pero una rueda va hacia atrás y el robot gira en círculos. Si eso pasa, no cambies el código: invierte los cables de ese motor en el L298N o intercambia los pines en el código, pero decide una sola corrección y aplicala.

Errores comunes y buenas prácticas

Calibrar en una superficie y probar en otra es el error número uno del seguidor de línea. Calibras los sensores sobre tu escritorio blanco y luego pruebas sobre el piso de la cocina que es beige: los umbrales ya no sirven. Calibra siempre sobre la pista real, con la luz real.

Los delays que ciegan al robot son el error número dos. Un delay(1000) en medio del loop significa un segundo entero sin leer sensores. A velocidad normal, en un segundo el robot avanza 20 o 30 centímetros: suficiente para salirse de la pista o chocar. Usa millis() para los tiempos, como viste en la máquina de estados. Reserva delay() solo para pausas puntuales donde el robot ya esta detenido, como esperar a que el servo explorador se acomode.

La velocidad muy alta es el error número tres. Un seguidor que funciona perfecto a velocidad media se vuelve incontrolable al máximo porque las correcciones llegan tarde: cuando el programa ordena girar, el robot ya se pasó de la línea. Empieza siempre con velocidad moderada (un valor PWM de 150 sobre 255 es buen punto de partida) y sube la velocidad solo cuando el comportamiento sea estable. Rápido y mal no sirve; lento y bien es la base para luego ir rápido y bien.

Las baterías bajas son el enemigo silencioso. Cuando el voltaje cae, los motores pierden fuerza y giran disparejo: un motor responde y el otro no, y el robot se va chueco aunque el programa este perfecto. Si tu robot funcionaba ayer y hoy se comporta raro sin que cambiaras nada, mide o cambia las baterías antes de tocar el código.

Como buenas prácticas: define todos los pines y umbrales como constantes con nombres claros al inicio del programa; escribe funciones de movimiento con nombres que digan lo que hacen (adelante(), atrás(), izquierda(), derecha(), alto()); prueba cada función y cada sensor por separado antes de juntar todo; calibra sobre la pista real con la luz real; y lleva un cuaderno donde anotes que cambiaste y que paso. La robótica autónoma es experimental: el que anota sus experimentos aprende el doble de rápido.

Ejemplos paso a paso

Vamos a construir de verdad. Esta sección es un taller guiado: primero la lista de materiales y las conexiones, después el código por partes, cada ejemplo completo y comentado, listo para compilar en tu Arduino Uno. Te recomiendo seguir el orden: primero haces funcionar las funciones de movimiento, luego el seguidor con dos sensores, luego lo mejoras a tres, y al final el evasor en sus dos versiones. Cada ejemplo se apoya en el anterior.

Lista de materiales del chasis (la base común de los dos robots):

CHASIS BASE (seguidor de línea y evasor de obstáculos):

- 1x Arduino Uno (o compatible)
- 1x Chasis de acrílico o madera para robot (tipo "carrito" 2 ruedas)
- 2x Motores DC con reductor (motores amarillos TT de 3-6V, los más comunes)
- 2x Ruedas para motor TT (diámetro aprox. 65 mm)
- 1x Rueda loca (rueda libre giratoria, tipo "caster")
- 1x Módulo puente H L298N (controla los 2 motores)
- 1x Portapilas para 6 pilas AA (9V aprox.) o batería LiPo 2S con su cargador
- 1x Protoboard pequeña o placa perforada (opcional, para ordenar conexiones)
- Cables Dupont macho-hembra y macho-macho (paquete surtido)
- Cinta de aislar negra (para la pista del seguidor)
- Cartulina blanca grande o superficie clara (base de la pista)

SOLO SEGUIDOR DE LÍNEA:

- 2x o 3x Módulo sensor infrarrojo TCRT5000 (uno por sensor)
- Tornillería y separadores para montar los sensores al frente

SOLO EVASOR DE OBSTÁCULOS:

- 1x Sensor ultrasonico HC-SR04
- 1x Servomotor SG90 (solo para la versión con explorador)
- 1x Soporte para montar el HC-SR04 sobre el servo (se puede improvisar con cartón)

Conexiones del chasis base con el L298N. El L298N tiene dos lados: el lado de potencia (donde entra la batería y salen los cables a los motores) y el lado de lógica (pines que van al Arduino). Conecta así: la batería (positivo y negativo) a los bornes de 12V y GND del L298N; el GND del L298N también al GND del Arduino (tierra común, sin esto nada funciona bien). Los dos cables del motor izquierdo a OUT1 y OUT2; los del motor derecho a OUT3 y OUT4. Del lado lógico: IN1 al pin 8 del Arduino, IN2 al pin 9, IN3 al pin 10, IN4 al pin 11, ENA al pin 5 y ENB al pin 6. Los pines ENA y ENB son los que reciben señal PWM para controlar la velocidad: por eso van a pines con virgüllita (5 y 6). IMPORTANTE: el L298N trae unos jumpers pequeños sobre ENA y ENB; si los dejas puestos, los motores van siempre a máxima velocidad; quítalos para controlar la velocidad por PWM desde el Arduino. El Arduino se alimenta por separado con su cable USB o una pila de 9V a su jack; la batería de los motores NO debe alimentar al Arduino por el pin de 5V.

Para el seguidor de línea, cada TCRT5000 tiene 3 pines: VCC (5V), GND y OUT (señal digital). Conecta el OUT del sensor izquierdo al pin 2 del Arduino y el del derecho al pin 3. Si usas tercer sensor central, su OUT va al pin 4. Monta los sensores en una pequeña regleta al frente del chasis, separados unos 3 centímetros entre sí, apuntando hacia abajo a unos 8 milímetros del piso.

Para el evasor, el HC-SR04 tiene 4 pines: VCC (5V), TRIG al pin 12, ECHO al pin 13 y GND. Montalo al frente del chasis, apuntando horizontalmente hacia adelante, a unos 12 centímetros del piso. En la version con explorador, el servo SG90 va al pin 7 (cable de senal naranja o amarillo; rojo a 5V y marron a GND) y el HC-SR04 se monta encima del brazo del servo.

(a) Funciones de movimiento para chasis de 2 motores

Antes de cualquier comportamiento autónomo, necesitas un "vocabulario" de movimientos confiable. Estas cinco funciones son la base de todo lo que sigue: adelante, atrás, izquierda, derecha y alto. La velocidad se controla con PWM en ENA y ENB (valores de 0 a 255). Guarda este código: lo vas a reutilizar en todos los ejemplos copiando el bloque completo.

```
// =====
// FUNCIONES DE MOVIMIENTO - Chasis 2 motores + L298N
// Pines: IN1=8, IN2=9, IN3=10, IN4=11, ENA=5, ENB=6
// =====

const int IN1 = 8;
const int IN2 = 9;
const int IN3 = 10;
const int IN4 = 11;
const int ENA = 5;    // PWM velocidad motor izquierdo
const int ENB = 6;    // PWM velocidad motor derecho

int velocidadBase = 150; // 0 a 255, punto de partida moderado

void setup() {
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);
}

// Motor izquierdo: true = adelante, false = atras
void motorIzq(boolean dir, int vel) {
  digitalWrite(IN1, dir ? HIGH : LOW);
  digitalWrite(IN2, dir ? LOW : HIGH);
  analogWrite(ENA, vel);
}

// Motor derecho: true = adelante, false = atras
void motorDer(boolean dir, int vel) {
```

```

    digitalWrite(IN3, dir ? HIGH : LOW);
    digitalWrite(IN4, dir ? LOW : HIGH);
    analogWrite(ENB, vel);
}

void adelante() {
    motorIzq(true, velocidadBase);
    motorDer(true, velocidadBase);
}

void atras() {
    motorIzq(false, velocidadBase);
    motorDer(false, velocidadBase);
}

void izquierda() {
    // giro en el lugar: izquierda atras, derecha adelante
    motorIzq(false, velocidadBase);
    motorDer(true, velocidadBase);
}

void derecha() {
    motorIzq(true, velocidadBase);
    motorDer(false, velocidadBase);
}

void alto() {
    analogWrite(ENA, 0);
    analogWrite(ENB, 0);
}

```

Como probar este bloque: sube un programa que en el setup configure los pines y en el loop llame a cada función dos segundos con alto() entre ellas. Observa las ruedas. Si al llamar adelante() una rueda gira al revés, invierte los dos cables de ese motor en el L298N (con la batería desconectada). No sigas hasta que las cinco funciones hagan exactamente lo que dicen sus nombres. Este paso parece lento pero es el cimiento: si el vocabulario de movimiento esta mal, ningún comportamiento autónomo va a funcionar.

Nota sobre los giros: izquierda() y derecha() hacen girar al robot sobre su propio eje (una rueda adelante y otra atrás). Es el giro mas cerrado posible y es ideal para correcciones rápidas en el seguidor y para la secuencia de escape del evasor. Existe otra forma de girar mas suave (una rueda avanza y la otra se detiene), que verás como variante en los ejercicios.

(b) Seguidor de línea con 2 sensores IR (código completo)

Este es el programa completo del seguidor con dos TCRT5000. Incluye el bloque de funciones de movimiento de la parte (a): cópialo tal cual al inicio de tu sketch o, mejor aun, guarda las funciones en una pestaña separada del IDE de Arduino. La lógica es control bang-bang puro: lee los dos sensores, decide con if encadenados y actúa. Calibra los sensores sobre tu pista antes de subir el código.

```
// =====
// SEGUIDOR DE LINEA - 2 sensores TCRT5000
// Sensores: IZQ pin 2, DER pin 3
// Motores: ver bloque de funciones de movimiento (a)
// =====

const int SEN_IZQ = 2;
const int SEN_DER = 3;

// Ajusta segun tu calibracion:
// LOW = blanco, HIGH = negro (lo habitual con el potenciómetro)
const int NEGRO = HIGH;
const int BLANCO = LOW;

void setup() {
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);
  pinMode(SEN_IZQ, INPUT);
  pinMode(SEN_DER, INPUT);
  Serial.begin(9600); // util para depurar: muestra que ven los sensores
}

void loop() {
  int izq = digitalRead(SEN_IZQ);
  int der = digitalRead(SEN_DER);

  // Depuracion: descomenta para ver lecturas en el monitor serial
  // Serial.print(izq); Serial.print(" "); Serial.println(der);

  if (izq == NEGRO && der == NEGRO) {
    // Los dos ven negro: cruce, linea gruesa o me perdi.
    // Decision segura: avanzar despacio para buscar la linea.
    motorIzq(true, 100);
  }
}
```

```

    motorDer(true, 100);
} else if (izq == NEGRO) {
    // La linea esta a la izquierda: giro hacia la linea.
    izquierda();
} else if (der == NEGRO) {
    // La linea esta a la derecha: giro hacia la linea.
    derecha();
} else {
    // Los dos ven blanco: voy centrado, avanzo.
    adelante();
}
}

// --- Pega aqui el bloque de funciones de movimiento (a) ---
// (motorIzq, motorDer, adelante, atras, izquierda, derecha, alto)

```

Explicación por partes. En el setup se configuran los pines de los motores como salidas y los de los sensores como entradas; el Serial a 9600 es tu ventana de depuración. En el loop, lo primero es percibir: se leen los dos sensores y se guardan en variables. Después viene decidir: la cadena de if evalúa los cuatro casos en orden de especificidad. El caso de los dos sensores en negro va primero porque es el mas raro y necesita trato especial: si lo pusieras al final, jamás se ejecutaría. La decisión de ese caso es avanzar despacio, que es una estrategia razonable cuando el robot se pierde: sigue de frente hasta que un sensor vuelva a encontrar la línea. Finalmente actuar: cada rama llama a una función de movimiento.

Para calibrar, coloca el robot con los sensores sobre la zona blanca de la pista y gira el potenciómetro de cada módulo hasta que su LED indicador cambie justo en el límite; luego verifica pasando cada sensor sobre la línea negra: el LED debe cambiar de estado con claridad. Hazlo con la luz que habra en tus pruebas. Sube el código, coloca el robot al inicio de la pista y observalo. Si zigzaguea pero avanza, funciona: el zigzag es normal en bang-bang. Si se sale siempre hacia el mismo lado, probablemente tienes invertida la lógica de giro (cambia izquierda() por derecha() en las ramas y prueba de nuevo). Si no reacciona a la línea, los sensores están mal calibrados o muy altos.

(c) Seguidor de línea con 3 sensores (mas suave)

Agregar un tercer sensor al centro cambia el juego: ahora el robot sabe cuando va perfectamente centrado y solo corrige cuando de verdad se desvia. El sensor central va justo sobre la línea cuando el robot va bien; los laterales son los vigilantes que detectan las desviaciones. El resultado es un movimiento mucho mas suave y la capacidad de tomar curvas mas cerradas a mayor velocidad.

```

// =====
// SEGUIDOR DE LINEA - 3 sensores TCRT5000
// Sensores: IZQ pin 2, CEN pin 4, DER pin 3
// =====

const int SEN_IZQ = 2;
const int SEN_CEN = 4;
const int SEN_DER = 3;

const int NEGRO = HIGH;
const int BLANCO = LOW;

void setup() {
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);
  pinMode(SEN_IZQ, INPUT);
  pinMode(SEN_CEN, INPUT);
  pinMode(SEN_DER, INPUT);
}

void loop() {
  int izq = digitalRead(SEN_IZQ);
  int cen = digitalRead(SEN_CEN);
  int der = digitalRead(SEN_DER);

  if (cen == NEGRO && izq == BLANCO && der == BLANCO) {
    // Perfectamente centrado: a toda velocidad base.
    adelante();
  } else if (izq == NEGRO) {
    // Desviado a la derecha, la linea quedo a la izquierda.
    // Correccion suave: freno la rueda izquierda, la derecha empuja.
    motorIzq(true, 80);
    motorDer(true, velocidadBase);
  } else if (der == NEGRO) {
    // Desviado a la izquierda, la linea quedo a la derecha.
    motorIzq(true, velocidadBase);
    motorDer(true, 80);
  } else if (izq == NEGRO && der == NEGRO) {
    // Caso raro: cruce o linea perdida. Avanzo despacio.
    motorIzq(true, 100);
    motorDer(true, 100);
  }
}

```

```

    } else {
        // Ningun sensor ve la linea: me perdi. Giro en el lugar
        // para buscarla (ultima direccion conocida seria lo ideal).
        izquierda();
    }
}

// --- Pega aqui el bloque de funciones de movimiento (a) ---

```

La diferencia clave con la version de dos sensores esta en las correcciones: en vez de girar sobre el propio eje (brusco), se frena solo la rueda del lado hacia donde hay que girar mientras la otra sigue empujando. El robot describe una curva suave en lugar de un giro seco. Fíjate también en el último else: si ningún sensor ve negro, el robot esta perdido y lo mas sensato es girar en el lugar para reencontrar la línea. Es un comportamiento de recuperación sencillo pero efectivo.

Un detalle de montaje: con tres sensores la separación entre ellos debe ser menor que el ancho de la línea, para que el sensor central siempre pueda estar sobre la línea mientras los laterales quedan en blanco. Con línea de 2 centímetros, separa los sensores 1.5 centímetros entre si. Si la línea es mas angosta que la separación, el sensor central nunca la verá bien y el robot se comportara peor que con dos sensores.

(d) Evasor de obstáculos básico con ultrasónico

Este programa implementa la máquina de estados completa: AVANZAR, DETENERSE, RETROCEDER y GIRAR. Usa millis() para los tiempos, sin delays que cieguen al robot. El HC-SR04 va al frente: TRIG al pin 12, ECHO al pin 13. El umbral de 20 centímetros es un buen punto de partida para interiores; ajústalo según tu espacio de pruebas.

```

// =====
// EVASOR DE OBSTACULOS - HC-SR04 al frente
// TRIG pin 12, ECHO pin 13
// Maquina de estados con millis() (sin delay)
// =====

const int TRIG = 12;
const int ECHO = 13;

const int UMBRAL_CM = 20;           // distancia de reaccion
const long T_REVERSA = 500;         // ms retrocediendo
const long T_GIRO = 700;            // ms girando

enum Estado { AVANZAR, DETENERSE, RETROCEDER, GIRAR };

```

```

Estado estadoActual = AVANZAR;
unsigned long tiempoInicio = 0;

void setup() {
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);
    pinMode(ENA, OUTPUT);
    pinMode(ENB, OUTPUT);
    pinMode(TRIG, OUTPUT);
    pinMode(ECHO, INPUT);
    Serial.begin(9600);
}

// Devuelve la distancia en centimetros al objeto mas cercano
int medirDistancia() {
    digitalWrite(TRIG, LOW);
    delayMicroseconds(2);
    digitalWrite(TRIG, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG, LOW);
    long duracion = pulseIn(ECHO, HIGH, 30000); // timeout 30 ms
    int cm = duracion / 58;
    if (cm == 0) cm = 999; // sin eco = muy lejos
    return cm;
}

void loop() {
    int distancia = medirDistancia();

    switch (estadoActual) {
        case AVANZAR:
            adelante();
            if (distancia < UMBRAL_CM) {
                estadoActual = DETENERSE;
            }
            break;

        case DETENERSE:
            alto();
            // Pausa breve para estabilizar antes de la reversa
            estadoActual = RETROCEDER;
            tiempoInicio = millis();
            break;
    }
}

```

```

    case RETROCEDER:
        atras();
        if (millis() - tiempoInicio > T_REVERSA) {
            estadoActual = GIRAR;
            tiempoInicio = millis();
        }
        break;

    case GIRAR:
        derecha();
        if (millis() - tiempoInicio > T_GIRO) {
            estadoActual = AVANZAR;
        }
        break;
}
}

// --- Pega aqui el bloque de funciones de movimiento (a) ---

```

Explicación por partes. La función `medirDistancia()` implementa el protocolo del HC-SR04: pulso de 10 microsegundos en TRIG, mide con `pulseIn()` cuanto tarda el eco en ECHO y convierte a centímetros dividiendo entre 58 (el sonido viaja 58 microsegundos por centímetro, ida y vuelta). El timeout de 30000 microsegundos en `pulseIn` evita que el programa se quede colgado si no hay eco; si no hay eco, se devuelve 999 para que el robot lo interprete como "camino libre". En el loop, el switch ejecuta el estado actual: AVANZAR mueve el robot y vigila la distancia; en cuanto algo se acerca mas del umbral, cambia a DETENERSE. DETENERSE frena y pasa de inmediato a RETROCEDER guardando el tiempo de inicio. RETROCEDER va hacia atrás hasta que pasan T_REVERSA milisegundos y entonces cambia a GIRAR, también con su tiempo. GIRAR rota sobre su eje hasta completar T_GIRO y vuelve a AVANZAR. Como todo usa `millis()`, el sensor se sigue leyendo en cada vuelta del loop: el robot nunca queda ciego.

Para calibrar los tiempos, prueba en un pasillo con una caja como obstáculo. Si el robot choca antes de girar, sube UMBRAL_CM o baja velocidadBase. Si gira muy poco y vuelve a toparse con el mismo obstáculo, sube T_GIRO. Si retrocede demasiado y se sale de tu zona de pruebas, baja T_REVERSA. Anota cada cambio: es experimentacion, no adivinanza.

(e) Evasor con servo explorador (mira antes de girar)

La version avanzada monta el HC-SR04 sobre un servomotor SG90. Ante un obstáculo, el robot se detiene, el servo apunta a la izquierda y mide, apunta a la derecha y mide, y gira hacia donde haya mas espacio. Necesitas la librería Servo.h, que viene incluida con el IDE de Arduino.

```
// =====
// EVASOR CON SERVO EXPLORADOR - HC-SR04 sobre SG90
// Servo pin 7. TRIG pin 12, ECHO pin 13.
// =====
#include <Servo.h>

const int TRIG = 12;
const int ECHO = 13;
const int PIN_SERVO = 7;

const int UMBRAL_CM = 20;
const long T_REVERSA = 400;
const long T_GIRO = 600;

const int ANG_FRENTE = 90;
const int ANG_IZQ = 160;
const int ANG_DER = 20;

Servo explorador;

enum Estado { AVANZAR, DETENERSE, RETROCEDER, EXPLORAR, GIRAR };
Estado estadoActual = AVANZAR;
unsigned long tiempoInicio = 0;
boolean girarAIzquierda = false;

void setup() {
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);
    pinMode(ENA, OUTPUT);
    pinMode(ENB, OUTPUT);
    pinMode(TRIG, OUTPUT);
    pinMode(ECHO, INPUT);
    explorador.attach(PIN_SERVO);
    explorador.write(ANG_FRENTE);
    Serial.begin(9600);
}
```

```

int medirDistancia() {
    digitalWrite(TRIG, LOW);
    delayMicroseconds(2);
    digitalWrite(TRIG, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG, LOW);
    long duracion = pulseIn(ECHO, HIGH, 30000);
    int cm = duracion / 58;
    if (cm == 0) cm = 999;
    return cm;
}

// Mueve el servo a un angulo, espera a que se acomode y mide
int mirarHacia(int angulo) {
    explorador.write(angulo);
    delay(400); // pausa aceptable: el robot esta detenido
    return medirDistancia();
}

void loop() {
    int distancia = medirDistancia();

    switch (estadoActual) {
        case AVANZAR:
            explorador.write(ANG_FRENTE);
            adelante();
            if (distancia < UMBRAL_CM) {
                estadoActual = DETENERSE;
            }
            break;

        case DETENERSE:
            alto();
            estadoActual = RETROCEDER;
            tiempoInicio = millis();
            break;

        case RETROCEDER:
            atras();
            if (millis() - tiempoInicio > T_REVERSA) {
                estadoActual = EXPLORAR;
            }
            break;
    }
}

```



```

case EXPLORAR: {
    // Mide a ambos lados y elige el de mas espacio
    int distIzq = mirarHacia(ANG_IZQ);
    int distDer = mirarHacia(ANG_DER);
    explorador.write(ANG_FRENTE);
    Serial.print("Izq: "); Serial.print(distIzq);
    Serial.print(" Der: "); Serial.println(distDer);
    girarAIzquierda = (distIzq > distDer);
    estadoActual = GIRAR;
    tiempoInicio = millis();
    break;
}

case GIRAR:
    if (girarAIzquierda) {
        izquierda();
    } else {
        derecha();
    }
    if (millis() - tiempoInicio > T_GIRO) {
        estadoActual = AVANZAR;
    }
    break;
}
}

// --- Pega aqui el bloque de funciones de movimiento (a) ---

```

La novedad es el estado EXPLORAR y la función mirarHacia(). Cuando el robot termina de retroceder, en vez de girar a ciegas siempre al mismo lado, el servo apunta a la izquierda, espera 400 milisegundos a que el servo se estabilice y mide la distancia; repite hacia la derecha; compara y guarda la decisión en la variable girarAIzquierda. El estado GIRAR usa esa variable para elegir el sentido. El delay(400) dentro de mirarHacia() es aceptable porque el robot esta completamente detenido en ese momento: no hay nada que percibir con urgencia mientras explora.

Un truco de montaje: el HC-SR04 sobre el servo debe quedar firme pero ligero; si el sensor baila, las mediciones salen mal. Un pedazo de cartón grueso o una pequeña pieza impresa en 3D sirven como soporte. Verifica que el servo pueda girar libremente sin que los cables se atoren: los cables del HC-SR04 deben tener holgura suficiente para el giro de 140 grados.

Con esto tienes los cinco programas de la unidad. Todos comparten el mismo bloque de movimiento y la misma filosofía: percibir, decidir, actuar. En la siguiente sección, la práctica, vas a modificarlos, combinarlos y ponerlos a prueba contra reloj.

Práctica

La teoría sin pista no sirve de nada. Estos cinco ejercicios te obligan a ensuciarte las manos: vas a calibrar, medir tiempos, romper cosas y arreglarlas. Hazlos en orden, porque cada uno usa lo aprendido en el anterior. Para cada ejercicio hay enunciado, objetivo y solución completa con código explicado. Intenta resolverlo tu primero antes de leer la solución: la solución esta para verificar, no para copiar a ciegas.

Ejercicio 1: encuentra la velocidad máxima estable del seguidor. Enunciado: usando el seguidor de línea con 2 sensores, prueba valores de velocidadBase de 100, 150, 200 y 255 sobre tu pista. Para cada valor, corre el robot 3 veces y anota en cuantas se salió de la línea. Objetivo: entender la relación entre velocidad y estabilidad, y aprender a calibrar un parámetro de comportamiento con datos en vez de corazonadas. Pista: a mayor velocidad, las correcciones llegan tarde. Busca el valor mas alto con el que el robot complete las 3 vueltas sin salirse.

Solución del ejercicio 1

No hay un código nuevo que escribir: la solución es metodologica. Cambia la línea "int velocidadBase = 150;" por cada valor a probar, sube el código y corre 3 vueltas por valor. Una tabla como esta es tu resultado:

```
Velocidad 100: 3/3 vueltas completas (lento pero seguro)
Velocidad 150: 3/3 vueltas completas (equilibrio)
Velocidad 200: 2/3 vueltas completas (se sale en la curva cerrada)
Velocidad 255: 0/3 vueltas completas (incontrolable)
```

La conclusión típica: existe una velocidad máxima estable para tu pista y tu robot, y esta por debajo del máximo del hardware. En robótica competitiva esto se llama "sintonizar" y es la diferencia entre un robot rápido que termina la carrera y uno rapidísimo que nunca la termina. Guarda tu valor ganador como constante VELOCIDAD_OPTIMA y úsalo en los siguientes ejercicios. Si cambias de pista o de baterías, repite la prueba: la velocidad optima depende del mundo real, no solo del código.

Ejercicio 2: pista con curvas cerradas. Enunciado: diseña una pista que incluya al menos dos curvas de 90 grados. El seguidor bang-bang de 2 sensores probablemente se saldrá en esas curvas. Modifica el programa para que las tome sin salirse. Objetivo: aprender que los comportamientos deben adaptarse a la dificultad del ambiente, y practicar la estrategia de "frenar antes de la curva". Pista: cuando un solo sensor ve negro de forma repentina y total (no un roce, sino que pisa de lleno), el robot entró a una curva cerrada. La respuesta es girar mas agresivo y bajar la velocidad general.

Solución del ejercicio 2

La estrategia es doble: bajar la velocidad base y hacer los giros mas agresivos usando giro sobre el propio eje (que ya es lo que hace izquierda() y derecha()), pero además detectar la curva cerrada por el tiempo que el sensor lleva viendo negro. Una forma simple y efectiva: si un sensor ve negro, girar sobre el eje; eso ya lo hace el código base. El problema real de las curvas de 90 grados es la velocidad de entrada: el robot llega tan rápido que cuando el sensor detecta la curva, el chasis ya esta fuera. La solución es reducir velocidadBase a 110-120 y, como mejora, hacer que el robot avance mas lento justo después de una corrección fuerte. Este código agrega una variable de "precaucion" que baja la velocidad durante un momento después de cada giro:

```

// Variables globales nuevas:
int velocidadBase = 120;      // mas lento para curvas cerradas
unsigned long finPrecaucion = 0;

void loop() {
    int izq = digitalRead(SEN_IZQ);
    int der = digitalRead(SEN_DER);

    // Velocidad efectiva: reducida justo despues de corregir
    int vel = (millis() < finPrecaucion) ? 90 : velocidadBase;

    if (izq == NEGRO && der == NEGRO) {
        motorIzq(true, 90);
        motorDer(true, 90);
    } else if (izq == NEGRO) {
        // Giro sobre el eje: correccion maxima para curva cerrada
        motorIzq(false, vel);
        motorDer(true, vel);
        finPrecaucion = millis() + 300; // 300 ms de precaucion
    } else if (der == NEGRO) {
        motorIzq(true, vel);
        motorDer(false, vel);
        finPrecaucion = millis() + 300;
    } else {
        motorIzq(true, vel);
        motorDer(true, vel);
    }
}
}

```

La idea: después de cada corrección fuerte, el robot avanza despacio 300 milisegundos, como un conductor que frena al salir de una curva cerrada. Con esto, curvas de 90 grados que antes eran imposibles se vuelven tomables. Pruébalo y ajusta los 300 milisegundos según tu pista.

Ejercicio 3: evasor con dos umbrales. Enunciado: modifica el evasor básico para que tenga dos distancias de reacción: un umbral de precaucion (por ejemplo 40 cm) donde el robot baja la velocidad, y un umbral de emergencia (20 cm) donde ejecuta la secuencia de escape completa. Objetivo: practicar comportamientos graduados en vez de todo o nada, y ver como varios umbrales hacen al robot mas elegante. Pista: necesitas un estado o una lógica que distinga "lejos", "cerca" y "muy cerca". Puedes hacerlo dentro del estado AVANZAR sin agregar estados nuevos.

Solución del ejercicio 3

La solución modifica solo el estado AVANZAR de la máquina de estados: según la distancia, elige entre tres velocidades o dispara el escape. Así el robot desacelera al acercarse a un obstáculo en vez de ir a toda velocidad hasta el último momento, lo que reduce choques por inercia y se ve mucho mas "inteligente".

```
const int UMBRAL_PRECAUCION = 40;
const int UMBRAL_EMERGENCIA = 20;

// Dentro del loop, reemplaza el case AVANZAR por:
case AVANZAR:
    if (distancia < UMBRAL_EMERGENCIA) {
        estadoActual = DETENERSE;
    } else if (distancia < UMBRAL_PRECAUCION) {
        // Zona de precaucion: avanza a media velocidad
        motorIzq(true, 90);
        motorDer(true, 90);
    } else {
        // Camino libre: velocidad normal
        adelante();
    }
    break;
```

Observa que la máquina de estados no creció: sigue teniendo los mismos cuatro estados, solo que AVANZAR ahora tiene tres sub-comportamientos. Esta es una técnica muy usada en robótica real: comportamientos por capas, donde cada capa se activa en su rango de distancia. Como extensión, puedes agregar un tercer umbral "lejano" (60 cm) donde el robot apenas reduce un poco la velocidad. Cada umbral que agregues es un parámetro mas para calibrar en tus pruebas.

Ejercicio 4: combina línea y obstáculo. Enunciado: coloca un obstáculo (una caja) sobre la línea de la pista. Modifica el seguidor de línea para que, al detectar el obstáculo a menos de 15 cm, se detenga, lo rodee (retrocede, gira, avanza rodeandolo) y luego busque la línea para seguir su camino. Objetivo: integrar dos comportamientos en un solo programa y enfrentar el problema clásico de la robótica: que hacer cuando dos objetivos compiten (seguir la línea vs. no chocar). Pista: la seguridad manda: evitar el choque tiene prioridad sobre seguir la línea. Usa una máquina de estados con dos modos: SIGUIENDO y EVADIENDO.

Solución del ejercicio 4

La clave arquitectónica es una máquina de estados de alto nivel con dos modos. En modo SIGUIENDO corre la lógica del seguidor; si el ultrasónico detecta algo a menos de 15 cm, cambia a modo EVADIENDO, que ejecuta una secuencia fija de rodeo con tiempos; al terminar, vuelve a SIGUIENDO y el robot busca la línea. El rodeo es "a ciegas" (tiempos fijos) porque es lo más simple que funciona; en la vida real se usaría el servo explorador.

```

enum Modo { SIGUIENDO, EVADIENDO };
Modo modo = SIGUIENDO;
unsigned long t0 = 0;
int pasoEvasion = 0;

void loop() {
    int distancia = medirDistancia();

    if (modo == SIGUIENDO) {
        if (distancia < 15) {
            // Obstaculo sobre la linea: cambio a modo evasion
            modo = EVADIENDO;
            pasoEvasion = 0;
            t0 = millis();
            alto();
        } else {
            seguidorLinea(); // la logica del ejemplo (b) o (c)
        }
    } else {
        // Secuencia de rodeo por tiempos: retrocede, gira izq,
        // avanza, gira der, avanza. Cada paso dura lo calibrado.
        switch (pasoEvasion) {
            case 0: atras(); break;
            case 1: izquierda(); break;
            case 2: adelante(); break;
            case 3: derecha(); break;
            case 4: adelante(); break;
        }
        if (millis() - t0 > 600) { // 600 ms por paso, ajustar
            t0 = millis();
            pasoEvasion++;
            if (pasoEvasion > 4) {
                modo = SIGUIENDO; // vuelvo a buscar la linea
            }
        }
    }
}

```

La función `seguidorLinea()` contiene el código del ejemplo (b) o (c), y `medirDistancia()` la del ejemplo (d). Los 600 milisegundos por paso son un punto de partida: calíbralos para que el rodeo deje al robot mas o menos paralelo a la línea, del otro lado del obstáculo. No será perfecto: al volver a modo `SIGUIENDO`, el robot probablemente este descentrado y tendrá que buscar la línea girando un poco. Eso esta bien: diseñaste un comportamiento de recuperación

sin darte cuenta. Este ejercicio es una miniatura de como se programan los robots reales, que siempre están combinando objetivos en competencia con prioridades claras.

Ejercicio 5: carrera contra reloj. Enunciado: marca una pista cerrada (un ovalo) y mide con cronometro cuanto tarda tu seguidor en dar 3 vueltas completas. Haz 3 intentos, calcula el promedio, y luego intenta bajar ese tiempo modificando lo que quieras (velocidad, lógica de giro, número de sensores) sin que el robot se salga. Documenta cada cambio y su efecto en el tiempo. Objetivo: experimentar el ciclo completo de mejora de un robot autónomo: medir, hipotetizar, modificar, volver a medir. Es el método científico aplicado a la robótica. Pista: los mayores ahorros de tiempo suelen venir de tomar las curvas sin frenar tanto, no de subir la velocidad en las rectas.

Solución del ejercicio 5

La solución es tu bitacora de experimentos. Un formato que funciona:

```
Pista: ovalo de 4 m, curvas amplias. Robot: seguidor 2 sensores.
Intento base (velocidad 150): 48 s, 51 s, 47 s. Promedio: 48.7 s.

Cambio 1: velocidad 180.
Resultado: 44 s, 52 s (se salio 1 vez), 45 s. Promedio: 47 s.
Conclusion: mas rapido pero inestable. No conviene.

Cambio 2: vuelvo a 150, agrego tercer sensor (version c).
Resultado: 42 s, 41 s, 43 s. Promedio: 42 s.
Conclusion: el tercer sensor suaviza las curvas y baja 6.7 s. Conviene.

Cambio 3: tercer sensor + velocidad 170.
Resultado: 38 s, 39 s, 38 s. Promedio: 38.3 s.
Conclusion: combinacion ganadora. Mejora total: 10.4 s (21%).
```

La lección profunda de este ejercicio: en robótica autonoma, "mas rápido" casi nunca significa "subir la velocidad de los motores". Significa tomar mejores decisiones: correcciones mas suaves, menos frenadas innecesarias, mejor lectura de sensores. El robot ganador no es el mas potente, es el mejor sintonizado. Guarda tu bitacora: cuando construyas tu siguiente robot, esos numeros valen oro.

Resumen de la unidad

En esta unidad tus robots dejaron de obedecer órdenes y empezaron a tomar decisiones. Aprendiste que un comportamiento autónomo es el ciclo percibir-decidir-actuar repitiéndose cientos de veces por segundo: los sensores convierten el mundo en números, el programa aplica reglas sobre esos números y los motores ejecutan la acción elegida.

La lógica de decisión se escribe con condicionales `if` y `else`, ordenados del caso mas específico al mas general, y con máquinas de estados cuando el comportamiento tiene secuencia (paso 1, luego paso 2, luego paso 3). Aprendiste a manejar los tiempos con `millis()` en lugar de `delay()`, para que el robot nunca quede ciego, y a organizar tus programas con funciones de movimiento reutilizables y parámetros con nombre en vez de números mágicos regados por el código.

Construiste dos robots clásicos sobre el mismo chasis de dos motores con L298N. El seguidor de línea usa sensores TCRT5000 para detectar una línea negra sobre fondo blanco, con lógica bang-bang ("sigue a la línea, no huyas de ella") y calibración obligatoria sobre la pista real con la luz real. Viste la version de dos sensores y la mejora de tres, con correcciones suaves frenando una rueda. El evasor de obstáculos usa el ultrasónico HC-SR04 al frente con un umbral de distancia y una secuencia de escape (detenerse, retroceder, girar, reanudar), primero en version básica que gira siempre al mismo lado y luego en version avanzada con servo explorador que mide a ambos lados y elige el camino con mas espacio.

También aprendiste el método del detective para depurar: probar cada sensor por separado con el monitor serial antes de integrar, verificar las decisiones impresas y validar cada función de movimiento. Y memorizaste los errores comunes: calibrar en una superficie y probar en otra, los delays que ciegan al robot, la velocidad excesiva y las baterías bajas que simulan fallas de programa.

En la práctica sintonizaste la velocidad máxima estable, adaptaste el seguidor a curvas de 90 grados, le diste al evasor dos umbrales de reacción, combinaste línea y obstáculo con prioridades claras (la seguridad manda) y corriste una carrera contra reloj con bitacora de experimentos. Si completaste los cinco ejercicios, ya no solo copias código de robots: diagnosticas comportamientos, calibras parámetros y mejoras con datos.

Lo que sigue en el curso construye sobre esta base: los comportamientos autónomos que programaste aquí son los bloques con los que se arman robots mas complejos, que combinan varios sensores y varias decisiones al mismo tiempo. El chasis que armaste te va a acompañar. Cuidalo, y sobre todo: sigue anotando tus experimentos.

Repasemos los términos clave que debes dominar antes de avanzar: autonomía (el robot decide solo), ciclo percibir-decidir-actuar (el latido de todo robot autónomo), control bang-bang (corrección de todo o nada), máquina de estados (el robot sabe en que situación esta y cada estado tiene su acción y su condición de salida), umbral (el valor de distancia o lectura que dispara una decisión), calibración (ajustar sensores y parámetros sobre la pista real) y sintonizar (buscar con experimentos los mejores valores de velocidad y tiempos).

Una lista de verificación final para tus dos robots. El seguidor: los sensores están a 5-10 milímetros del piso, calibrados sobre la pista real con la luz real, la lógica de giro sigue la regla "gira hacia el sensor que ve negro", la velocidad es la máxima estable que mediste en el ejercicio 1, y no hay delays dentro del loop principal. El evasor: el HC-SR04 apunta al frente a unos 12 centímetros del piso, el umbral de distancia evita choques por inercia, la secuencia de escape usa `millis()` y no `delay()`, y en la version con servo los cables tienen holgura para girar. Si algo falla, vuelve al método del detective: primero sensores por separado, luego decisiones, luego motores.

Un último consejo de taller: guarda cada version de tus programas con nombre y fecha (`seguidor_v1`, `seguidor_v2_curvas`, `evasor_dos_umbrales`). Cuando un cambio empeore el comportamiento, podrás regresar a la version que si funcionaba en segundos. Los programadores profesionales hacen exactamente esto con herramientas sofisticadas; tu puedes hacerlo con nombres de archivo ordenados. Tu yo del futuro te lo va a agradecer.

Felicidades: programaste comportamientos autónomos reales. El robot que sigue una línea y el que esquiva obstáculos son los ancestros directos de las aspiradoras robot, los carros autónomos y los exploradores planetarios. Los principios son los mismos; solo cambia la escala. Nos vemos en la siguiente unidad, donde estos comportamientos se vuelven piezas de robots todavía mas capaces.

Y recuerda la lección mas importante de toda la unidad, la que no esta en ningún código: el robot autónomo perfecto no existe, pero el robot bien depurado si. Cada vez que tu seguidor complete una pista sin salirse, cada vez que tu evasor rodee una caja sin tocarla, estas viendo el resultado de decenas de micro-decisiones correctas por segundo, escritas por ti. Esa sensación de ver a tu creación moverse sola por primera vez no se olvida. Disfrutala, anota que funcionó, y sigue construyendo. La autonomía que programaste hoy es la base de todo lo que viene.

Guarda bien tus dos robots y tus bitácoras: en la próxima unidad los vas a necesitar.

El siguiente capitulo te espera con robots aun mas capaces e interesantes.

UNIDAD 9: BRAZOS ROBÓTICOS

El robot que se mueve por el suelo es un gran logro. Ya lo hiciste: construiste un chasis, le pusiste motores, lo guiaste con sensores y lo hiciste evitar obstáculos. Pero hay otro tipo de robot que no se mueve de su lugar y, sin embargo, es el mas usado en todo el mundo: el brazo robótico. Piensa en las fabricas de autos que aparecen en la television. Esas grandes máquinas metálicas que levantan carrocerías enteras, que sueldan piezas, que atornillan puertas con una precisión imposible para un humano. Todos esos son brazos robóticos. Y en esta unidad vas a construir el tuyo propio, pequeño, barato y funcional, con cuatro servomotores, un Arduino y una pinza que puede tomar objetos.

Un brazo robótico es, en esencia, una cadena de articulaciones móviles conectadas por piezas rígidas llamadas eslabones. Cada articulación puede girar o deslizarse, y el conjunto completo puede posicionar su extremo (donde va la herramienta o la pinza) en distintos puntos del espacio. El extremo del brazo se llama efector final. En un robot industrial, el efector final puede ser una antorcha de soldadura, un taladro, una ventosa o una pinza. En el brazo que vas a construir en esta unidad, el efector final es una pinza de dos dedos que abre y cierra con un servo.

Antes de tocar un solo tornillo, detente un momento y haz un experimento con tu propio cuerpo. Levanta tu brazo derecho frente a ti. Eso que acabas de hacer uso tu hombro. Ahora dobla el codo: tu mano se acerca hacia tu cara. Ahora gira la muñeca: la orientación de tu palma cambia. Acabas de usar tres articulaciones, y cada una aportó un movimiento distinto. Tu hombro, tu codo y tu muñeca son, funcionalmente hablando, servomotores biológicos. El brazo que vas a construir funciona con la misma lógica: una articulación en la base que gira como tu hombro, una articulación intermedia que se dobla como tu codo, y una pinza al final que funciona como tus dedos. Cuando entiendes tu propio brazo, entiendes el brazo robótico. Esa es la belleza de la robótica: muchas máquinas son imitaciones de la naturaleza.

Los brazos robóticos están en todas partes, aunque no los veas. En la industria automotriz, miles de brazos sueldan, pintan y ensamblan vehículos las 24 horas del día. En la industria electrónica, brazos pequeños colocan componentes minúsculos en tarjetas de circuito con una precisión de décimas de milímetro, algo imposible para una mano humana cansada. En medicina, los sistemas de cirugía asistida por robot permiten a un cirujano operar con movimientos miniaturizados: el médico mueve unos controles y el brazo robótico reproduce esos movimientos dentro del cuerpo del paciente, con más precisión y menos temblor. En la exploración espacial, el brazo del robot de Marte recoge muestras del suelo. En los almacenes gigantes de las tiendas en línea, brazos robóticos clasifican paquetes. Y en tu casa, dentro de unos años, probablemente habrá un brazo robótico ayudando en la cocina o en la limpieza.

Todos estos brazos, desde el que suelda un auto hasta el que vas a construir con servos de plástico, comparten los mismos principios. Todos tienen articulaciones. Todos tienen un número definido de grados de libertad. Todos necesitan una fuente de energía, un sistema de control y una estructura mecánica que soporte el peso. La diferencia está en la escala, la precisión y el precio, no en los conceptos. Eso significa que lo que aprendas aquí no es un juguete sin valor: son los fundamentos reales de la robótica industrial.

El concepto más importante de esta unidad es el de grados de libertad. Se abrevia GDL y se pronuncia "grados de libertad". Un grado de libertad es cada movimiento independiente que puede hacer una articulación. Tu hombro, por ejemplo, tiene más de un grado de libertad: puedes levantar el brazo hacia adelante, hacia el lado, y también rotarlo. Tu codo tiene un solo grado de libertad: solo se dobla en una dirección. Tu muñeca tiene dos: puedes inclinarla arriba y abajo, y girarla.

El brazo que vas a construir en esta unidad tiene 4 grados de libertad, uno por cada servo: el servo de la base gira todo el brazo a la izquierda y a la derecha (como cuando giras tu torso), el servo del hombro levanta y baja la primera sección del brazo, el servo del codo dobla la segunda sección, y el servo de la pinza abre y cierra los dedos. Con esos cuatro movimientos puedes tomar un objeto pequeño de una mesa, levantarlo, girar y soltarlo en otro lugar. No parece mucho, pero esa secuencia (tomar, levantar, girar, soltar) es exactamente lo que hacen los brazos industriales cuando clasifican piezas.

Hay una relación directa entre grados de libertad y capacidad: mientras más grados de libertad tiene un brazo, a más lugares puede llegar y en más orientaciones puede colocar su pinza. Un brazo de 2 grados de libertad solo puede moverse en un plano, como dibujar sobre una hoja de papel. Un brazo de 3 grados de libertad ya puede alcanzar puntos en el espacio tridimensional. Un brazo de 6 grados de libertad puede colocar su pinza en cualquier posición y con cualquier orientación, como tu brazo completo. Los brazos industriales típicos tienen 6 grados de libertad por esa razón. El nuestro tendrá 4, que es el equilibrio perfecto entre capacidad y simplicidad para un principiante: suficiente para hacer tareas interesantes, lo bastante simple para construirlo y programarlo sin frustrarte.

Otra idea clave es la diferencia entre la cinemática directa y la cinemática inversa. La cinemática directa responde esta pregunta: si se en qué ángulo está cada articulación, donde queda la pinza. Es como decir: "mi hombro está a 45 grados y mi codo a 90 grados, entonces mi mano está aquí". La cinemática inversa es la pregunta contraria: quiero que la pinza llegue a este punto específico, ¿qué ángulo debe tener cada articulación? La cinemática inversa es mucho más difícil y requiere matemáticas avanzadas. En esta unidad trabajaremos con la cinemática directa de forma intuitiva: vas a mover cada articulación y observar donde queda la pinza, y vas a programar secuencias de movimientos memorizando posiciones. Es el método

mas práctico para empezar y es, de hecho, como se programan muchos brazos industriales en la práctica: un operador mueve el brazo a las posiciones deseadas, el sistema las graba, y luego el brazo las repite. Ese método se llama "grabar y repetir" y lo vas a implementar tu mismo en la clase 27. El brazo que vamos a construir usa servomotores SG90, los pequeños servos azules de plástico que ya conoces de unidades anteriores. Cada servo es una articulación completa: tiene su propio motor, su sistema de engranes y su electrónica de control. Tu solo le dices el ángulo y el se encarga de llegar ahí. Esa es la gran ventaja del servo frente a un motor de corriente continua común: el servo ya trae el control de posición integrado.

La estructura mecánica del brazo es sencilla: una base firme que se atornilla o se pega a una tabla, un soporte vertical donde va el servo del hombro, dos eslabones (piezas planas de madera, acrílico o impresas en 3D) que forman el "antebrazo" y el "brazo", y al final la pinza de dos dedos movida por el cuarto servo. No necesitas una impresora 3D ni herramientas caras: el brazo se puede construir con piezas de madera contrachapada cortadas con segueta, con acrílico, o con los kits de brazo robótico de acrílico que se venden en línea a bajo precio. Lo importante no es el material, sino que las piezas sean rígidas y que las articulaciones giren sin atorarse.

El control del brazo lo harás en dos etapas, que corresponden a las clases 26 y 27. Primero, control manual con potenciómetros: conectas cuatro potenciómetros (uno por cada servo) y al girar cada perilla mueves una articulación. Es la forma mas directa de sentir como responde el brazo, y es útil para calibrar y para probar que todo funciona. Después, movimientos programados: escribes en el código las posiciones que debe tomar cada servo, en secuencia, y el brazo ejecuta la rutina solo. Entre esos dos extremos esta la técnica de grabar y repetir: mueves el brazo con los potenciómetros hasta una posición que te gusta, presionas un botón para guardarla en la memoria del Arduino, repites para varias posiciones, y luego el brazo reproduce toda la secuencia automáticamente. Es la técnica mas divertida de esta unidad y la mas parecida a como se programan los robots reales.

Un aviso importante antes de empezar: los servos consumen mas corriente de la que el Arduino puede entregar por su pin de 5V. Cuando mueves cuatro servos a la vez, especialmente con carga, el Arduino se puede reiniciar o los servos se pueden mover de forma errática. Por eso en esta unidad usarás una fuente de alimentación externa de 5V y al menos 2A dedicada a los servos, conectando el GND (tierra) de la fuente con el GND del Arduino para que ambos "hablen el mismo idioma eléctrico". Este detalle, que parece menor, es la causa número uno de fallas en proyectos con servos. Lo veremos con calma en la sección de explicación.

Al terminar esta unidad habrás construido un brazo robótico de 4 grados de libertad completamente funcional: lo controlarás con potenciómetros, le programarás secuencias de movimiento, le enseñarás a saludar, a tomar y soltar objetos, y lo harás repetir rutinas grabadas. Entenderás que es un grado de libertad, como se relacionan los ángulos de las articulaciones con la posición de la pinza, como alimentar correctamente varios servos y como estructurar un programa que controle varios actuadores a la vez. Son habilidades que te servirán para cualquier robot articulado que construyas en el futuro, desde una mano robótica hasta un hexapodo.

Así que consigue tus servos, tus potenciómetros y tus piezas mecánicas, y prepárate: vas a darle a tu Arduino algo parecido a un brazo humano. Empecemos por entender a fondo cada parte.

Explicación detallada

Articulaciones y grados de libertad

Una articulación es un punto del brazo donde dos piezas rígidas se conectan y pueden moverse una respecto a la otra. En el cuerpo humano, las articulaciones son el hombro, el codo, la muñeca, la rodilla. En un brazo robótico, cada articulación es un servo (o un motor con su sistema de transmisión). El eslabón es la pieza rígida entre dos articulaciones: en tu brazo, el humero (del hombro al codo) y el cubito y radio (del codo a la muñeca) son eslabones. En el brazo robótico que vas a construir, los eslabones son las piezas planas que unen el servo del hombro con el servo del codo, y el servo del codo con la pinza.

Los grados de libertad, como vimos en la introducción, son los movimientos independientes de cada articulación. Pero vale la pena profundizar porque este concepto determina todo lo que tu brazo podrá o no podrá hacer. Un servo SG90 común tiene un rango de giro de 0 a 180 grados, pero en la práctica de un brazo robótico no siempre usas el rango completo: los topes mecánicos de la estructura, los cables y la propia geometría del brazo limitan el movimiento útil de cada articulación. Por ejemplo, el servo del hombro de tu brazo probablemente solo usará entre 20 y 160 grados, porque más allá de esos límites el brazo chocaría con la base o con la mesa.

Vamos a analizar el brazo de 4 servos articulación por articulación, de la base hacia la pinza, que es el orden estándar para describir un brazo robótico:

1. Base (servo 1): gira todo el brazo sobre su eje vertical, de izquierda a derecha. Es como girar tu torso en la silla. Este grado de libertad es el que le da al brazo su alcance circular:

sin el, el brazo solo podría moverse en un plano vertical frente a el, como un limpiaparabrisas. Con el, la pinza puede alcanzar cualquier punto de un círculo alrededor de la base.

2. Hombro (servo 2): levanta y baja el primer eslabón. Es el servo que mas fuerza necesita porque carga con el peso de todo lo que esta encima: el segundo eslabón, la pinza y el objeto que estés levantando. Piensa en cuando levantas una bolsa pesada del supermercado con el brazo extendido: tu hombro hace el mayor esfuerzo. Por eso, en brazos mas grandes, el servo o motor del hombro es siempre el mas potente.
3. Codo (servo 3): dobla el segundo eslabón respecto al primero. Trabaja en conjunto con el hombro para posicionar la pinza a la altura y distancia deseadas. Si el hombro levanta el brazo, el codo decide que tan extendida o recogida queda la "mano". La combinación de hombro y codo es la que determina a que distancia de la base y a que altura queda la pinza.
4. Pinza (servo 4): abre y cierra los dos dedos. Técnicamente, abrir y cerrar la pinza no cambia la posición del brazo en el espacio, así que algunos puristas no lo cuentan como grado de libertad de posicionamiento. Pero para efectos prácticos de este curso, lo contamos como el cuarto grado de libertad porque es un movimiento independiente que tu controlas. Sin la pinza, el brazo podría empujar cosas pero no agarrarlas, y agarrar es la mitad de la diversion.

Fíjate en algo importante: el orden en que nombramos las articulaciones va de la mas cercana a la base (la que mueve mas peso y define el movimiento mas amplio) a la mas lejana (la que hace el ajuste fino). Esta jerarquia es util cuando programas: primero posicionas la base y el hombro (movimientos grandes), luego ajustas el codo (movimiento medio) y al final la pinza (acción fina). Es el mismo orden en que tu cerebro organiza un movimiento: para tomar un vaso, primero giras el torso, luego extiendes el brazo, luego orientas la mano y al final cierras los dedos.

Un concepto util para visualizar los grados de libertad es el espacio de trabajo. El espacio de trabajo de un brazo es el conjunto de todos los puntos a los que puede llegar su pinza. Para nuestro brazo de 4 servos, el espacio de trabajo tiene forma aproximada de una porcion de esfera (o de dona, si consideras que la base gira 360 grados): hay una distancia mínima (muy cerca de la base, donde los eslabones se pliegan) y una distancia máxima (con el brazo totalmente extendido) a las que la pinza puede llegar. Fuera de ese volumen, por mas que programes, la pinza no va a llegar. Entender los limites del espacio de trabajo te evita horas de frustración programando posiciones imposibles.

Tipos de pinzas

El efector final mas común en brazos pequeños es la pinza de dos dedos, también llamada gripper. Funciona como unas pinzas de depilar gigantes: dos dedos paralelos o en angulo que se abren y se cierran con un solo servo. Cuando el servo gira en una dirección, un mecanismo (usualmente un engrane pequeño o una biela) abre los dos dedos de forma simetrica; cuando gira en la dirección contraria, los cierra.

La pinza de dos dedos es ideal para principiantes por tres razones: es fácil de construir (dos piezas móviles y un servo), es fácil de controlar (un solo valor de angulo: abierto o cerrado, con posiciones intermedias), y sirve para la mayoría de objetos pequeños (cubos, pelotas de ping pong, tornillos grandes, dulces). Sus limitaciones también son instructivas: no puede tomar objetos muy grandes, ni muy pesados, ni con formas muy irregulares. Si algún día quieres tomar objetos mas complejos, existen pinzas de tres dedos, ventosas de vacío y pinzas magneticas, pero todas se controlan con la misma idea: un servo que abre y cierra.

Un detalle práctico que marca la diferencia: pega un pedazo pequeño de foami, goma o lija fina en la cara interna de cada dedo de la pinza. Ese material aumenta la fricción y evita que los objetos resbalen. Es un truco barato que usan hasta los robots industriales (sus pinzas llevan recubrimientos de goma) y transforma una pinza mediocre en una pinza confiable. También puedes doblar ligeramente las puntas de los dedos hacia adentro para formar una pequeña "cuchara" que ayude a retener objetos redondos como pelotas de ping pong.

El rango util del servo de la pinza no es de 0 a 180 grados. En la práctica, la pinza completamente abierta corresponde a un angulo (por ejemplo 90 grados) y la pinza completamente cerrada a otro (por ejemplo 20 grados), y esos valores dependen de como hayas montado el mecanismo. Por eso la calibración que harás en los ejercicios es tan importante: cada brazo es ligeramente distinto y los numeros "mágicos" se descubren probando, no adivinando.

Cinematica directa intuitiva

La cinematica directa responde una pregunta sencilla: si se el angulo de cada articulacion, donde queda la pinza. No necesitas fórmulas para entenderla; necesitas experimentar. Vamos a hacerlo con ejemplos concretos usando tu brazo de 4 servos. Imagina que el brazo esta montado sobre una mesa y que defines los ángulos en grados, donde 90 es la posición "neutral" o central de cada servo.

Ejemplo 1: todos los servos en 90 grados. El brazo queda en una posición intermedia: la base apuntando al frente, el hombro a la mitad de su recorrido, el codo a la mitad, la pinza a medio abrir. Es la posición de "descanso" o posición inicial desde la que conviene partir en tus

programas. Siempre que enciendas el brazo, llevalo primero a esta posición neutral antes de hacer cualquier otra cosa: así evitas movimientos bruscos desde una posición desconocida.

Ejemplo 2: mueves solo el servo de la base de 90 a 140 grados. Todo el brazo gira hacia un lado como una torre. La altura de la pinza no cambia, la distancia a la base no cambia, solo cambia la dirección a la que apunta. Este es el movimiento mas fácil de predecir: la pinza describe un arco horizontal.

Ejemplo 3: mueves solo el servo del hombro de 90 a 140 grados. El primer eslabón se levanta y con el sube todo lo demás: el codo, el segundo eslabón y la pinza suben juntos. La pinza gana altura pero también se acerca un poco a la base, porque al levantar el primer eslabón su proyección horizontal se acorta. Es como levantar tu brazo extendido: tu mano sube pero también se acerca a tu cuerpo.

Ejemplo 4: mueves solo el servo del codo de 90 a 40 grados. El segundo eslabón se pliega sobre el primero y la pinza se acerca mucho a la base, bajando su altura. Es el movimiento de "recoger": sirve para traer un objeto cercano hacia ti o para plegar el brazo en una posición compacta.

Ejemplo 5: combinas hombro a 130 y codo a 50. El brazo se extiende hacia adelante y hacia arriba, y la pinza queda lejos de la base y a buena altura. Esta es una típica posición de "alcanzar un objeto lejano sobre la mesa".

La lección de estos ejemplos es que la posición de la pinza es el resultado acumulado de todos los ángulos. Cada articulación aporta su parte y el efecto se suma a lo largo de la cadena. Por eso, cuando programes secuencias, no pienses en cada servo por separado sino en la pose completa del brazo: una pose es el conjunto de los cuatro ángulos en un instante dado. "Base 90, hombro 120, codo 60, pinza 30" es una pose. Tu programa será, en el fondo, una lista de poses y transiciones entre ellas.

Hay un fenómeno que vas a notar de inmediato cuando muevas el brazo con los potenciómetros: las articulaciones cercanas a la base tienen un efecto grande y las lejanas un efecto fino. Mover la base 10 grados desplaza la pinza varios centímetros; mover la pinza 10 grados apenas cambia la posición del brazo (solo abre o cierra los dedos). Esto es lógico: la base mueve toda la estructura, así que su efecto se amplifica con la longitud del brazo. Tenlo en cuenta cuando busques precisión: los ajustes finos se hacen con el codo, los movimientos grandes con la base y el hombro.

Otro fenómeno importante es la gravedad. A diferencia de un robot móvil sobre ruedas, donde la gravedad apenas importa, en un brazo articulado la gravedad es un actor principal. Cuando el brazo esta extendido horizontalmente, el servo del hombro sostiene todo el peso con el

brazo de palanca mas largo posible, y ahí es donde mas corriente consume y mas se calienta. Cuando el brazo esta plegado o en vertical, el esfuerzo es mínimo. Por eso verás que el servo del hombro se calienta mas que los demás: es normal hasta cierto punto, pero si quema al tacto, algo anda mal (lo veremos en errores comunes).

El brazo de 4 servos SG90: lista de materiales

Antes de armar, reúne todo. No hay nada mas frustrante que estar a mitad del montaje y descubrir que te falta un tornillo. Esta es la lista completa:

- 4 servomotores SG90 (los azules de plástico). Verifica que los cuatro funcionen antes de montarlos: conéctalos uno por uno al Arduino con el ejemplo "Sweep" de la librería Servo y observa que giren suavemente en todo su rango.
- 1 Arduino Uno (o compatible) con su cable USB.
- 4 potenciómetros de 10k ohmios lineales, con sus perillas. Serán tus "controles manuales", uno por articulación.
- 1 protoboard grande para conectar todo de forma ordenada.
- 1 fuente de alimentación externa de 5V y mínimo 2A (un eliminador o cargador de pared de buena calidad). No uses el pin 5V del Arduino para alimentar los servos.
- Cables jumper macho-macho y macho-hembra, varios.
- Estructura mecánica del brazo: puedes usar un kit de brazo robótico de acrílico para SG90 (se consigue en línea y trae todas las piezas cortadas y la tornillería), o construirla con madera contrachapada de 3 mm, o piezas impresas en 3D si tienes acceso a una impresora. El kit de acrílico es la opción recomendada para principiantes porque las piezas ya vienen a la medida.
- Tornillería: tornillos M2 o M3 con tuercas, según tu estructura. Si usas el kit de acrílico, ya vienen incluidos.
- 1 botón pulsador (para la función de grabar posiciones) y 1 resistencia de 10k ohmios (o usa el pull-up interno del Arduino).
- 1 tabla o base de madera de al menos 20 x 20 cm donde fijar el brazo, para que no se voltee.
- Opcional pero recomendado: un capacitor electrolítico de 470 uF o 1000 uF a 10V o mas, para conectar entre 5V y GND cerca de los servos y suavizar los picos de corriente.
- Cinta adhesiva, cinchos de plástico (para sujetar cables) y foami o goma para las puntas de la pinza.

Sobre los servos SG90: son baratos y suficientes para un brazo pequeño, pero tienen límites que debes respetar. Su torque es de aproximadamente 1.8 kg-cm, lo que significa que pueden levantar poco peso si el brazo es largo. Mantén los eslabones cortos (10 a 15 cm cada uno) y los objetos ligeros (menos de 50 gramos). Si intentas levantar algo pesado con el brazo extendido, el servo del hombro se forzará, se calentará y eventualmente se dañará. Respetar los límites del hardware es parte de ser un buen robotista.

Montaje mecánico paso a paso

Arma el brazo sobre una mesa despejada y con buena luz. Ten a la mano un desarmador pequeño. El orden de montaje importa: se construye de abajo hacia arriba, de la base a la pinza, igual que el orden en que describimos las articulaciones.

Paso 1: prepara la base. Fija la base del brazo a la tabla de madera con tornillos o con pegamento fuerte. La base debe quedar perfectamente firme: todo el brazo se apoya en ella y cualquier movimiento de la base se traduce en imprecisión en la pinza. Si la base se tambalea, el brazo no servirá. Asegúrate de que la tabla sea lo bastante pesada o amplía para que el brazo extendido no la voltee.

Paso 2: monta el servo de la base. El servo 1 va en posición horizontal, con su eje apuntando hacia arriba. Sobre el eje del servo se fija la plataforma giratoria (el "hombro" de la estructura) usando el brazo plástico (horn) que viene con el servo, atornillado a la plataforma. Antes de atornillar, gira manualmente el servo a su posición central (90 grados) usando el programa de prueba: así la plataforma quedará centrada y tendrá el mismo recorrido hacia ambos lados. Este truco de centrar cada servo antes de fijar su brazo plástico se repite en cada articulación y es la mitad del secreto de un brazo bien calibrado.

Paso 3: monta el servo del hombro. El servo 2 va en posición vertical sobre la plataforma giratoria, con su eje horizontal apuntando hacia un lado. A su eje se fija el primer eslabón (la pieza larga que sube). De nuevo: centra el servo a 90 grados antes de fijar el eslabón, y procura que con el servo en 90 el eslabón quede en posición vertical o ligeramente inclinada. El servo del hombro es el que más sufre, así que verifica que sus tornillos queden bien apretados y que el eslabón no tenga juego (movimiento flojo) sobre el eje.

Paso 4: monta el servo del codo. El servo 3 va en el extremo superior del primer eslabón, también con el eje horizontal. A su eje se fija el segundo eslabón. Centra a 90 grados antes de fijar. El segundo eslabón suele ser un poco más corto que el primero: eso reduce la carga sobre el hombro.

Paso 5: monta la pinza. El servo 4 va al final del segundo eslabón y mueve los dos dedos de la pinza mediante el mecanismo del kit (usualmente un pequeño engrane o una pieza que

convierte el giro del servo en apertura simétrica de los dedos). Ajusta el mecanismo de modo que con el servo en un extremo del recorrido util los dedos queden bien abiertos y en el otro extremo queden cerrados sin forzarse entre sí. Si los dedos chocan y el servo sigue empujando, se calienta y se daña: deja siempre un pequeño margen.

Paso 6: organiza los cables. Los cuatro servos tienen tres cables cada uno: señal (usualmente naranja o amarillo), positivo (rojo) y tierra (marrón o negro). Lleva los cables por un lado de la estructura sujetándolos con cinchos, dejando suficiente holgura (cable de sobra) en cada articulación para que el movimiento no los tense. Los cables tensos son una causa común de fallas intermitentes: el brazo funciona bien en unas posiciones y falla en otras porque un cable se estira y hace falso contacto. Deja "panza" de cable en cada articulación, como las mangueras de una excavadora.

Paso 7: prueba mecánica en seco. Antes de conectar la electrónica, mueve cada articulación con la mano suavemente en todo su recorrido. Deben girar sin atorarse, sin raspar y sin juego excesivo. Si algo raspa, revisa que no haya tornillos demasiado largos rozando, o piezas mal alineadas. Es mucho más fácil corregir problemas mecánicos ahora que cuando ya hay cables y código de por medio.

Alimentación: la fuente externa y el GND común

Este tema merece su propia sección porque es donde más proyectos fracasan. Un servo SG90 en movimiento consume entre 200 y 500 mA, y en el arranque o bajo carga puede pedir picos de hasta 700 mA. Cuatro servos moviéndose a la vez pueden pedir más de 1.5A en picos. El regulador de 5V del Arduino Uno puede entregar de forma segura unos 500 mA cuando se alimenta por USB (y un poco más con eliminador, pero no mucho). La cuenta no sale: si alimentas los cuatro servos desde el pin 5V del Arduino, el voltaje se desploma, el Arduino se reinicia solo, los servos se vuelven locos y tu concluyes, erróneamente, que tu código está mal.

La solución es una fuente de alimentación externa de 5V con al menos 2A de capacidad, dedicada exclusivamente a los servos. El cableado es así: el cable rojo (positivo) de cada servo va al positivo de la fuente externa; el cable marrón o negro (tierra) de cada servo va al negativo de la fuente externa; y el cable de señal (naranja o amarillo) de cada servo va a un pin digital del Arduino (por ejemplo, pines 3, 5, 6 y 9, que soportan PWM, aunque la librería Servo puede usar otros pines digitales). El Arduino se alimenta por separado con su cable USB o su propio eliminador.

El detalle crítico: el GND (tierra, negativo) de la fuente externa DEBE conectarse al GND del Arduino. Sin esa conexión común, la señal que el Arduino envía a los servos no tiene

referencia y los servos no la entienden: se quedan quietos o tiemblan. Es el error mas común y el mas fácil de olvidar. Repítelo como mantra: "positivo de servos a la fuente, señales al Arduino, tierras todas juntas".

El capacitor electrolítico opcional (470 uF o 1000 uF) se conecta entre el positivo y el negativo de la fuente, lo mas cerca posible de los servos, respetando la polaridad (la pata larga al positivo). Su función es entregar energía instantánea en los picos de arranque de los servos, evitando que el voltaje baje momentáneamente. No es obligatorio, pero si notas que los servos "parpadean" o pierden fuerza en movimientos bruscos, el capacitor suele resolverlo.

Nunca alimentes los servos con mas de 6V: el SG90 esta diseñado para 4.8V a 6V y con mas voltaje se quema. Y nunca conectes la fuente externa al pin 5V del Arduino "para que también se alimente de ahí": el pin 5V no esta pensado como entrada de esa forma en todos los modelos y puedes dañar la placa. Arduino por su USB, servos por su fuente, tierras unidas. Así de simple.

Control con potenciómetros

El control manual con potenciómetros es la primera forma de darle vida al brazo y la herramienta principal para calibrarlo y probar ideas. La idea es de una simpleza hermosa: un potenciómetro por articulacion. Giras la perilla 1 y se mueve la base; giras la perilla 2 y se mueve el hombro; y así con el codo y la pinza. Tu mano se convierte en el "cerebro" del brazo y el Arduino solo traduce.

Un potenciómetro conectado entre 5V y GND, con su terminal central a una entrada analógica del Arduino, entrega valores de 0 a 1023 según la posición de la perilla. Un servo espera ángulos de 0 a 180 grados. La función `map()` hace la traducción: `map(valorPot, 0, 1023, 0, 180)` convierte el rango del potenciómetro al rango del servo. Es la misma técnica que usaste en unidades anteriores para controlar un servo con una perilla; aquí simplemente la repites cuatro veces, una por articulacion.

Hay dos refinamientos importantes. Primero, los limites reales: como vimos, cada articulacion del brazo tiene topes mecanicos y no conviene usar el rango completo de 0 a 180. En lugar de mapear a (0, 180), mapea a los limites seguros de cada articulacion, por ejemplo (20, 160) para la base, (30, 150) para el hombro, y los valores que hayas calibrado para la pinza. Así, aunque gires la perilla hasta el tope, el servo nunca intentara ir a una posición imposible. Es una protección por software que cuida tu hardware.

Segundo, el suavizado. Los potenciómetros baratos tienen un poco de ruido eléctrico: el valor leído brinca unos dígitos aunque no toques la perilla, y eso hace que el servo tiemble ligeramente. La solución simple es leer el potenciómetro varias veces y promediar, o solo

actualizar el servo si el nuevo valor difiere del anterior en mas de un pequeño umbral (por ejemplo, 3 grados). En los ejemplos de código verás ambas técnicas aplicadas de forma sencilla.

El modo de potenciómetros también es tu banco de pruebas para la cinemática directa: mueve una perilla y observa donde queda la pinza, combina dos perillas y observa el efecto acumulado. Este juego libre, que parece simple, es como los ingenieros aprenden el comportamiento de un brazo nuevo antes de programarlo. Dedícale tiempo: gira, observa, anota que combinaciones de ángulos producen poses útiles (como "pinza sobre la mesa al frente" o "brazo plegado"). Esas poses anotadas serán la materia prima de tus secuencias programadas.

Movimientos programados

Programar movimientos significa escribir en el código las poses que el brazo debe adoptar, en orden, con pausas entre ellas. La estructura de datos natural para esto es el arreglo: un arreglo por servo (o un arreglo bidimensional) con la lista de ángulos de cada pose. Por ejemplo, si tu rutina de "saludo" tiene 4 poses, guardas 4 valores para la base, 4 para el hombro, 4 para el codo y 4 para la pinza, y un ciclo `for` recorre las poses enviando cada valor a su servo, con un `delay` entre pose y pose.

El problema del `delay` simple es que el movimiento se ve robótico y brusco: el servo salta de una pose a la siguiente a toda velocidad. La solución es la función de movimiento suave: en lugar de enviar el ángulo final de golpe, la función avanza grado por grado (o en pasos pequeños) desde el ángulo actual hasta el ángulo destino, con una pequeña pausa entre pasos. El brazo se mueve de forma fluida y elegante, y además reduces los picos de corriente porque los servos no arrancan de golpe. En los ejemplos implementarás una función `moverSuave(servo, destino, velocidad)` que encapsula esta lógica y que reutilizarás en todos tus programas de secuencias.

La técnica de grabar y repetir combina lo mejor de los dos mundos. Funciona así: el brazo está en modo manual (potenciómetros). Lo mueves a una pose que te gusta. Presionas el botón de grabar: el Arduino lee los cuatro ángulos actuales y los guarda en arreglos en la memoria (y si quieres que sobrevivan al apagar, en la EEPROM). Repites el proceso para cada pose de tu rutina: tomar objeto, levantar, girar, soltar, regresar. Cuando terminas de grabar, cambias a modo automático (con otro botón o manteniendo presionado) y el brazo reproduce la secuencia completa, usando la función de movimiento suave entre pose y pose, en ciclo hasta que lo detengas.

Esta técnica es poderosa por una razón práctica: es muchísimo mas fácil "mostrarle" al brazo lo que quieres que calcular los ángulos a mano. Tu ojo y tu mano encuentran la pose correcta en segundos; calcular "base 97, hombro 118, codo 64" para tomar un cubo específico te tomaría muchos intentos de prueba y error. Por eso los brazos industriales reales se programan así: un operador los guía (a mano o con un control) por la tarea una vez, el sistema graba, y luego repite miles de veces. Tu brazo de escritorio usará el mismo principio que un robot de fábrica. No esta nada mal.

Un detalle de implementacion: la memoria RAM del Arduino Uno es limitada (2 KB). Guardar 10 poses de 4 ángulos como bytes (cada angulo cabe en un byte porque va de 0 a 180) ocupa solo 40 bytes, así que no hay problema. Si quieres mas poses o que se conserven al apagar, usa la EEPROM (1 KB en el Uno), que también sobra para este uso. En el ejemplo de la clase 27 usarás arreglos en RAM con capacidad para 10 poses, que es mas que suficiente para rutinas interesantes.

Errores comunes y buenas prácticas

Error 1: servos que se calientan. Un servo tibio después de trabajar es normal; un servo que quema al tacto es una alarma. Causas típicas: carga excesiva (brazo muy largo u objeto muy pesado), el servo empujando contra un tope mecánico (la pinza cerrando contra si misma, o un angulo programado mas allá del límite físico), o la fuente de alimentación insuficiente que hace que el servo trabaje a bajo voltaje y se esfuerce mas. Solución: reduce la carga, ajusta los limites de angulo en el código, verifica la fuente. Regla de oro: si un servo se calienta mucho, apaga todo y revisa la mecánica antes de seguir programando.

Error 2: el brazo se cae o no sostiene la posición. El SG90 tiene torque limitado. Si el brazo extendido con un objeto se "vence" y baja solo, estas pidiendo mas de lo que el servo puede dar. Soluciones: acorta los eslabones, aligera la estructura (el acrílico pesa menos que la madera gruesa), no levantes objetos de mas de 50 gramos, y evita mantener el brazo extendido horizontalmente por mucho tiempo: es la posición de máximo esfuerzo. Diseña tus rutinas para que el brazo "descanse" en posiciones plegadas entre movimientos.

Error 3: el Arduino se reinicia cuando se mueven varios servos. Síntoma clásico de alimentación insuficiente o de servos conectados al 5V del Arduino. Solución: fuente externa de 5V 2A mínimo, solo para servos, con GND común. Si ya tienes fuente externa y sigue pasando, revisa que los cables de alimentación sean gruesos y cortos: cables largos y delgados tienen resistencia y el voltaje cae en ellos.

Error 4: los servos tiemblan o se mueven solos. Causas: ruido en los potenciómetros (aplica el umbral de cambio mínimo), GND común mal conectado o flojo, o la señal de un servo

compartiendo un cable largo con cables de potencia (separa en lo posible los cables de señal de los de alimentación). También puede ser un potenciómetro defectuoso: prueba cambiándolo de pin para diagnosticar.

Error 5: cables que se enredan o se tensan. Con la base girando 180 grados una y otra vez, los cables de los servos superiores pueden enrollarse alrededor de la estructura. Solución: deja holgura generosa, sujeta los cables a la estructura con cinchos dejando "panza" en cada articulación, y si tu rutina gira la base muchas veces en la misma dirección, programa que de vez en cuando regrese (como desenrollar).

Buenas prácticas para cerrar: siempre inicia tus programas llevando el brazo a la posición neutral (90 en todo) de forma suave; define los límites de cada articulación como constantes al inicio del código para ajustarlos en un solo lugar; prueba cada rutina primero sin objeto en la pinza y a velocidad baja; y anota en una libreta (o en comentarios del código) las poses útiles que descubras con los potenciómetros, con una descripción ("pose tomar cubo", "pose soltar en caja"). Esa libreta de poses es oro cuando empieces a programar secuencias complejas.

Ejemplos paso a paso

En esta sección vas a construir los cuatro programas fundamentales del brazo robótico. Cada ejemplo incluye el código completo comentado y una explicación por partes. Todos compilan en Arduino Uno y usan la librería Servo.h. Antes de cargar cualquier programa, verifica el cableado: señales de servos a los pines indicados, potenciómetros a las entradas analógicas, fuente externa para los servos con GND común al Arduino.

Ejemplo A: control de 4 servos con 4 potenciómetros

Este es el programa base de todo lo demás. Cuatro potenciómetros conectados a A0, A1, A2 y A3 controlan los servos de base, hombro, codo y pinza conectados a los pines 3, 5, 6 y 9. Incluye límites de seguridad por articulación y un umbral anti-temblor.

```
#include <Servo.h>

// Pines de señal de los servos
const int PIN_BASE = 3;
const int PIN_HOMBRO = 5;
const int PIN_CODO = 6;
const int PIN_PINZA = 9;

// Pines analogicos de los potenciometros
const int POT_BASE = A0;
```



```

const int POT_HOMBRO = A1;
const int POT_CODO = A2;
const int POT_PINZA = A3;

// Limites seguros de cada articulacion (ajustalos a tu brazo)
const int BASE_MIN = 20, BASE_MAX = 160;
const int HOMBRO_MIN = 30, HOMBRO_MAX = 150;
const int CODO_MIN = 20, CODO_MAX = 160;
const int PINZA_ABIERTA = 90, PINZA_CERRADA = 25;

// Umbral anti-temblor: solo mueve el servo si el cambio supera esto
const int UMBRAL = 3;

Servo srvBase, srvHombro, srvCodo, srvPinza;
int ultBase = 90, ultHombro = 90, ultCodo = 90, ultPinza = 60;

void setup() {
  srvBase.attach(PIN_BASE);
  srvHombro.attach(PIN_HOMBRO);
  srvCodo.attach(PIN_CODO);
  srvPinza.attach(PIN_PINZA);
  // Posicion neutral inicial
  srvBase.write(90);
  srvHombro.write(90);
  srvCodo.write(90);
  srvPinza.write(60);
  delay(1000);
}

// Lee un potenciómetro y lo convierte al rango del servo
int leerAngulo(int pinPot, int minA, int maxA) {
  int valor = analogRead(pinPot);      // 0 a 1023
  // Promedio de 5 lecturas para reducir el ruido
  for (int i = 0; i < 4; i++) {
    valor += analogRead(pinPot);
    delay(2);
  }
  valor = valor / 5;
  return map(valor, 0, 1023, minA, maxA);
}

void loop() {
  int aBase = leerAngulo(POT_BASE, BASE_MIN, BASE_MAX);
  int aHombro = leerAngulo(POT_HOMBRO, HOMBRO_MIN, HOMBRO_MAX);
  int aCodo = leerAngulo(POT_CODO, CODO_MIN, CODO_MAX);

```

```

int aPinza = leerAngulo(POT_PINZA, PINZA_CERRADA, PINZA_ABIERTA);

// Solo actualiza si el cambio supera el umbral (evita temblores)
if (abs(aBase - ultBase) > UMBRAL) { srvBase.write(aBase); ultBase = aBase; }
    if (abs(aHombro - ultHombro) > UMBRAL) { srvHombro.write(aHombro); ultHombro = aHombro; }
if (abs(aCodo - ultCodo) > UMBRAL) { srvCodo.write(aCodo); ultCodo = aCodo; }
if (abs(aPinza - ultPinza) > UMBRAL) { srvPinza.write(aPinza); ultPinza = aPinza; }

delay(20);
}

```

Explicación por partes: primero se incluyen la librería y se definen los pines como constantes, para que cambiar el cableado sea cuestión de editar una línea. Los límites (BASE_MIN, BASE_MAX, etc.) protegen la mecánica: aunque gires la perilla al máximo, el servo nunca irá más allá de lo seguro. La función leerAngulo() hace dos trabajos: promedia 5 lecturas para suavizar el ruido del potenciómetro y usa map() para convertir el rango 0-1023 al rango útil de cada servo. Fíjate que para la pinza el mapeo va de PINZA_CERRADA a PINZA_ABIERTA, es decir, el orden está invertido según como montaste tu mecanismo; si tu pinza abre al revés, intercambia esos dos valores. En loop(), cada articulación se actualiza solo si el nuevo ángulo difiere del anterior en más de UMBRAL grados: esto elimina el temblor por ruido eléctrico sin que notes ningún retraso al girar las perillas. El delay(20) le da un respiro al procesador y estabiliza las lecturas.

Prueba sugerida: carga el programa, gira cada perilla lentamente y observa que articulación se mueve. Si una perilla mueve la articulación equivocada, intercambia los pines en las constantes. Anota en una libreta las combinaciones de ángulos que producen poses útiles: las necesitarás en el siguiente ejemplo.

Ejemplo B: secuencia programada, el brazo saluda

Ahora el brazo se mueve solo. Este programa ejecuta una rutina de saludo: se endereza, levanta la "mano", la mueve de un lado a otro dos veces (como saludando) y regresa a la posición neutral. La clave es la función moverSuave(), que lleva cada servo a su destino grado por grado para lograr movimientos fluidos.

```

#include <Servo.h>

const int PIN_BASE = 3;
const int PIN_HOMBRO = 5;
const int PIN_CODO = 6;
const int PIN_PINZA = 9;

```

```

Servo srvBase, srvHombro, srvCodo, srvPinza;

// Posicion actual de cada servo (se actualiza en moverSuave)
int posBase = 90, posHombro = 90, posCodo = 90, posPinza = 60;

void setup() {
  srvBase.attach(PIN_BASE);
  srvHombro.attach(PIN_HOMBRO);
  srvCodo.attach(PIN_CODO);
  srvPinza.attach(PIN_PINZA);
  irAPose(90, 90, 90, 60, 15); // neutral al encender
  delay(1000);
}

// Mueve UN servo de forma suave hasta el destino
void moverSuave(Servo &s, int &posActual, int destino, int espera) {
  if (destino > posActual) {
    for (int a = posActual; a <= destino; a++) { s.write(a); delay(espera); }
  } else {
    for (int a = posActual; a >= destino; a--) { s.write(a); delay(espera); }
  }
  posActual = destino;
}

// Mueve los 4 servos a una pose completa (secuencia: base, hombro, codo, pinza)
void irAPose(int b, int h, int c, int p, int espera) {
  moverSuave(srvBase, posBase, b, espera);
  moverSuave(srvHombro, posHombro, h, espera);
  moverSuave(srvCodo, posCodo, c, espera);
  moverSuave(srvPinza, posPinza, p, espera);
}

void loop() {
  // 1. Enderezarse: brazo vertical, pinza arriba como una mano
  irAPose(90, 140, 150, 60, 15);
  delay(500);
  // 2. Saludar: mover la "mano" (codo) de un lado a otro, dos veces
  irAPose(90, 140, 120, 60, 12);
  delay(300);
  irAPose(90, 140, 160, 60, 12);
  delay(300);
  irAPose(90, 140, 120, 60, 12);
  delay(300);
  irAPose(90, 140, 160, 60, 12);
  delay(500);
}

```

```
// 3. Regresar a neutral
irAPose(90, 90, 90, 60, 15);
delay(3000); // pausa antes de repetir el saludo
}
```

Explicación por partes: las variables posBase, posHombro, etc. guardan en todo momento donde esta cada servo; son indispensables porque moverSuave() necesita saber desde donde parte. La función moverSuave() recibe el servo, su posición actual (por referencia, para actualizarla), el destino y la velocidad (milisegundos entre pasos: un número mas grande es mas lento). El ciclo for avanza o retrocede de uno en uno según el destino este arriba o abajo. La función irAPose() es azúcar sintactica: mueve los cuatro servos en orden (base, hombro, codo, pinza) con una sola llamada, lo que hace que la rutina del loop() se lea como una lista de poses: "enderezarse", "mano a la izquierda", "mano a la derecha". En loop(), la secuencia del saludo usa el codo para el movimiento de vaiven, que es lo mas parecido a agitar una mano. Ajusta los ángulos a tu brazo: si el saludo se ve timido, aumenta la diferencia entre las dos poses del vaiven.

Nota de diseño: mover los servos en secuencia (uno tras otro) es mas simple y consume menos corriente pico que moverlos todos a la vez, porque nunca hay cuatro servos arrancando simultaneamente. Para rutinas mas elegantes puedes mover varios a la vez, pero para empezar, la secuencia es segura y se ve bien.

Ejemplo C: grabar y repetir, 10 posiciones en ciclo

Este es el programa estrella de la unidad. Combina el control manual con la programación: mueves el brazo con los potenciómetros, presionas un botón para grabar cada pose (hasta 10), y luego el brazo repite la secuencia en ciclo. Es la técnica de "grabar y repetir" de la robótica industrial, implementada en tu escritorio.

```
#include <Servo.h>
#include <EEPROM.h>

const int PIN_BASE = 3, PIN_HOMBRO = 5, PIN_CODO = 6, PIN_PINZA = 9;
const int POT_BASE = A0, POT_HOMBRO = A1, POT_CODO = A2, POT_PINZA = A3;
const int BTN_GRABAR = 7; // boton para grabar pose (con pull-up interno)
const int BTN_MODAL = 8; // boton para cambiar entre manual y automatico

const int MAX_POSES = 10;
// Arreglos donde se guardan las poses grabadas
byte poses[MAX_POSES][4];
int numPoses = 0;
bool modoAuto = false;
```

```

Servo srvBase, srvHombro, srvCodo, srvPinza;
int posBase = 90, posHombro = 90, posCodo = 90, posPinza = 60;

void setup() {
  pinMode(BTN_GRABAR, INPUT_PULLUP);
  pinMode(BTN_MODAL, INPUT_PULLUP);
  srvBase.attach(PIN_BASE);
  srvHombro.attach(PIN_HOMBRO);
  srvCodo.attach(PIN_CODO);
  srvPinza.attach(PIN_PINZA);
  irAPose(90, 90, 90, 60, 15);
}

void moverSuave(Servo &s, int &posActual, int destino, int espera) {
  if (destino > posActual) {
    for (int a = posActual; a <= destino; a++) { s.write(a); delay(espera); }
  } else {
    for (int a = posActual; a >= destino; a--) { s.write(a); delay(espera); }
  }
  posActual = destino;
}

void irAPose(int b, int h, int c, int p, int espera) {
  moverSuave(srvBase, posBase, b, espera);
  moverSuave(srvHombro, posHombro, h, espera);
  moverSuave(srvCodo, posCodo, c, espera);
  moverSuave(srvPinza, posPinza, p, espera);
}

// Lee los 4 potenciómetros y mueve los servos (modo manual)
void controlManual() {
  int b = map(analogRead(POT_BASE), 0, 1023, 20, 160);
  int h = map(analogRead(POT_HOMBRO), 0, 1023, 30, 150);
  int c = map(analogRead(POT_CODO), 0, 1023, 20, 160);
  int p = map(analogRead(POT_PINZA), 0, 1023, 25, 90);
  srvBase.write(b); posBase = b;
  srvHombro.write(h); posHombro = h;
  srvCodo.write(c); posCodo = c;
  srvPinza.write(p); posPinza = p;
}

// Espera a que se suelte un boton (evita rebotes y lecturas multiples)
void esperarSoltar(int pin) {
  while (digitalRead(pin) == LOW) { delay(10); }
  delay(50);
}

```

```

}

void loop() {
  // Boton MOD0: alterna entre manual y automatico
  if (digitalRead(BTN_MOD0) == LOW) {
    esperarSoltar(BTN_MOD0);
    modoAuto = !modoAuto;
    if (modoAuto && numPoses == 0) modoAuto = false; // sin poses no hay auto
  }

  if (!modoAuto) {
    controlManual();
    // Boton GRABAR: guarda la pose actual
    if (digitalRead(BTN_GRABAR) == LOW && numPoses < MAX_POSES) {
      esperarSoltar(BTN_GRABAR);
      poses[numPoses][0] = posBase;
      poses[numPoses][1] = posHombro;
      poses[numPoses][2] = posCodo;
      poses[numPoses][3] = posPinza;
      numPoses++;
      delay(300);
    }
  } else {
    // Modo automatico: reproduce las poses grabadas en ciclo
    for (int i = 0; i < numPoses; i++) {
      irAPose(poses[i][0], poses[i][1], poses[i][2], poses[i][3], 12);
      delay(600); // pausa en cada pose
      if (digitalRead(BTN_MOD0) == LOW) break; // salir si piden modo manual
    }
  }
  delay(20);
}

```

Explicación por partes: el arreglo bidimensional `poses[MAX_POSES][4]` es la memoria de la rutina: cada renglon es una pose y cada columna un servo (0=base, 1=hombro, 2=codo, 3=pinza). Se usa byte porque cada angulo (0-180) cabe en un byte y así 10 poses ocupan solo 40 bytes de RAM. Los dos botones usan `INPUT_PULLUP`, así que se conectan entre el pin y GND sin resistencia externa, y se detectan cuando `digitalRead()` devuelve LOW. La función `esperarSoltar()` evita que una sola presión se registre varias veces por el rebote mecánico del botón. En modo manual, `controlManual()` copia los potenciómetros a los servos y actualiza las variables de posición, de modo que al grabar se guarda exactamente lo que ves. En modo automático, un `for` recorre las poses y las ejecuta con `irAPose()`; el `break` permite interrumpir el ciclo con el botón de modo. Nota que si intentas entrar a modo automático sin haber grabado nada, el programa lo impide: un detalle de robustez que evita comportamientos extraños.

Como usarlo: 1) enciende, el brazo va a neutral; 2) mueve los potenciómetros hasta la primera pose (por ejemplo, pinza sobre un cubo, dedos abiertos); 3) presiona GRABAR; 4) mueve a la segunda pose (dedos cerrando el cubo); 5) presiona GRABAR; sigue así hasta completar tu rutina (tomar, levantar, girar, soltar, regresar); 6) presiona MODO y mira como el brazo repite tu rutina una y otra vez. Si una pose quedó mal, presiona MODO para volver a manual: las poses se conservan y puedes regrabar desde cero reiniciando el Arduino (o mejorando el programa para borrar, como verás en los ejercicios).

Ejemplo D: pinza que abre y cierra con botón

Un ejemplo corto pero muy útil: la pinza se controla con un botón en lugar de un potenciómetro. Cada presión alterna entre abierta y cerrada, con movimiento suave. Es el control ideal cuando las otras tres articulaciones ya están ocupadas con potenciómetros y quieres operar la pinza con una sola mano libre, o cuando la pinza forma parte de una secuencia y solo necesitas dos estados.

```

#include <Servo.h>

const int PIN_PINZA = 9;
const int BTN_PINZA = 7;    // boton entre pin 7 y GND (pull-up interno)

const int PINZA_ABIERTA = 90;
const int PINZA_CERRADA = 25;

Servo srvPinza;
int posPinza = 60;
bool pinzaAbierta = false;

void setup() {
  pinMode(BTN_PINZA, INPUT_PULLUP);
  srvPinza.attach(PIN_PINZA);
  srvPinza.write(posPinza);
  delay(500);
}

void loop() {
  if (digitalRead(BTN_PINZA) == LOW) {
    // Espera a soltar (antirrebote simple)
    while (digitalRead(BTN_PINZA) == LOW) { delay(10); }
    delay(50);
    // Alterna el estado y mueve suave
    int destino = pinzaAbierta ? PINZA_CERRADA : PINZA_ABIERTA;
    if (destino > posPinza) {
      for (int a = posPinza; a <= destino; a++) { srvPinza.write(a); delay(12); }
    } else {
      for (int a = posPinza; a >= destino; a--) { srvPinza.write(a); delay(12); }
    }
    posPinza = destino;
    pinzaAbierta = !pinzaAbierta;
  }
  delay(20);
}

```

Explicación por partes: la variable booleana `pinzaAbierta` recuerda el estado actual para saber hacia donde alternar; el operador ternario elige el destino contrario al estado actual. El movimiento se hace grado por grado con `delay(12)`, lo mismo que `moverSuave()` pero escrito en línea para que el ejemplo sea autocontenido. El `while` espera a que sueltes el botón antes de continuar, lo que evita que una presión larga se interprete como varias presiones. Calibra `PINZA_ABIERTA` y `PINZA_CERRADA` con tu mecanismo real: la pinza debe abrir lo suficiente

para rodear tus objetos de prueba y cerrar lo suficiente para sujetarlos sin que los dedos choquen entre si ni el servo se fuerce al final del recorrido.

Variante util: si prefieres que la pinza solo cierre mientras mantienes el botón presionado (como el gatillo de una herramienta) y se abra al soltarlo, elimina la lógica de alternado y escribe directamente: si el botón esta presionado, ve a PINZA_CERRADA; si no, ve a PINZA_ABIERTA. Es el comportamiento de "hombre muerto" que usan muchas herramientas por seguridad: si sueltas el control, la pinza se abre y suelta el objeto en lugar de quedarse agarrandolo.

Práctica

Ejercicio 1: calibrar los centros de cada servo

Enunciado: antes de programar secuencias, cada articulacion debe tener bien definidos su centro (90 grados mecanicos reales) y sus limites seguros. Escribe un programa que te permita mover cada servo individualmente con un solo potenciómetro y un selector, y úsalo para encontrar y anotar: el angulo donde cada articulacion queda en su posición neutral, y los ángulos mínimo y máximo seguros (donde el mecanismo no choca ni se fuerza). Usa un botón para cambiar cual servo controla el potenciómetro.

Objetivo: entender que los valores "ideales" del código dependen del montaje físico real, y crear tu tabla personal de limites para usar en todos los programas siguientes.

Solución del ejercicio 1

La idea es un "modo servicio": un potenciómetro en A0, un botón en el pin 7 para seleccionar el servo activo (base, hombro, codo, pinza, en ciclo), y el monitor serie mostrando el servo seleccionado y el angulo actual para que anotes los valores.

```

#include <Servo.h>

const int PINES[4] = {3, 5, 6, 9};
const char* NOMBRES[4] = {"BASE", "HOMBRO", "CODO", "PINZA"};
const int POT = A0;
const int BTN_SEL = 7;

Servo servos[4];
int sel = 0;
int ultimoReportado = -1;

void setup() {
  Serial.begin(9600);
  pinMode(BTN_SEL, INPUT_PULLUP);
  for (int i = 0; i < 4; i++) servos[i].attach(PINES[i]);
  Serial.println("Calibracion: gira el potenciómetro, boton para cambiar servo.");
  Serial.println("Servo activo: BASE");
}

void loop() {
  if (digitalRead(BTN_SEL) == LOW) {
    while (digitalRead(BTN_SEL) == LOW) { delay(10); }
    delay(50);
    sel = (sel + 1) % 4;
    Serial.print("Servo activo: ");
    Serial.println(NOMBRES[sel]);
    delay(300);
  }
  int angulo = map(analogRead(POT), 0, 1023, 0, 180);
  servos[sel].write(angulo);
  // Reporta solo si cambio, para no saturar el monitor serie
  if (abs(angulo - ultimoReportado) > 2) {
    Serial.print(NOMBRES[sel]);
    Serial.print(": ");
    Serial.println(angulo);
    ultimoReportado = angulo;
  }
  delay(30);
}

```

Explicación: se usa un arreglo de objetos Servo, una técnica que simplifica mucho el código cuando manejas varios servos iguales: servos[sel] selecciona el activo con un índice. El botón avanza el índice en ciclo con módulo 4. El monitor serie es tu libreta digital: gira el potenciómetro lentamente hasta que la articulacion llegue a su tope mecánico en cada

dirección y anota los valores justo antes de que se fuerce; esos son tus límites seguros. Haz lo mismo para encontrar el centro real de cada articulación. Al terminar tendrás una tabla como: base centro 92, límites 22-158; hombro centro 88, límites 35-145; etc. Copia esos valores a las constantes de tus demás programas y notarás de inmediato movimientos más precisos y servos más contentos.

Ejercicio 2: secuencia de "tomar y soltar"

Enunciado: coloca un cubo pequeño (o una pelota de ping pong) sobre la mesa frente al brazo y una caja pequeña a un lado. Programa una secuencia que: 1) se posicione sobre el cubo con la pinza abierta, 2) baje y cierre la pinza tomando el cubo, 3) levante el brazo, 4) gire la base hacia la caja, 5) baje y abra la pinza soltando el cubo en la caja, 6) regrese a la posición neutral. Usa la función `moverSuave()` del ejemplo B.

Objetivo: practicar la programación de una tarea útil completa como lista de poses, y descubrir que tan precisos deben ser los ángulos para tomar un objeto real.

Solución del ejercicio 2

Primero encuentra las poses con el programa del ejemplo A (potenciómetros) y anotalas. Supongamos que descubriste estas poses para tu brazo (ajusta los números a los tuyos):

- P1 acercar: base 90, hombro 100, codo 70, pinza abierta
- P2 tomar: base 90, hombro 70, codo 55, pinza cerrada (baja y agarra)
- P3 levantar: base 90, hombro 120, codo 80, pinza cerrada
- P4 girar: base 140, hombro 120, codo 80, pinza cerrada
- P5 soltar: base 140, hombro 85, codo 65, pinza abierta
- P6 neutral: 90, 90, 90, semiabierta

```
#include <Servo.h>

const int PIN_BASE = 3, PIN_HOMBRO = 5, PIN_CODO = 6, PIN_PINZA = 9;
const int ABIERTA = 90, CERRADA = 25; // ajusta a tu pinza

Servo srvBase, srvHombro, srvCodo, srvPinza;
int posBase = 90, posHombro = 90, posCodo = 90, posPinza = 60;

void moverSuave(Servo &s, int &posActual, int destino, int espera) {
    if (destino > posActual) {
        for (int a = posActual; a <= destino; a++) { s.write(a); delay(espera); }
```

```

    } else {
        for (int a = posActual; a >= destino; a--) { s.write(a); delay(espera); }
    }
    posActual = destino;
}

void irAPose(int b, int h, int c, int p, int espera) {
    moverSuave(srvBase, posBase, b, espera);
    moverSuave(srvHombro, posHombro, h, espera);
    moverSuave(srvCodo, posCodo, c, espera);
    moverSuave(srvPinza, posPinza, p, espera);
}

void setup() {
    srvBase.attach(PIN_BASE);
    srvHombro.attach(PIN_HOMBRO);
    srvCodo.attach(PIN_CODO);
    srvPinza.attach(PIN_PINZA);
    irAPose(90, 90, 90, 60, 15);
    delay(1000);
}

void loop() {
    irAPose(90, 100, 70, ABIERTA, 14); // 1. posicionarse sobre el cubo
    delay(400);
    irAPose(90, 70, 55, ABIERTA, 14); // bajar con pinza abierta
    delay(300);
    irAPose(90, 70, 55, CERRADA, 14); // 2. cerrar y tomar
    delay(500);
    irAPose(90, 120, 80, CERRADA, 14); // 3. levantar
    delay(400);
    irAPose(140, 120, 80, CERRADA, 14); // 4. girar hacia la caja
    delay(400);
    irAPose(140, 85, 65, CERRADA, 14); // bajar sobre la caja
    delay(300);
    irAPose(140, 85, 65, ABIERTA, 14); // 5. abrir y soltar
    delay(500);
    irAPose(90, 90, 90, 60, 15); // 6. regresar a neutral
    delay(4000); // pausa antes de repetir
}

```

Explicación: la rutina esta dividida en micro-poses con comentarios numerados para que cada paso sea legible: bajar y cerrar son dos poses separadas (primero bajas con la pinza abierta, luego cierras) porque así evitas empujar el cubo al bajar con los dedos cerrados. Los delays entre poses dan tiempo a que el objeto se asiente y a que tu verifiques visualmente cada

etapa. Si el brazo no toma el cubo, el problema casi siempre esta en la pose P2: ajusta de 2 en 2 grados el hombro y el codo hasta que los dedos rodeen el cubo antes de cerrar. Este ejercicio te enseña la lección mas valiosa de la robótica práctica: la precisión se logra iterando con el hardware real, no calculando en papel.

Ejercicio 3: control de velocidad de la secuencia

Enunciado: toma la secuencia de "tomar y soltar" del ejercicio 2 y agrégale control de velocidad con un potenciómetro: al girar la perilla, toda la secuencia se ejecuta mas rápido o mas lento. La velocidad debe poder ajustarse incluso mientras la secuencia esta en curso.

Objetivo: aprender a parametrizar el comportamiento del programa (no solo las posiciones) y entender como la velocidad afecta la estabilidad: muy rápido y el brazo vibra o suelta el objeto; muy lento y la tarea se vuelve eterna.

Solución del ejercicio 3

La clave es que moverSuave() ya recibe la velocidad como parámetro (la espera entre pasos). Basta con leer el potenciómetro antes de cada pose y convertirlo a un rango de espera util, por ejemplo de 4 ms (rapidismo) a 30 ms (lento y elegante).

```
#include <Servo.h>

const int PIN_BASE = 3, PIN_HOMBRO = 5, PIN_CODO = 6, PIN_PINZA = 9;
const int POT_VEL = A0;
const int ABIERTA = 90, CERRADA = 25;

Servo srvBase, srvHombro, srvCodo, srvPinza;
int posBase = 90, posHombro = 90, posCodo = 90, posPinza = 60;

void moverSuave(Servo &s, int &posActual, int destino, int espera) {
    if (destino > posActual) {
        for (int a = posActual; a <= destino; a++) { s.write(a); delay(espera); }
    } else {
        for (int a = posActual; a >= destino; a--) { s.write(a); delay(espera); }
    }
    posActual = destino;
}

void irAPose(int b, int h, int c, int p, int espera) {
    moverSuave(srvBase, posBase, b, espera);
    moverSuave(srvHombro, posHombro, h, espera);
    moverSuave(srvCodo, posCodo, c, espera);
}
```

```

    moverSuave(srvPinza, posPinza, p, espera);
}

// Lee la perilla y devuelve la espera entre pasos: 4 (rapido) a 30 (lento)
int leerVelocidad() {
    int v = analogRead(POT_VEL);
    return map(v, 0, 1023, 30, 4); // invertido: perilla a la derecha = rapido
}

void setup() {
    srvBase.attach(PIN_BASE);
    srvHombro.attach(PIN_HOMBRO);
    srvCodo.attach(PIN_CODO);
    srvPinza.attach(PIN_PINZA);
    irAPose(90, 90, 90, 60, 15);
    delay(1000);
}

void loop() {
    int vel;
    vel = leerVelocidad(); irAPose(90, 100, 70, ABIERTA, vel); delay(400);
    vel = leerVelocidad(); irAPose(90, 70, 55, ABIERTA, vel); delay(300);
    vel = leerVelocidad(); irAPose(90, 70, 55, CERRADA, vel); delay(500);
    vel = leerVelocidad(); irAPose(90, 120, 80, CERRADA, vel); delay(400);
    vel = leerVelocidad(); irAPose(140, 120, 80, CERRADA, vel); delay(400);
    vel = leerVelocidad(); irAPose(140, 85, 65, CERRADA, vel); delay(300);
    vel = leerVelocidad(); irAPose(140, 85, 65, ABIERTA, vel); delay(500);
    vel = leerVelocidad(); irAPose(90, 90, 90, 60, vel); delay(4000);
}

```

Explicación: leerVelocidad() se llama antes de cada pose, así que puedes girar la perilla en cualquier momento y el siguiente movimiento usará la nueva velocidad. El map() esta invertido (30 a 4) para que girar la perilla a la derecha signifique "mas rápido", que es lo intuitivo. Observa que la velocidad también cambia la física de la tarea: a máxima velocidad, la inercia puede hacer que el cubo salga volando de la pinza al frenar bruscamente; a mínima velocidad, todo es suave pero la rutina tarda mucho. El punto ideal suele estar en el medio. Este ejercicio demuestra un principio general: en robótica, la velocidad no es solo estetica, es parte del control.

Ejercicio 4: agregar un quinto movimiento

Enunciado: modifica el programa de grabar y repetir (ejemplo C) para que, además de las 10 poses, el brazo haga una pausa programable entre poses: al grabar cada pose también se graba cuanto tiempo debe esperar el brazo en esa pose (corto, medio o largo, elegido con el potenciómetro de la pinza al momento de grabar). En modo automático, el brazo respeta esas pausas.

Objetivo: practicar la extensión de un programa existente agregando un arreglo paralelo (tiempos) y entender el patrón de "arreglos paralelos": dos arreglos donde el índice *i* de uno corresponde al índice *i* del otro.

Solución del ejercicio 4

Se agrega un arreglo `tiempos[MAX_POSES]` paralelo a `poses[MAX_POSES][4]`. Al grabar, se lee el potenciómetro de la pinza (A3) y se convierte a una pausa de 200, 600 o 1200 ms según su posición (tercio izquierdo, centro, tercio derecho). En modo automático se usa `tiempos[i]` en lugar del delay fijo.

```

// Agregados al programa del ejemplo C:
const int MAX_POSES = 10;
byte poses[MAX_POSES][4];
int tiempos[MAX_POSES];    // NUEVO: pausa en ms despues de cada pose
int numPoses = 0;

// Dentro del bloque de GRABAR, despues de guardar la pose:
int selTiempo = analogRead(POT_PINZA); // reutiliza la perilla de la pinza
if (selTiempo < 341) tiempos[numPoses] = 200;           // perilla izquierda: pausa
corta
else if (selTiempo < 682) tiempos[numPoses] = 600;    // centro: pausa media
else tiempos[numPoses] = 1200;                       // derecha: pausa larga
numPoses++;

// En modo automatico, el ciclo de reproduccion queda asi:
for (int i = 0; i < numPoses; i++) {
    irAPose(poses[i][0], poses[i][1], poses[i][2], poses[i][3], 12);
    // Pausa grabada, pero interrumpible con el boton de modo
    unsigned long inicio = millis();
    while (millis() - inicio < (unsigned long)tiempos[i]) {
        if (digitalRead(BTN_MODAL) == LOW) break;
        delay(10);
    }
    if (digitalRead(BTN_MODAL) == LOW) break;
}

```

Explicación: el patrón de arreglos paralelos es simple y potente: `poses[i]` guarda DONDE ir y `tiempos[i]` guarda CUANTO esperar ahí; el mismo índice `i` los une. Para elegir la pausa se reutiliza la perilla de la pinza dividida en tres tercios, una interfaz minimalista pero funcional: antes de presionar GRABAR, pones la perilla de la pinza en la zona de la pausa deseada (la pinza quedara en la posición correspondiente a esa zona, así que graba primero la pose con la pinza donde la quieres y luego ajusta solo un poco la perilla a la zona de tiempo, o acepta el pequeño ajuste). La espera usa `millis()` en lugar de `delay()` para que el botón de modo siga respondiendo durante la pausa: con `delay()` el programa quedaría "sordo" hasta 1.2 segundos. Este es un primer paso hacia la programación no bloqueante, una técnica que usarás mucho en proyectos avanzados.

Ejercicio 5: diseñar una coreografía

Enunciado: programa una coreografía completa de 30 segundos para tu brazo, como si fuera a presentarse en un concurso de talentos de robots. Debe incluir: una reverencia inicial (inclinarse hacia adelante), un saludo con la "mano", un giro completo de la base mostrando la

pinza abierta y cerrada, y una reverencia final. Cada fase debe tener su velocidad: la reverencia lenta y elegante, el saludo rápido y alegre. No se vale usar potenciómetros: todo programado.

Objetivo: integrar todo lo aprendido (poses, secuencias, movimiento suave, velocidades distintas por fase) en un proyecto creativo donde tu decides la estética del movimiento, como un coreógrafo.

Solución del ejercicio 5

La coreografía se organiza en fases, cada una, una función separada. Eso hace el código legible y te permite ensayar cada fase por separado antes del estreno.

```
#include <Servo.h>

const int PIN_BASE = 3, PIN_HOMBRO = 5, PIN_CODO = 6, PIN_PINZA = 9;
const int ABIERTA = 90, CERRADA = 25;

Servo srvBase, srvHombro, srvCodo, srvPinza;
int posBase = 90, posHombro = 90, posCodo = 90, posPinza = 60;

void moverSuave(Servo &s, int &posActual, int destino, int espera) {
    if (destino > posActual) {
        for (int a = posActual; a <= destino; a++) { s.write(a); delay(espera); }
    } else {
        for (int a = posActual; a >= destino; a--) { s.write(a); delay(espera); }
    }
    posActual = destino;
}

void irAPose(int b, int h, int c, int p, int espera) {
    moverSuave(srvBase, posBase, b, espera);
    moverSuave(srvHombro, posHombro, h, espera);
    moverSuave(srvCodo, posCodo, c, espera);
    moverSuave(srvPinza, posPinza, p, espera);
}

void setup() {
    srvBase.attach(PIN_BASE);
    srvHombro.attach(PIN_HOMBRO);
    srvCodo.attach(PIN_CODO);
    srvPinza.attach(PIN_PINZA);
    irAPose(90, 90, 90, 60, 15);
    delay(1500);
}
```

```

}

// Fase 1: reverencia lenta y elegante
void reverencia() {
    irAPose(90, 55, 45, CERRADA, 28); // inclinarse hacia adelante, lento
    delay(1200);                       // mantener la reverencia
    irAPose(90, 110, 90, CERRADA, 28); // enderezarse, lento
    delay(600);
}

// Fase 2: saludo rapido y alegre con la "mano"
void saludo() {
    irAPose(90, 140, 130, ABIERTA, 10);
    delay(300);
    for (int i = 0; i < 3; i++) {      // vaiven tres veces
        irAPose(90, 140, 100, ABIERTA, 8);
        irAPose(90, 140, 155, ABIERTA, 8);
    }
    delay(400);
}

// Fase 3: giro completo mostrando la pinza
void giroConPinza() {
    irAPose(90, 120, 80, ABIERTA, 14);
    delay(300);
    irAPose(30, 120, 80, ABIERTA, 14); // girar a un lado
    irAPose(30, 120, 80, CERRADA, 14); // cerrar pinza
    delay(300);
    irAPose(150, 120, 80, CERRADA, 14); // girar al otro lado
    irAPose(150, 120, 80, ABIERTA, 14); // abrir pinza
    delay(300);
    irAPose(90, 120, 80, ABIERTA, 14); // volver al centro
    delay(300);
}

void loop() {
    reverencia(); // fase 1: lenta
    saludo();     // fase 2: rapida
    giroConPinza(); // fase 3: media
    reverencia(); // fase 4: reverencia final
    irAPose(90, 90, 90, 60, 15);
    delay(5000); // aplausos imaginarios antes de repetir
}

```

Explicación: cada fase es una función con su propia velocidad (28 ms para la reverencia elegante, 8-10 ms para el saludo energetico, 14 ms para el giro). Esa diferencia de velocidades es lo que le da carácter a la coreografía: el mismo brazo se ve solemne o jugueton según la velocidad. La reverencia se logra bajando el hombro y plegando el codo hacia adelante, como una persona inclinando el torso. El saludo reutiliza el vaiven del ejemplo B pero con la pinza abierta, como una mano saludando. El giro muestra la pinza abriendo y cerrando en los extremos, un gesto de "miren lo que puedo hacer". El loop() ordena las fases como un programa de mano: reverencia, saludo, giro, reverencia final. Prueba a agregar tu propia fase: un "baile" moviendo la base de lado a lado mientras el codo sube y baja, o un final dramático donde el brazo se pliega lentamente hasta "dormirse". La robótica también es expresión.

Resumen de la unidad

En esta unidad construiste y programaste un brazo robótico de 4 grados de libertad, y en el camino aprendiste los conceptos fundamentales de la robótica articulada. Un brazo robótico es una cadena de articulaciones (servos) unidas por eslabones rígidos, que termina en un efector final; en tu caso, una pinza de dos dedos. Cada articulacion aporta un grado de libertad: la base gira el conjunto, el hombro levanta, el codo dobla y la pinza agarra. Con esos cuatro movimientos puedes tomar un objeto, levantarlo, girar y soltarlo en otro lugar, que es la tarea básica de los brazos industriales.

Aprendiste cinemática directa de forma intuitiva: la posición de la pinza es el resultado acumulado de todos los ángulos, las articulaciones cercanas a la base producen movimientos grandes y las lejanas ajustes finos, y la gravedad es un actor principal que castiga las posiciones extendidas. Montaste la mecánica de la base a la pinza centrando cada servo antes de fijarlo, y resolviste el tema eléctrico mas importante: los servos se alimentan con una fuente externa de 5V y 2A mínimo, con el GND común entre la fuente y el Arduino, porque el Arduino solo no puede con la corriente que piden cuatro servos.

En programación dominaste tres niveles de control. Primero, el control manual con potenciómetros: un potenciómetro por articulacion, map() para convertir 0-1023 a ángulos, límites de seguridad por articulacion y un umbral anti-temblor. Segundo, los movimientos programados: poses guardadas en arreglos, la función moverSuave() para transiciones fluidas grado por grado, y la función irAPose() para expresar rutinas como listas legibles de poses. Tercero, la técnica de grabar y repetir: mover el brazo a mano, guardar poses con un botón, y reproducirlas en ciclo, el mismo principio con el que se programan los robots de fábrica.

También te llevas una colección de buenas prácticas: calibrar centros y límites de cada servo con tu montaje real, probar a velocidad baja antes de acelerar, diseñar rutinas que descansen en posiciones plegadas, dejar holgura en los cables, y anotar tus poses útiles. Y conoces los errores típicos (servos calientes, brazo que se vence, reinicios por falta de corriente, temblores por ruido) con sus soluciones.

Lo mas importante: comprobaste que los principios son los mismos del brazo pequeño al brazo industrial. Grados de libertad, cinematica, efector final, alimentación adecuada y secuencias programadas: ese vocabulario te acompañara en cada robot articulado que construyas. En la siguiente unidad, el brazo dejará de ser sordo y ciego: le agregaremos sensores para que reaccione al mundo. Por ahora, deja tu brazo haciendo su coreografía en ciclo: se lo gano.

UNIDAD 10: COMUNICACIÓN INALÁMBRICA Y PROYECTO FINAL

Bienvenido a la última unidad de este curso de robótica. Si llegaste hasta aquí, ya sabes leer sensores, mover motores, esquivar obstáculos y programar comportamientos. Eso es muchísimo. Pero hay algo que cambia por completo la experiencia de construir un robot: controlarlo a distancia, sin cables, desde tu celular. Imagínate manejar tu carrito robot desde el sofa, hacerlo girar con un dedo, activar una bocina, cambiar entre modo autónomo y modo manual como si fuera un videojuego. Eso es lo que vas a lograr en esta unidad.

Además, esta unidad es especial por otro motivo: aquí no solo vas a aprender un tema nuevo, la comunicación inalámbrica con el módulo Bluetooth HC-05, sino que también vas a cerrar el curso con un proyecto final integrador. Un proyecto integrador es aquel que junta todo lo que aprendiste en las unidades anteriores en una sola máquina funcionando. Tu robot final va a tener dos cerebros en uno: un modo autónomo donde el robot se mueve solo esquivando obstáculos, y un modo manual donde tu lo controlas por Bluetooth desde tu teléfono. Poder cambiar entre esos modos, probar cada parte por separado y luego ver el sistema completo funcionando es la mejor forma de demostrarte a ti mismo que ya eres capaz de construir robots de verdad.

Pensemos primero en por qué la comunicación inalámbrica es tan importante. En las primeras unidades, todo tu robot dependía del cable USB: lo conectabas a la computadora, le subías el programa y luego, para moverlo, tenías que desconectarlo y dejar que funcionará solo. Si querías cambiar algo, vuelta a conectar el cable. Esa es la vida normal de un robot autónomo y esta bien. Pero hay robots que necesitan un piloto humano: drones de carreras, brazos robóticos industriales que un operador mueve con un control, sondas submarinas que un científico dirige desde la superficie. En todos esos casos, el robot y la persona se comunican sin cables. La tecnología más sencilla y barata para lograr eso en proyectos de principiantes es el Bluetooth, y el módulo HC-05 es el caballo de batalla de miles de proyectos en todo el mundo. Cuesta poco, se consigue fácil y se conecta al Arduino como si fuera un puerto serie más.

El camino del cable al Bluetooth es más corto de lo que crees. Todo empieza con la comunicación serial, que es la forma en que el Arduino habla con tu computadora a través del cable USB. El monitor serie del IDE de Arduino te permite enviar letras y números al Arduino y ver lo que el Arduino te responde. Si logras que tu robot obedezca cuando escribes "A" en el monitor serie, ya tienes el 80 por ciento del trabajo hecho para controlarlo por Bluetooth, porque el HC-05 se comporta exactamente igual: lo que escribas en la app del celular llega al

Arduino como si lo hubieras escrito en el monitor serie. Esa es la idea clave de esta unidad: primero dominas los comandos por cable con el monitor serie, luego cambias el cable por el módulo Bluetooth y todo sigue funcionando igual.

El proyecto final que vas a construir es un robot móvil de dos ruedas, del tipo carrito que ya armaste en unidades anteriores, pero mejorado. Sus componentes principales son: un chasis con dos motores de corriente directa controlados por un puente H (como el L298N), un sensor ultrasónico HC-SR04 al frente para detectar obstáculos, un módulo Bluetooth HC-05 para recibir tus órdenes, y opcionalmente una bocina o buzzer para efectos de sonido y algunos LEDs para senalar el modo activo. El comportamiento es el siguiente: al encenderlo, el robot arranca en modo autónomo y avanza esquivando obstáculos usando el sensor ultrasónico, tal como lo programaste en la unidad de sensores. En cualquier momento, tu puedes enviarle un comando por Bluetooth para pasarlo a modo manual, y entonces el robot se detiene y espera tus órdenes: adelante, atrás, izquierda, derecha, alto, y control de velocidad. Otro comando lo regresa al modo autónomo. Así de simple y así de poderoso.

Como usar esta unidad como guía de proyecto es algo que quiero explicarte con calma, porque aquí el orden de trabajo importa tanto como el código. No intentes armar todo de golpe. La estrategia ganadora es trabajar por módulos: primero el módulo de comunicación serial, luego el módulo Bluetooth, luego el módulo de movimiento, luego el módulo del sensor, y al final la integración de todo en el programa del proyecto final. Cada módulo se prueba por separado antes de conectarlo con los demás. Si algo falla, ya sabes en que módulo esta el problema. Esta forma de trabajar se llama depuración sistemática y es exactamente lo que hacen los ingenieros profesionales, solo que en proyectos mas grandes. Al final de la unidad encontrarás un plan de pruebas paso a paso para que no te pierdas.

También vamos a hablar de temas prácticos que suelen ignorarse en los tutoriales y que después causan dolores de cabeza: como alimentar tu robot móvil con baterías (porque un robot con cable no es un robot móvil), como calcular aproximadamente cuanta autonomía te dan tus baterías, que cuidados de mantenimiento necesita tu robot para durar, y las reglas de seguridad que debes seguir para no dañar tus componentes ni lastimarte. Y para cerrar con broche de oro, te voy a contar que sigue después de este curso: ROS, el sistema operativo de robots que usan las universidades y la industria; el ESP32, un microcontrolador con WiFi y Bluetooth integrados que es el siguiente paso natural después del Arduino Uno; y la impresión 3D, para que diseñes tus propios chasis y piezas.

Una advertencia honesta antes de empezar: el módulo Bluetooth a veces se porta caprichoso la primera vez que lo configuras. El emparejamiento puede fallar una o dos veces, la app puede no encontrar el módulo, o los comandos pueden llegar pero el robot no responde. Todo eso es normal y tiene solución, y en la sección de depuración vamos a cubrir cada uno de

esos problemas. La paciencia es parte del kit de herramientas del robótico. Si algo no funciona a la primera, respira, revisa las conexiones una por una y vuelve a intentarlo. El momento en que tu carrito avanza por primera vez obedeciendo a tu dedo en la pantalla del celular vale cada minuto de frustración previa.

Para aprovechar esta unidad necesitas tener a la mano lo siguiente: tu Arduino Uno, el carrito robot con sus motores y puente H ya armado de unidades anteriores, el sensor ultrasónico HC-SR04, un módulo Bluetooth HC-05, un protoboard, cables jumper, dos resistencias para el divisor de voltaje (una de 1k ohm y una de 2k ohm, o valores cercanos), baterías o un power bank para alimentar el robot, y un celular Android con alguna app gratuita de control Bluetooth (te recomendaré opciones). Si no tienes el HC-05 todavía, puedes hacer toda la primera parte de la unidad, la de comunicación serial con el monitor, sin el módulo, y agregarlo cuando lo consigas. El código está diseñado para que el cambio sea mínimo.

Vamos por partes. Primero entenderás a fondo la comunicación serial y el parseo de comandos. Después conocerás el HC-05 por dentro y por fuera: sus pines, su voltaje, su velocidad y como emparejarlo. Luego diseñaras tu protocolo de comandos, como un lenguaje secreto entre tu celular y tu robot. Y finalmente planearas, armaras, probaras y mejoraras tu proyecto final. Al terminar esta unidad no solo sabrás de Bluetooth: habrás construido, probado y documentado un robot completo, que es exactamente lo que se espera de alguien que termina un curso de robótica. Manos a la obra.

Explicación detallada

Comunicación serial: el lenguaje del Arduino

La comunicación serial es la forma más básica en que el Arduino intercambia información con el mundo exterior. Serial significa que los datos viajan bit por bit, uno detrás de otro, por un solo cable (más la tierra de referencia). Es como mandar un mensaje letra por letra en lugar de mandar toda la página de un golpe. El Arduino Uno tiene un puerto serie por hardware en los pines 0 (RX) y 1 (TX), y además ese mismo puerto está conectado al chip que convierte USB a serial, que es por donde hablas con tu computadora cuando usas el monitor serie.

Para usar la comunicación serial en tu programa necesitas tres instrucciones fundamentales. La primera es `Serial.begin(9600)`, que se pone en el setup y abre el puerto serie a una velocidad de 9600 baudios. Un baudio es, en la práctica, un bit por segundo, así que 9600 baudios significa que se transmiten aproximadamente 9600 bits por segundo. Esa velocidad es el estándar para hablar con el monitor serie y con el módulo HC-05, así que memorízala: 9600. Si el Arduino y el dispositivo del otro lado no usan la misma velocidad, los datos llegan como

basura ilegible. Es como si dos personas intentaran hablar y una hablara rapidísimo y la otra lentísimo: no se entenderían.

La segunda instrucción es `Serial.available()`, que te dice cuántos bytes (caracteres) han llegado y están esperando a ser leídos. Devuelve 0 si no ha llegado nada. Se usa casi siempre dentro de un `if`: si hay datos disponibles, entonces los leo. La tercera es `Serial.read()`, que lee un solo byte del búfer de entrada y lo devuelve como un número (su código ASCII). Por ejemplo, si envías la letra "A" desde el monitor serie, `Serial.read()` devuelve 65, que es el código ASCII de la "A" mayúscula. En la práctica casi siempre convertimos ese valor a `char` para compararlo con letras: `char comando = Serial.read();` y luego `if (comando == 'A')`.

El monitor serie es la ventana del IDE de Arduino (el icono de la lupa en la esquina superior derecha) que te permite ver lo que el Arduino imprime con `Serial.print()` o `Serial.println()`, y también enviarle texto con la barra superior. Cuando escribes algo ahí y presionas Enter o el botón Enviar, ese texto viaja por el cable USB hasta el Arduino. Hay un detalle importante en la configuración del monitor serie: el menú de "fin de línea". Puedes elegir "Sin ajuste de línea", "Nueva línea", "Retorno de carro" o "Ambos NL y CR". Esto determina que caracteres extra se envíen después de tu texto. Si tu programa espera comandos de una sola letra, lo más cómodo es usar "Sin ajuste de línea", porque así llega exactamente lo que escribiste, sin caracteres extra que podrían confundir tu parseo. Si usas "Nueva línea", después de tu letra llegará también un carácter de salto de línea, y tu programa debería ignorarlo.

El parseo simple de comandos es la técnica de interpretar lo que llega por el puerto serie. La versión más sencilla, y la que usaremos en este curso, es el comando de una sola letra. Cada letra significa una acción: "A" es adelante, "T" es atrás, "I" es izquierda, "D" es derecha, "S" es alto (stop). Cuando llega un carácter, el programa lo compara con cada comando conocido usando `if` o `switch`, y ejecuta la acción correspondiente. Si llega algo desconocido, se ignora. Este esquema es fácil de depurar porque puedes probar cada comando a mano desde el monitor serie, y es exactamente el mismo esquema que usan la mayoría de las apps de control Bluetooth: cada botón de la app envía una letra. Por eso primero lo dominamos con cable y luego lo usamos sin cable.

Un ejemplo mental del flujo: en el `loop()` del Arduino pones `if (Serial.available() > 0)`, dentro lees el carácter, y con una estructura `switch` decides qué hacer. Si el carácter es 'A', enciendes los motores hacia adelante; si es 'S', los apagas; si es 'V', aumentas la velocidad; si es 'v', la disminuyes. Así de directo. Más adelante veremos que también puedes enviar números o comandos de varias letras, pero para nuestro proyecto final el protocolo de una letra es suficiente, rápido y confiable.

El módulo Bluetooth HC-05

El HC-05 es un pequeño módulo azul con una antena que le permite al Arduino comunicarse por Bluetooth con celulares, computadoras y otros módulos. Por dentro es un chip de radio Bluetooth mas un microcontrolador que traduce entre Bluetooth y comunicación serial. Para tu Arduino, el HC-05 es transparente: todo lo que el módulo recibe por Bluetooth lo entrega al Arduino por sus pines TX y RX como si fuera comunicación serial normal, y todo lo que el Arduino le envía por serial el módulo lo transmite por Bluetooth. Por eso el código que escribas para el monitor serie funciona igual con el HC-05 sin cambios.

El HC-05 puede trabajar en dos modos: maestro y esclavo. En modo esclavo, el módulo espera a que otro dispositivo (tu celular) se conecte a el; en modo maestro, el módulo busca y se conecta a otro dispositivo. Para nuestro proyecto el modo esclavo es el que necesitamos, y es el modo en que el HC-05 viene configurado de fábrica, así que normalmente no tienes que cambiar nada. Tu celular verá al HC-05 como un dispositivo Bluetooth mas, te pedirá un PIN (casi siempre es 1234 o 0000) y quedara emparejado.

Los pines del HC-05 son seis, aunque solo usaremos cuatro: VCC, GND, TXD y RXD. VCC se conecta a 5V del Arduino, GND a tierra, y aquí viene la parte delicada: TXD del módulo se conecta al RX del Arduino, y RXD del módulo se conecta al TX del Arduino. Es decir, van cruzados: el que transmite de un lado se conecta al que recibe del otro. Tiene lógica si lo piensas: para que dos personas hablen, la boca de una debe conectarse al oído de la otra. Si conectas TX con TX, seria como conectar dos bocas: nadie escucha. Este es el error número uno de los principiantes con el HC-05, así que grábatelo: TX va con RX y RX va con TX.

Ahora el tema del voltaje, que es critico. El Arduino Uno trabaja con niveles logicos de 5V: un "1" lógico son 5 voltios. El HC-05, en cambio, tiene su pin RXD diseñado para 3.3V. Si conectas directamente el TX del Arduino (que sale a 5V) al RXD del módulo (que espera 3.3V), con el tiempo puedes dañar el módulo. Mucha gente lo conecta directo y "funciona", pero es una loteria: algunos módulos lo aguantan y otros se queman a las pocas horas. La solución profesional y barata es un divisor de voltaje con dos resistencias.

Un divisor de voltaje es un truco muy simple: pones dos resistencias en serie entre la señal de 5V y tierra, y tomas la señal reducida del punto medio. La fórmula es $V_{salida} = V_{entrada} * (R2 / (R1 + R2))$, donde R1 es la resistencia conectada a los 5V y R2 la conectada a tierra. Si usas $R1 = 1k\ \Omega$ y $R2 = 2k\ \Omega$, la salida es $5 * (2000 / 3000) = 3.33V$, que es perfecto para el RXD del módulo. En la práctica conectas así: el TX del Arduino va a un extremo de la resistencia de 1k, el otro extremo de la 1k se une con un extremo de la 2k y ese punto medio va al RXD del HC-05, y el otro extremo de la 2k va a tierra (GND). El pin TXD del módulo, en cambio, se conecta directo al RX del Arduino, porque el Arduino lee sin problema los 3.3V del módulo como un "1" lógico. Solo el camino Arduino-hacia-módulo necesita el divisor.

La velocidad de comunicación del HC-05 de fábrica es 9600 baudios, igual que la que usamos en `Serial.begin(9600)`. Esa coincidencia no es casualidad: está pensada para que conectes y funcione. Si alguna vez cambias la velocidad del módulo con comandos AT (los comandos de configuración del HC-05), tendrás que cambiar también el `Serial.begin` para que coincidan. Para este curso, déjalo en 9600 y no toques la configuración.

El emparejamiento con el celular se hace así: alimenta el HC-05 (conectado al Arduino o solo con 5V y GND), espera a que su LED parpadee rápido (eso significa que está visible y esperando conexión), abre los ajustes de Bluetooth de tu celular, busca dispositivos y debería aparecer uno llamado "HC-05". Tócalo, escribe el PIN 1234 (o 0000 si el primero no funciona) y listo. Cuando el módulo se conecta de verdad, su LED cambia el ritmo de parpadeo: parpadea lento, como dos veces por segundo, o en algunas versiones se queda fijo. Ese cambio de parpadeo es tu confirmación visual de que hay conexión. Después abres la app de control, la conectas al HC-05 y empiezas a mandar comandos.

Apps de control y protocolo de comandos

Para controlar tu robot desde el celular necesitas una app que se conecte por Bluetooth y envíe caracteres cuando tocas botones. Hay varias gratuitas en la Play Store. Las más usadas en proyectos de Arduino son "Bluetooth RC Controller", "Arduino Bluetooth Control" y "BT Controller". Casi todas funcionan igual: las abres, eliges el dispositivo HC-05 emparejado, y te muestran una pantalla con botones de flechas y botones extra. Cada botón está configurado para enviar una letra o un número cuando lo presionas. Lo importante es que la app te permita ver o configurar qué carácter envía cada botón, porque tu programa de Arduino debe esperar exactamente esos caracteres.

Nuestro protocolo de comandos, es decir, el acuerdo sobre qué letra significa qué acción, va a ser el siguiente. Lo definimos nosotros y lo programamos igual en el Arduino. Aquí está la tabla completa:

Comando	Accion
-----	-----
A	Avanzar (motores adelante)
T	Retroceder (motores atras)
I	Girar a la izquierda
D	Girar a la derecha
S	Detenerse (apagar motores)
V	Aumentar velocidad
v	Disminuir velocidad
B	Tocar bocina / buzzer
M	Cambiar modo (autonomo <-> manual)
L	Encender / apagar luces LED

Fíjate que usamos letras fáciles de recordar en español: A de Adelante, T de atrás, I de Izquierda, D de Derecha, S de Stop. La V mayuscula sube la velocidad y la v minúscula la baja. La M cambia el modo del robot, que es la estrella de nuestro proyecto final. Este protocolo es nuestro "lenguaje secreto": mientras la app envíe esas letras y el Arduino las interprete igual, todo funciona. Si usas una app cuyos botones envían otras letras, tienes dos opciones: reconfigurar los botones en la app (si lo permite) o cambiar las letras en tu código de Arduino. Yo te recomiendo lo segundo cuando sea necesario, porque así mantienes tu protocolo documentado en un solo lugar: tu código.

Planificacion del proyecto final

Un proyecto final no se improvisa: se planea. La planificacion tiene cinco pasos: definir el objetivo, hacer la lista de materiales, dibujar el diagrama de conexiones, escribir el código por módulos y armar un plan de pruebas. Vamos uno por uno aplicados a nuestro robot.

El objetivo se escribe en una frase clara y medible: "Construir un robot móvil con dos modos de operación: modo autónomo evasor de obstáculos usando sensor ultrasónico, y modo manual controlado por Bluetooth desde el celular, con cambio de modo por comando, que funcione al menos 20 minutos con baterías y que no se caiga a pedazos al chocar". Esa frase te sirve de contrato contigo mismo: cuando el robot haga todo eso, el proyecto esta terminado. Todo lo extra (bocina, luces, telemetria) es decoración bienvenida pero no obligatoria.

La lista de materiales es tu lista de compras y de verificación. Para el proyecto final necesitas: 1 Arduino Uno, 1 chasis de carrito con 2 motores DC y rueda loca, 1 driver de motores L298N (o L293D), 1 sensor ultrasónico HC-SR04 con su base, 1 módulo Bluetooth HC-05, 1 protoboard pequeña o placa de conexiones, cables jumper macho-hembra y macho-macho, 2 resistencias (1k y 2k ohm) para el divisor de voltaje, 1 buzzer activo de 5V, 2 LEDs con sus resistencias de 220 ohm, 1 portabaterias para 6 pilas AA o un power bank con cable USB, 1

interruptor para encendido y apagado, cinta aislante y cinchos para fijar todo al chasis. Revisa esta lista antes de empezar cada sesión de trabajo: nada frena mas un proyecto que descubrir a mitad del armado que te falta un cable.

El diagrama de conexiones es el mapa de tu robot. Dibujalo en papel antes de conectar nada. Debe mostrar cada módulo, cada pin del Arduino que usa y a donde va cada cable. Para nuestro proyecto el mapa es: motores al L298N (salidas OUT1-OUT4), L298N al Arduino (pines digitales para IN1-IN4 y ENA/ENB para velocidad por PWM), HC-SR04 al Arduino (Trig y Echo), HC-05 al Arduino (TX-RX cruzados, con divisor de voltaje en el RX del módulo), buzzer a un pin digital, LEDs a pines digitales con resistencia, y la alimentación: baterías al L298N para los motores y al Arduino por su jack o pin VIN. Dibujar primero y conectar después te ahorra el 90 por ciento de los errores de cableado.

El código por módulos significa escribir el programa como piezas separadas que luego se unen. Nuestros módulos de código son: módulo de movimiento (funciones adelante(), atrás(), izquierda(), derecha(), detener(), setVelocidad()), módulo de sensor (función medirDistancia()), módulo Bluetooth (función leerComando() que devuelve el carácter recibido o 0 si no hay nada), módulo de modo autónomo (función modoAutonomo() que decide que hacer según la distancia), módulo de modo manual (función modoManual(comando) que ejecuta el comando recibido), y el programa principal que enciende todo, lee comandos y decide que modo esta activo. Cada módulo se puede probar solo, y eso nos lleva al último paso.

El plan de pruebas es la lista ordenada de verificaciones, del módulo mas simple al sistema completo. Por ejemplo: prueba 1, el Arduino enciende y el monitor serie muestra mensajes; prueba 2, los motores giran en ambas direcciones con un programa de prueba; prueba 3, el sensor mide distancias correctas mostradas en el monitor; prueba 4, los comandos por monitor serie mueven el robot; prueba 5, el HC-05 se empareja y los comandos llegan desde el celular; prueba 6, el modo autónomo esquiva obstáculos; prueba 7, el cambio de modo funciona en ambas direcciones; prueba 8, prueba de autonomía con baterías y recorrido completo. Si una prueba falla, no avanzas a la siguiente hasta resolverla. Así nunca te encuentras con un robot que "no hace nada" sin saber por donde empezar a buscar el problema.

Depuración sistematica

Depurar es el arte de encontrar por que algo no funciona, y la depuración sistematica es hacerlo con método en lugar de mover cables al azar esperando un milagro. La regla de oro es: prueba por módulos, del simple al complejo. Si el robot no responde al Bluetooth, no revises todo el robot: primero verifica que el HC-05 este emparejado (LED parpadeando lento), luego verifica que los comandos lleguen al Arduino (haz que el Arduino reenvie por el monitor serie lo que recibe), luego verifica que el parseo funcione (enciende un LED con cada

comando antes de mover motores), y solo al final revisa los motores. Cada paso que funciona descarta una familia entera de problemas.

Los problemas típicos del HC-05 y sus soluciones: si el celular no encuentra el módulo, verifica que este alimentado (LED parpadeando rápido) y que no este ya conectado a otro dispositivo; el Bluetooth solo permite una conexión a la vez. Si pide PIN y el 1234 no funciona, prueba 0000. Si se empareja pero la app no se conecta, cierra y reabre la app, y verifica que elegiste el HC-05 correcto en la lista. Si los comandos llegan pero el robot hace cosas raras, casi siempre es que la app envía caracteres extra (como saltos de línea) o letras distintas a las de tu protocolo: haz que el Arduino imprima por el monitor serie cada carácter que recibe, con su código ASCII, y compara con tu tabla. Si el módulo se desconecta solo, suele ser falta de alimentación: el HC-05 consume picos de corriente al transmitir y si lo alimentas desde un Arduino con muchas cosas conectadas, el voltaje cae; alimentalo con una fuente mas robusta o reduce la carga.

Para depurar el código usa `Serial.println()` como tu mejor amigo: imprime el modo activo, la distancia medida, el comando recibido y la velocidad actual. Esos mensajes te cuentan la historia de lo que el robot "piensa". Cuando todo funcione, puedes dejar algunos mensajes o quitarlos para que el programa corra mas limpio. Y un consejo de oro: cuando cambies algo y deje de funcionar, lo primero que debes revisar es lo último que cambiaste. El 80 por ciento de los errores nuevos los introduce el último cambio.

Fuentes de poder para robots móviles

Un robot móvil que depende del cable USB no es móvil de verdad. Necesitas baterías, y elegir bien la alimentación es tan importante como el código. El Arduino Uno puede alimentarse por su jack de corriente con 7 a 12V (el regulador interno los baja a 5V), o por el pin VIN con el mismo rango, o por USB con 5V. Los motores, en cambio, se alimentan aparte a través del driver L298N, que acepta de 5V a 35V para los motores según el modelo. La configuración típica de principiantes es: un portabaterías de 6 pilas AA (9V en total si son alcalinas, 7.2V si son recargables) conectado a la entrada de 12V del L298N, y del L298N se toma 5V para el Arduino, o se alimenta el Arduino por separado. Ojo: los motores generan ruido eléctrico que puede reiniciar el Arduino si comparten la misma fuente sin cuidado; por eso muchos proyectos usan dos fuentes: una para motores y otra para la electrónica. Con un power bank USB para el Arduino y las pilas para los motores tienes un sistema robusto y fácil.

El calculo aproximado de autonomía se hace con la capacidad de la batería y el consumo. La capacidad se mide en miliamperios-hora (mAh): una pila AA alcalina tiene unos 2000 mAh, una batería de 9V pequeña unos 500 mAh. El consumo es la suma de lo que gasta cada cosa: el Arduino unos 50 mA, el HC-05 unos 30 mA transmitiendo, el sensor ultrasónico 15 mA, y

cada motor entre 100 y 250 mA trabajando (mas al arrancar). Un robot típico consume entre 300 y 600 mA en movimiento. La formula es: autonomía en horas = capacidad (mAh) / consumo (mA). Ejemplo: 6 pilas AA de 2000 mAh alimentando un robot que consume 400 mA en promedio dan $2000 / 400 = 5$ horas en teoría. En la práctica, con arranques, giros y el voltaje bajando, espera entre la mitad y dos tercios de eso: unas 2 a 3 horas reales. Para nuestro objetivo de 20 minutos de operación continua, unas pilas AA van sobradas. Evita las baterías cuadradas de 9V para los motores: tienen poca capacidad y se agotan rapidísimo con motores.

Consejos prácticos de energía: usa pilas recargables si vas a probar mucho (salen mas baratas a la larga), pon un interruptor entre las baterías y el circuito para no estar conectando y desconectando cables, nunca alimentes los motores desde el pin 5V del Arduino (el regulador se sobrecalienta y se apaga), y si el robot se comporta raro solo cuando usa baterías pero bien con USB, sospecha de baterías bajas: mide el voltaje con multímetro y cambia las pilas. Un robot "loco" que se reinicia solo casi siempre tiene hambre de energía.

Mantenimiento y seguridad

Un robot bien mantenido dura años; uno descuidado muere en semanas. El mantenimiento básico incluye: revisar periódicamente que los cables jumper sigan firmes (con la vibración se aflojan), limpiar el polvo del chasis y de los sensores, verificar que las ruedas no tengan cabellos o hilos enredados en los ejes, recargar o cambiar las pilas antes de cada sesión de pruebas importantes, y guardar el robot en un lugar seco. El sensor ultrasónico es delicado: no lo toques por los cilindros metálicos y no lo expongas a golpes. Revisa también que los tornillos del chasis sigan apretados.

La seguridad no es opcional. Reglas básicas: nunca conectes o desconectes cables con el circuito energizado; apaga siempre antes de modificar el cableado. No pongas en corto las baterías (un cable pelado tocando ambos polos se calienta al rojo vivo en segundos). Si algo huele a quemado o sale humo, desconecta la alimentación de inmediato y no la reconectes hasta encontrar la causa. No mires de frente al sensor ultrasónico a corta distancia de forma prolongada (no es peligroso en dosis normales, pero no hay necesidad). Prueba el robot en un área despejada, lejos de escaleras, mascotas pequeñas y objetos frágiles: un carrito a toda velocidad puede tirar cosas o caerse. Y si trabajas con niños, un adulto debe supervisar el manejo de baterías y herramientas.

Que sigue después de este curso

Terminar este curso te pone en un gran punto de partida, no en la meta final. El mundo de la robótica es enorme y estos son los siguientes pasos naturales. ROS (Robot Operating System) es el estándar en universidades e industria para programar robots complejos: es un conjunto de herramientas y librerías que corre en Linux y permite que sensores, motores y algoritmos se comuniquen entre sí. Aprender ROS es un salto grande, pero con la base de este curso lo entenderás mucho mejor. El ESP32 es el microcontrolador que sigue después del Arduino Uno: tiene WiFi y Bluetooth integrados (adiós módulo externo), es mas rápido, mas barato y se programa con el mismo IDE de Arduino. Muchos proyectos que hoy haces con Uno + HC-05 mañana los harás con un solo ESP32. Y la impresión 3D te permite diseñar tus propios chasis, soportes para sensores y piezas personalizadas: tu robot dejará de verse como un kit genérico y empezara a verse como tu diseño. Con electrónica, programación y diseño 3D en tus manos, no hay robot que no puedas imaginar y construir.

Ejemplos paso a paso

Ejemplo A: encender un LED con comandos del monitor serie

Este es el "hola mundo" de la comunicación serial. Vas a controlar un LED conectado al pin 13 (o el LED integrado de la placa, que también esta en el pin 13) escribiendo "1" para encenderlo y "0" para apagarlo en el monitor serie. No necesitas nada mas que tu Arduino y el cable USB.

El circuito es trivial: si usas el LED integrado no conectas nada. Si usas un LED externo, conecta su pata larga (anodo) al pin 13 a través de una resistencia de 220 ohm, y su pata corta (catodo) a GND. La resistencia protege al LED de quemarse con demasiada corriente.

El programa abre el puerto serie a 9600 baudios, configura el pin 13 como salida y luego en el loop revisa si llegó algún carácter. Si llegó "1", enciende el LED; si llegó "0", lo apaga; cualquier otra cosa la ignora. Además imprime mensajes de confirmación para que veas en el monitor lo que el Arduino entendió.

```
// Ejemplo A: control de LED por monitor serie
// Comandos: '1' enciende, '0' apaga

const int pinLED = 13;

void setup() {
  pinMode(pinLED, OUTPUT);
  Serial.begin(9600);          // abrimos el puerto serie a 9600 baudios
  Serial.println("Listo. Escribe 1 para encender, 0 para apagar.");
}

void loop() {
  if (Serial.available() > 0) { // llego algun caracter?
    char comando = Serial.read(); // lo leemos
    if (comando == '1') {
      digitalWrite(pinLED, HIGH);
      Serial.println("LED encendido");
    }
    else if (comando == '0') {
      digitalWrite(pinLED, LOW);
      Serial.println("LED apagado");
    }
    // cualquier otro caracter se ignora
  }
}
```

Para probarlo: sube el programa, abre el monitor serie, pon la velocidad en 9600 baudios y el fin de línea en "Sin ajuste de línea", escribe 1 y presiona Enviar. El LED debe encenderse y el monitor debe mostrar "LED encendido". Escribe 0 y se apaga. Si no pasa nada, revisa que la velocidad del monitor serie sea 9600 y que hayas seleccionado el puerto correcto. Este programita es la semilla de todo lo demás: el parseo de comandos ya esta funcionando.

Ejemplo B: configurar y probar el HC-05

Ahora vamos a conectar el módulo Bluetooth HC-05 al Arduino. Este ejemplo no mueve motores todavía: su unico objetivo es verificar que el módulo se empareja con tu celular y que los caracteres viajan en ambas direcciones. Es la prueba de comunicación pura, y es importante hacerla antes de conectar los motores, porque así sabes que cualquier problema futuro no es del Bluetooth.

Las conexiones paso a paso son las siguientes. Primero, sin energía en nada, conecta GND del HC-05 a GND del Arduino. Segundo, conecta VCC del HC-05 a 5V del Arduino. Tercero,

conecta TXD del HC-05 directamente al pin 10 del Arduino (vamos a usar el pin 10 como RX por software, para no pelear con el USB). Cuarto, arma el divisor de voltaje: conecta el pin 11 del Arduino a un extremo de la resistencia de 1k ohm; el otro extremo de la 1k unelo con un extremo de la resistencia de 2k ohm y lleva ese punto medio al pin RXD del HC-05; el extremo libre de la 2k va a GND del Arduino. Quinto, revisa todo dos veces: VCC a 5V, GND a GND, TXD a pin 10, y el punto medio del divisor al RXD.

Usaremos la librería SoftwareSerial para crear un segundo puerto serie en los pines 10 y 11, porque el puerto serie por hardware (pines 0 y 1) lo necesitamos libre para hablar con la computadora por USB mientras depuramos. SoftwareSerial nos permite decir: pin 10 será RX y pin 11 será TX. El programa crea ese puerto a 9600 baudios y luego hace de puente: todo lo que llega del HC-05 lo muestra en el monitor serie, y todo lo que escribes en el monitor serie lo manda al HC-05. Es un espejo entre el Bluetooth y el USB.

```
// Ejemplo B: puente entre HC-05 y monitor serie
// Sirve para probar que el modulo comunica en ambas direcciones

#include <SoftwareSerial.h>

SoftwareSerial bt(10, 11); // RX en pin 10, TX en pin 11

void setup() {
  Serial.begin(9600);      // puerto con la computadora
  bt.begin(9600);          // puerto con el HC-05
  Serial.println("Puente listo. Empareja el HC-05 y abre la app.");
}

void loop() {
  if (bt.available() > 0) { // llego algo por Bluetooth?
    char c = bt.read();
    Serial.print("BT -> PC: "); // lo mostramos en el monitor
    Serial.println(c);
  }
  if (Serial.available() > 0) { // escribiste algo en el monitor?
    char c = Serial.read();
    bt.print(c);                // se lo mandamos al modulo
    Serial.print("PC -> BT: ");
    Serial.println(c);
  }
}
```

La prueba completa es así: sube el programa y abre el monitor serie. Alimenta el circuito y observa el LED del HC-05 parpadeando rápido. En tu celular, abre ajustes de Bluetooth, busca

"HC-05", emparejalo con PIN 1234. Abre tu app de control Bluetooth, conéctala al HC-05 y observa el LED del módulo: debe cambiar a parpadeo lento. Ahora presiona un botón en la app, por ejemplo la flecha de adelante: en el monitor serie de tu computadora debe aparecer "BT -> PC: A" (o la letra que envíe ese botón). Anota que letra envía cada botón de tu app, porque esa es la información que necesitas para tu protocolo. Luego escribe una letra en el monitor serie y presionala: si la app tiene una terminal o pantalla de recepción, debería mostrarla. Si todo esto funciona, tu enlace Bluetooth está perfecto y puedes pasar al carrito.

Si el celular no encuentra el HC-05, verifica la alimentación y que el LED parpadee rápido (si parpadea lento, ya está conectado a otro dispositivo: apaga el Bluetooth de los demás aparatos cercanos). Si la app se conecta pero no llega nada al monitor, revisa que los pines TX/RX estén cruzados correctamente: TXD del módulo al pin 10 (RX del software) y el divisor del pin 11 (TX del software) al RXD del módulo. El error de TX con TX es el más común.

Ejemplo C: carrito controlado por Bluetooth

Con la comunicación probada, ahora si vamos a mover el robot. Este ejemplo es un carrito de control remoto: recibe los comandos A, T, I, D, S por Bluetooth y mueve los motores según corresponda, con control de velocidad con V y v, y bocina con B. Es el módulo de movimiento más el módulo Bluetooth trabajando juntos, y es la base directa del modo manual del proyecto final.

El circuito suma el driver de motores L298N a lo del ejemplo B. Conecta así: los dos motores a las salidas OUT1/OUT2 (motor izquierdo) y OUT3/OUT4 (motor derecho) del L298N. Los pines IN1, IN2, IN3, IN4 del L298N a los pines digitales 4, 5, 6, 7 del Arduino. Los pines ENA y ENB del L298N (control de velocidad por PWM) a los pines 3 y 9 del Arduino (ambos con PWM, marcados con ~). La alimentación de motores: el portabaterías al borne de 12V y GND del L298N. El HC-05 igual que en el ejemplo B (pines 10 y 11 con divisor de voltaje). El buzzer: su positivo al pin 8 del Arduino y su negativo a GND. Importante: une las tierras, es decir, el GND del L298N, el GND del Arduino y el GND del HC-05 deben estar conectados entre sí. Sin tierra común, las señales no tienen referencia y nada funciona bien.

El programa define funciones de movimiento claras: adelante(), atrás(), izquierda(), derecha(), detener() y setVelocidad(). La velocidad se guarda en una variable (0 a 255) y se aplica con analogWrite a ENA y ENB. El loop lee el Bluetooth con SoftwareSerial y usa un switch para ejecutar cada comando. Nota que los comandos de dirección solo configuran los pines IN1-IN4; la velocidad siempre sale de la variable velocidad. Así, V y v cambian la velocidad sin detener el movimiento actual.

```
// Ejemplo C: carrito controlado por Bluetooth
```

```

// Comandos: A adelante, T atras, I izquierda, D derecha, S alto
//           V mas velocidad, v menos velocidad, B bocina

#include <SoftwareSerial.h>

SoftwareSerial bt(10, 11); // RX, TX para el HC-05

// Pines del driver L298N
const int IN1 = 4, IN2 = 5, IN3 = 6, IN4 = 7;
const int ENA = 3, ENB = 9; // PWM para velocidad
const int pinBuzzer = 8;

int velocidad = 180; // velocidad inicial (0-255)

void setup() {
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
  pinMode(ENA, OUTPUT); pinMode(ENB, OUTPUT);
  pinMode(pinBuzzer, OUTPUT);
  Serial.begin(9600);
  bt.begin(9600);
  aplicarVelocidad();
  detener();
  Serial.println("Carrito BT listo. Esperando comandos...");
}

void loop() {
  if (bt.available() > 0) {
    char cmd = bt.read();
    Serial.print("Comando recibido: ");
    Serial.println(cmd);
    switch (cmd) {
      case 'A': adelante(); break;
      case 'T': atras(); break;
      case 'I': izquierda(); break;
      case 'D': derecha(); break;
      case 'S': detener(); break;
      case 'V': subirVelocidad(); break;
      case 'v': bajarVelocidad(); break;
      case 'B': bocina(); break;
    }
  }
}

void aplicarVelocidad() {

```

```

    analogWrite(ENA, velocidad);
    analogWrite(ENB, velocidad);
}

void adelante() {
    digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
    digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
    aplicarVelocidad();
}

void atras() {
    digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
    digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
    aplicarVelocidad();
}

void izquierda() {
    digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
    digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
    aplicarVelocidad();
}

void derecha() {
    digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
    aplicarVelocidad();
}

void detener() {
    digitalWrite(IN1, LOW); digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW); digitalWrite(IN4, LOW);
}

void subirVelocidad() {
    velocidad += 30;
    if (velocidad > 255) velocidad = 255;
    aplicarVelocidad();
    Serial.print("Velocidad: ");
    Serial.println(velocidad);
}

void bajarVelocidad() {
    velocidad -= 30;
    if (velocidad < 0) velocidad = 0;
    aplicarVelocidad();
}

```

```

    Serial.print("Velocidad: ");
    Serial.println(velocidad);
}

void bocina() {
    digitalWrite(pinBuzzer, HIGH);
    delay(200);
    digitalWrite(pinBuzzer, LOW);
}

```

Para probarlo, coloca el carrito elevado (ruedas al aire) la primera vez, por seguridad. Empareja el celular, abre la app y presiona adelante: las ruedas deben girar hacia adelante. Prueba cada dirección y verifica que izquierda y derecha giren correctamente; si un motor gira al revés, intercambia sus dos cables en el L298N. Ajusta la velocidad con V y v y nota como cambia el zumbido de los motores. Si un motor no gira, revisa su cableado y que el ENA/ENB tenga PWM. Cuando todo responda bien con las ruedas al aire, baja el carrito al piso y prueba en un área despejada.

Ejemplo D: proyecto final, robot con dos modos

Este es el proyecto final completo: el robot arranca en modo autónomo esquivando obstáculos con el sensor ultrasónico, y con el comando 'M' (o un botón físico) cambia a modo manual por Bluetooth, donde obedecen los comandos del ejemplo C. Otro 'M' lo regresa al modo autónomo. Además hay dos LEDs que indican el modo activo: verde para autónomo, azul para manual.

El circuito es la unión de todo lo anterior más el sensor y los LEDs. Agrega el HC-SR04: VCC a 5V, GND a GND, Trig al pin 12 del Arduino, Echo al pin 13. Ojo: si usabas el pin 13 para el LED integrado, ahora ese pin lo ocupa el Echo del sensor, así que los LEDs indicadores van en otros pines: LED verde (modo autónomo) al pin A0 con resistencia de 220 ohm a GND, LED azul (modo manual) al pin A1 igual. El botón de cambio de modo (opcional pero recomendado) va entre el pin 2 y GND, usando la resistencia pull-up interna del Arduino. El resto queda igual que el ejemplo C: L298N en pines 3-9, HC-05 en 10 y 11 con divisor de voltaje, buzzer en pin 8.

El programa está organizado por módulos, como planeamos. La variable modo guarda el estado: 0 es autónomo, 1 es manual. En el loop, primero se revisa si llegó un comando Bluetooth: si es 'M', se cambia de modo y se detienen los motores por seguridad; si estamos en modo manual, el comando se ejecuta como en el ejemplo C. Después, según el modo activo, se llama a modoAutonomo() o no se hace nada (en manual el robot solo obedece comandos). La función modoAutonomo() mide la distancia y decide: si hay espacio (más de 25

cm), avanza; si hay un obstáculo cerca, se detiene, retrocede un poco, gira y sigue. Los LEDs siempre reflejan el modo actual. El botón físico se lee con antirrebote simple: si se presiona, cambia el modo igual que el comando 'M'.

```
// Proyecto final: robot con modo autonomo y modo manual Bluetooth
// 'M' cambia de modo. En manual: A T I D S V v B como en el ejemplo C.

#include <SoftwareSerial.h>

SoftwareSerial bt(10, 11); // RX, TX para el HC-05

// Driver L298N
const int IN1 = 4, IN2 = 5, IN3 = 6, IN4 = 7;
const int ENA = 3, ENB = 9;
const int pinBuzzer = 8;

// Sensor ultrasonico HC-SR04
const int pinTrig = 12;
const int pinEcho = 13;

// Indicadores y boton
const int ledAuto = A0; // verde: modo autonomo
const int ledManual = A1; // azul: modo manual
const int pinBoton = 2; // boton de cambio de modo

int velocidad = 180;
int modo = 0; // 0 = autonomo, 1 = manual
int ultimoBoton = HIGH;
unsigned long tiempoBoton = 0;

void setup() {
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
  pinMode(ENA, OUTPUT); pinMode(ENB, OUTPUT);
  pinMode(pinBuzzer, OUTPUT);
  pinMode(pinTrig, OUTPUT); pinMode(pinEcho, INPUT);
  pinMode(ledAuto, OUTPUT); pinMode(ledManual, OUTPUT);
  pinMode(pinBoton, INPUT_PULLUP);
  Serial.begin(9600);
  bt.begin(9600);
  aplicarVelocidad();
  detener();
  actualizarLEDs();
  Serial.println("Proyecto final listo. Modo autonomo activo.");
  Serial.println("Envia 'M' para cambiar a modo manual.");
}
```

```

}

void loop() {
    revisarBoton();          // boton fisico tambien cambia el modo

    if (bt.available() > 0) {
        char cmd = bt.read();
        Serial.print("Recibido: ");
        Serial.println(cmd);
        if (cmd == 'M') {
            cambiarModo();
        }
        else if (modo == 1) {
            ejecutarManual(cmd);    // solo en modo manual
        }
    }

    if (modo == 0) {
        modoAutonomo();          // el robot se maneja solo
    }
    // en modo manual no hacemos nada aqui: solo obedecemos comandos
}

void cambiarModo() {
    detener();                  // seguridad: parar antes de cambiar
    modo = 1 - modo;           // alterna entre 0 y 1
    actualizarLEDs();
    bocinaCorta();
    if (modo == 0) Serial.println("Modo AUTONOMO");
    else Serial.println("Modo MANUAL");
}

void actualizarLEDs() {
    digitalWrite(ledAuto, modo == 0 ? HIGH : LOW);
    digitalWrite(ledManual, modo == 1 ? HIGH : LOW);
}

void revisarBoton() {
    int lectura = digitalRead(pinBoton);
    if (lectura == LOW && ultimoBoton == HIGH &&
        millis() - tiempoBoton > 300) {
        cambiarModo();
        tiempoBoton = millis();
    }
    ultimoBoton = lectura;
}

```

```

}

// ---- Modulo sensor ----
float medirDistancia() {
    digitalWrite(pinTrig, LOW);
    delayMicroseconds(2);
    digitalWrite(pinTrig, HIGH);
    delayMicroseconds(10);
    digitalWrite(pinTrig, LOW);
    long duracion = pulseIn(pinEcho, HIGH, 30000);
    float distancia = duracion * 0.0343 / 2.0;
    if (distancia == 0) distancia = 999; // sin eco = muy lejos
    return distancia;
}

// ---- Modo autonomo: evasor de obstaculos ----
void modoAutonomo() {
    float d = medirDistancia();
    if (d > 25) {
        adelante();
    } else {
        detener();
        delay(200);
        atras();
        delay(400);
        detener();
        izquierda();           // gira para buscar otro camino
        delay(500);
        detener();
    }
}

// ---- Modo manual: ejecuta el comando recibido ----
void ejecutarManual(char cmd) {
    switch (cmd) {
        case 'A': adelante(); break;
        case 'T': atras(); break;
        case 'I': izquierda(); break;
        case 'D': derecha(); break;
        case 'S': detener(); break;
        case 'V': subirVelocidad(); break;
        case 'v': bajarVelocidad(); break;
        case 'B': bocina(); break;
        case 'L': alternarLuces(); break;
    }
}

```



```

}

void alternarLuces() {
    // las luces aquí son los mismos LEDs de modo, parpadean como aviso
    for (int i = 0; i < 3; i++) {
        digitalWrite(ledAuto, HIGH); digitalWrite(ledManual, HIGH);
        delay(150);
        actualizarLEDs();
        delay(150);
    }
}

// ---- Modulo de movimiento (igual que el ejemplo C) ----
void aplicarVelocidad() {
    analogWrite(ENA, velocidad);
    analogWrite(ENB, velocidad);
}

void adelante() {
    digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
    digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
    aplicarVelocidad();
}

void atras() {
    digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
    digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
    aplicarVelocidad();
}

void izquierda() {
    digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
    digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
    aplicarVelocidad();
}

void derecha() {
    digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
    aplicarVelocidad();
}

void detener() {
    digitalWrite(IN1, LOW); digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW); digitalWrite(IN4, LOW);
}

void subirVelocidad() {
    velocidad += 30;
    if (velocidad > 255) velocidad = 255;
    aplicarVelocidad();
}

```

```

}
void bajarVelocidad() {
    velocidad -= 30;
    if (velocidad < 0) velocidad = 0;
    aplicarVelocidad();
}
void bocina() {
    digitalWrite(pinBuzzer, HIGH);
    delay(200);
    digitalWrite(pinBuzzer, LOW);
}
void bocinaCorta() {
    digitalWrite(pinBuzzer, HIGH);
    delay(80);
    digitalWrite(pinBuzzer, LOW);
}
}

```

El plan de pruebas del proyecto final, en orden, es: 1) con el robot elevado, verifica que arranca en modo autónomo (LED verde) y las ruedas avanzan; 2) pon tu mano frente al sensor a menos de 25 cm y verifica que retrocede y gira (modo evasor funcionando); 3) empareja el celular y envía 'M': el robot debe detenerse, encenderse el LED azul y sonar la bocina corta; 4) en modo manual prueba A, T, I, D, S, V, v y B uno por uno; 5) envía 'M' de nuevo y verifica que regresa al modo autónomo (LED verde); 6) prueba el botón físico: debe alternar el modo igual que 'M'; 7) baja el robot al piso en un área despejada y repite la secuencia completa; 8) prueba de autonomía: déjalo operar 20 minutos y verifica que las baterías aguantan y nada se calienta de mas. Si alguna prueba falla, regresa al módulo correspondiente: sensor, Bluetooth, movimiento o lógica de modos. Cuando las ocho pasen, tu proyecto final esta terminado. Felicidades: construiste un robot con dos modos de operación, comunicación inalámbrica y comportamiento autónomo. Eso es robótica de verdad.

Práctica

Ejercicio 1: agregar comando de bocina con melodía

Enunciado: el comando 'B' actual solo hace un pitido simple. Modifica el proyecto final para que 'B' toque una pequeña melodía de tres tonos (por ejemplo, tres pitidos de distinta duración) usando el buzzer. Si tu buzzer es activo (solo hace un tono fijo), logra la melodía con pitidos y silencios de distinta duración; si es pasivo, puedes usar la función tone() para cambiar la frecuencia.

Objetivo: practicar la extensión del protocolo de comandos y el control de tiempos con `millis()` o `delay()` en un módulo ya existente, sin romper el resto del programa.

Solución del ejercicio 1

La idea es reemplazar la función `bocina()` por una version que emita una secuencia: pitido corto, pausa, pitido largo, pausa, pitido corto. Con un buzzer activo no podemos cambiar el tono, pero si el ritmo, y eso ya suena a melodía. Usamos una función que enciende y apaga el buzzer con los tiempos dados. El resto del programa no cambia: el comando 'B' sigue llamando a `bocina()`, solo que ahora `bocina()` hace mas cosas.

```
// Melodia ritmica con buzzer activo: corto - largo - corto
void bocina() {
  pitido(120);    // corto
  delay(100);     // silencio
  pitido(350);    // largo
  delay(100);
  pitido(120);    // corto
}

void pitido(int duracionMs) {
  digitalWrite(pinBuzzer, HIGH);
  delay(duracionMs);
  digitalWrite(pinBuzzer, LOW);
}
```

Explicación: `pitido()` es una función auxiliar que recibe la duración y se encarga de encender y apagar el buzzer, así `bocina()` queda legible como una partitura: cada línea es una nota o un silencio. Si tienes un buzzer pasivo, puedes mejorarla usando `tone(pinBuzzer, frecuencia, duración)` con frecuencias distintas, por ejemplo 523, 659 y 784 hercios (las notas do, mi y sol), y `noTone(pinBuzzer)` al terminar. Prueba la melodía primero con el robot elevado para no asustar a nadie con pitidos inesperados.

Ejercicio 2: modo seguidor de línea conmutable por Bluetooth

Enunciado: agrega un tercer modo al proyecto final: modo seguidor de línea. Necesitas dos sensores infrarrojos de línea (tipo TCRT5000) bajo el frente del robot, conectados a los pines A2 (izquierdo) y A3 (derecho). El comando 'M' ahora debe rotar entre tres modos: autónomo, evasor, manual Bluetooth y seguidor de línea. Usa el LED azul parpadeando para indicar el modo seguidor.

Objetivo: practicar la máquina de estados con mas de dos estados, integrar un sensor nuevo al proyecto y diseñar como el usuario distingue los modos.

Solución del ejercicio 2

Cambiamos la variable modo para que valga 0, 1 o 2, y cambiarModo() ahora suma 1 y vuelve a 0 después del 2. Agregamos la función modoLinea() que lee los dos sensores: si ambos ven línea (valor bajo en estos sensores, según como los calibres), avanza; si solo el izquierdo ve línea, gira a la izquierda; si solo el derecho, gira a la derecha; si ninguno ve línea, se detiene o retrocede un poco para reencontrarla. Los sensores TCRT5000 tienen un potenciómetro para ajustar el umbral; lo ideal es leerlos como digital con un umbral calibrado sobre tu pista.

```

// Pines de sensores de linea (agregar a las constantes)
const int sensorLinIzq = A2;
const int sensorLinDer = A3;

int modo = 0; // 0 = autonomo, 1 = manual, 2 = seguidor de linea

void cambiarModo() {
    detener();
    modo = (modo + 1) % 3; // rota: 0 -> 1 -> 2 -> 0
    actualizarLEDs();
    bocinaCorta();
}

void actualizarLEDs() {
    digitalWrite(ledAuto, modo == 0 ? HIGH : LOW);
    // el LED azul fijo indica manual, parpadeante indica seguidor
    digitalWrite(ledManual, modo == 1 ? HIGH : LOW);
}

void modoLinea() {
    int izq = digitalRead(sensorLinIzq); // LOW = sobre linea oscura
    int der = digitalRead(sensorLinDer);
    // parpadeo del LED azul mientras sigue la linea
    digitalWrite(ledManual, (millis() / 300) % 2);

    if (izq == LOW && der == LOW) {
        adelante(); // centrado: sigue recto
    }
    else if (izq == LOW && der == HIGH) {
        izquierda(); // se salio a la derecha: corrige
    }
    else if (izq == HIGH && der == LOW) {
        derecha();
    }
    else {
        detener(); // perdio la linea: se detiene
    }
}

```

Explicación: el truco principal es $\text{modo} = (\text{modo} + 1) \% 3$, que hace que el modo rote entre 0, 1 y 2 sin ifs complicados. En el loop debes agregar: `if (modo == 2) modoLinea();`. El parpadeo del LED azul se logra con $(\text{millis}() / 300) \% 2$, que alterna entre 0 y 1 cada 300 milisegundos sin bloquear el programa con delay. Calibra los sensores sobre tu pista antes de probar en movimiento: haz un programa pequeño que imprima los valores y ajusta el potenciómetro hasta que distinga bien la línea del fondo.

Ejercicio 3: control de brazo robótico por Bluetooth

Enunciado: monta un pequeño brazo de dos servomotores (base y pinza, o hombro y codo) sobre el carrito y controlalo por Bluetooth con comandos nuevos: 'Q' y 'W' mueven el servo 1 a la izquierda y derecha, 'E' y 'R' mueven el servo 2. Cada pulsación mueve el servo 10 grados. Los servos van a los pines 5 y 6? No: esos los usa el L298N. Usa los pines A4 y A5 como salidas digitales para los servos (los pines analógicos también sirven como digitales).

Objetivo: practicar la ampliación del protocolo con comandos nuevos, el uso de la librería Servo y la convivencia de servos con motores DC en el mismo programa.

Solución del ejercicio 3

Incluimos la librería Servo.h, creamos dos objetos Servo y los asociamos a los pines A4 y A5 en el setup. Guardamos el ángulo actual de cada servo en variables y cada comando suma o resta 10 grados, limitando entre 0 y 180 para no forzar el mecanismo. Los comandos nuevos se atienden en `ejecutarManual()` o en una función propia si prefieres separarlos.

```

#include <Servo.h>

Servo servo1;    // base del brazo
Servo servo2;    // pinza o codo
const int pinServo1 = A4;
const int pinServo2 = A5;
int ang1 = 90, ang2 = 90;    // posicion inicial centrada

void setup() {
    // ... resto del setup ...
    servo1.attach(pinServo1);
    servo2.attach(pinServo2);
    servo1.write(ang1);
    servo2.write(ang2);
}

void ejecutarBrazo(char cmd) {
    switch (cmd) {
        case 'Q': ang1 -= 10; break;
        case 'W': ang1 += 10; break;
        case 'E': ang2 -= 10; break;
        case 'R': ang2 += 10; break;
        default: return;    // no era comando de brazo
    }
    ang1 = constrain(ang1, 0, 180);
    ang2 = constrain(ang2, 0, 180);
    servo1.write(ang1);
    servo2.write(ang2);
    Serial.print("Brazo: ");
    Serial.print(ang1);
    Serial.print(", ");
    Serial.println(ang2);
}

```

Explicación: constrain() es una función de Arduino que limita un valor a un rango; así los servos nunca reciben un angulo imposible. Cada pulsacion mueve 10 grados, lo que da control fino sin ser desesperante. En el loop, llama a ejecutarBrazo(cmd) para cada comando recibido, antes o después de ejecutarManual(). Nota importante de energía: los servos consumen bastante corriente al moverse; si los alimentas desde el Arduino y este se reinicia, alimentalos desde una fuente de 5V aparte (uniendo las tierras). Documenta tus comandos nuevos en la tabla del protocolo para no olvidarlos.

Ejercicio 4: telemetria, el robot reporta distancia por Bluetooth

Enunciado: modifica el proyecto final para que, en modo autónomo, el robot envíe por Bluetooth la distancia medida por el sensor ultrasónico una vez por segundo, en formato de texto como "D:45" (distancia 45 cm). En tu app de celular, abre la terminal o pantalla de recepcion para ver los datos llegando en tiempo real. Agrega el comando 'P' para activar o pausar el envío de telemetria.

Objetivo: practicar la comunicación en el sentido robot-hacia-celular, el envío de datos con formato y el control de tiempos sin bloquear el programa usando millis().

Solución del ejercicio 4

Usamos `bt.println()` para enviar texto al celular, igual que `Serial.println()` lo envía a la computadora. Para no saturar el enlace, enviamos una vez por segundo usando `millis()` en lugar de `delay()`, porque `delay()` detendría al robot. La variable `telemetriaActiva` se alterna con el comando 'P'.

```
bool telemetriaActiva = true;
unsigned long ultimaTelemetria = 0;

void loop() {
    // ... lectura de comandos ...
    // en ejecutarManual agregar: case 'P': telemetriaActiva = !telemetriaActiva;
    break;

    if (modo == 0) {
        modoAutonomo();
        enviarTelemetria();
    }
}

void enviarTelemetria() {
    if (!telemetriaActiva) return;
    unsigned long ahora = millis();
    if (ahora - ultimaTelemetria >= 1000) {    // una vez por segundo
        ultimaTelemetria = ahora;
        float d = medirDistancia();
        bt.print("D:");
        bt.println((int)d);    // enviamos "D:45" por ejemplo
    }
}
```


Explicación: el patrón con `millis()` es el temporizador no bloqueante clásico: guardas cuando fue la última vez y comparas con el tiempo actual; si ya paso un segundo, actúas y actualizas la marca. Así el robot sigue esquivando obstáculos mientras reporta datos. El formato "D:45" es fácil de leer para ti y fácil de parsear si después quieres graficarlo en una app mas avanzada. Con 'P' puedes pausar el envío si te estorba mientras manejas en modo manual. Este ejercicio es tu primera telemetria real: así empiezan los sistemas de monitoreo de drones y rovers profesionales, solo que a mayor escala.

Ejercicio 5: diseña tu propia mejora al proyecto final

Enunciado: este ejercicio es libre y es el mas importante. Diseña una mejora original para tu robot y documentala como un mini-proyecto: escribe el objetivo en una frase, la lista de materiales extra, los comandos nuevos del protocolo (actualiza tu tabla), el diagrama de conexiones y el código del módulo nuevo integrado al programa principal. Ideas para inspirarte: un modo "patrulla" que recorra un cuadrado y reporte cuando encuentra obstáculos; luces direccionales que parpadeen al girar como las de un coche; un sensor de luz que haga al robot buscar o huir de la luz; un modo "sigueme" usando un segundo HC-05 o un sensor infrarrojo; o una alarma antirrobo que suene si alguien mueve el robot (usa un sensor de vibración o un acelerómetro).

Objetivo: practicar el ciclo completo de un proyecto real, desde la idea hasta la prueba, aplicando todo lo aprendido en el curso sin una receta paso a paso.

Solución del ejercicio 5

No hay una solución única porque el ejercicio es de diseño libre, pero aquí tienes un ejemplo resuelto para guiarte: luces direccionales automáticas. Objetivo: "El robot enciende una luz amarilla intermitente del lado hacia donde va a girar, como las direccionales de un coche". Materiales extra: 2 LEDs amarillos con resistencias de 220 ohm. Comandos nuevos: ninguno, es automático. Conexiones: LED izquierdo al pin A2 (con resistencia a GND), LED derecho al pin A3. Código: modificamos `izquierda()` y `derecha()` para activar la direccional correspondiente, y `detener()` y `adelante()` para apagarlas. Para el parpadeo sin bloquear usamos `millis()`.

```

const int dirIzq = A2;    // direccional izquierda
const int dirDer = A3;    // direccional derecha
unsigned long ultimoParpadeo = 0;
bool estadoDir = false;
int dirActiva = 0;        // 0 = ninguna, 1 = izquierda, 2 = derecha

void setup() {
    // ... resto del setup ...
    pinMode(dirIzq, OUTPUT);
    pinMode(dirDer, OUTPUT);
}

void loop() {
    // ... resto del loop ...
    actualizarDireccionales(); // llamar siempre al final del loop
}

void actualizarDireccionales() {
    unsigned long ahora = millis();
    if (ahora - ultimoParpadeo >= 400) {
        ultimoParpadeo = ahora;
        estadoDir = !estadoDir;
        digitalWrite(dirIzq, (dirActiva == 1 && estadoDir) ? HIGH : LOW);
        digitalWrite(dirDer, (dirActiva == 2 && estadoDir) ? HIGH : LOW);
    }
}

// En tus funciones de movimiento agrega:
// izquierda(): dirActiva = 1;
// derecha():   dirActiva = 2;
// adelante(), atras(), detener(): dirActiva = 0;

```

Explicación: la variable `dirActiva` recuerda que direccional esta encendida y `actualizarDireccionales()` la hace parpadear cada 400 ms sin detener el programa. Cada vez que el robot gira, la función de giro correspondiente registra el lado; cuando avanza o se detiene, se apagan. Este ejemplo muestra el método general para cualquier mejora: define que quieres, lista lo que necesitas, conecta, programa el módulo, intégralo al loop y pruébalo por separado antes de mezclarlo. Ahora te toca a ti: documenta tu mejora con ese mismo orden y tendrás tu primer proyecto de diseño propio.

Resumen de la unidad

En esta unidad aprendiste a comunicar tu robot sin cables y cerraste el curso con un proyecto final integrador. Empezamos con la comunicación serial: `Serial.begin` para abrir el puerto a 9600 baudios, `Serial.available` para saber si llegaron datos y `Serial.read` para leerlos, todo probado primero con el monitor serie. Descubriste que el parseo de comandos de una sola letra ("A" adelante, "S" alto) es la base de todo control remoto en este nivel, porque es simple, depurable y fácil de extender.

Luego conociste el módulo Bluetooth HC-05: sus modos maestro y esclavo, el emparejamiento con PIN 1234, la velocidad de 9600 baudios y sus conexiones, con el detalle crítico de cruzar TX con RX y de proteger su pin RXD de 3.3V con un divisor de voltaje de 1k y 2k ohm. Aprendiste a usar `SoftwareSerial` para tener un segundo puerto serie en los pines 10 y 11 sin pelear con el USB, y a elegir una app de control en tu celular cuyos botones envíen las letras de tu protocolo.

Definiste tu propio protocolo de comandos como una tabla clara y lo aplicaste primero a un LED, luego a un puente de prueba con el HC-05 y después a un carrito completo con control de velocidad y bocina. Con esos módulos probados, integraste el proyecto final: un robot móvil con modo autónomo evasor de obstáculos y modo manual por Bluetooth, con cambio de modo por comando 'M' o botón físico, LEDs indicadores y plan de pruebas de ocho pasos.

También aprendiste la parte seria de todo proyecto: planificar con objetivo, materiales, diagrama, código por módulos y pruebas; depurar de forma sistemática del módulo simple al sistema completo; alimentar tu robot con baterías calculando la autonomía aproximada; darle mantenimiento; y trabajar con seguridad. Y miraste hacia adelante: ROS, el ESP32 y la impresión 3D como tus próximos pasos.

Felicidades por terminar el curso. Tienes un robot que se mueve solo, obedece a tu celular y que tu diseñaste, armaste, programaste y depuraste. Guarda tu código, documenta tu proyecto y presume tu robot: te lo ganaste. Nos vemos en el siguiente nivel, en lucasegaacademy.com.

EXAMEN FINAL

Este es el examen final del libro "Curso Completo de Robótica" de Lucas EGA Academy (lucasegaacademy.com). Contiene 50 preguntas que cubren las 10 unidades del curso, con aproximadamente 5 preguntas por unidad: 20 de opción múltiple, 15 de verdadero o falso y 15 abiertas. Para calificarte, asigna 2 puntos a cada pregunta para un total de 100 puntos. En las preguntas de opción múltiple y en las de verdadero o falso solo hay una respuesta correcta. En las preguntas abiertas se evalúa que tu explicación incluya las ideas clave que aparecen en la solución, que el procedimiento de los cálculos sea correcto y que el código compile y cumpla lo que se pide. Se considera aprobado con 60 puntos o mas, notable con 80 puntos o mas y excelente con 90 puntos o mas. Te recomendamos resolverlo sin consultar el libro la primera vez y usar la sección de respuestas solo para revisar al final.

PREGUNTA 1: Cual es la mejor definición de un robot? (Unidad 1)

a) Una máquina que solo puede funcionar con un control remoto b) Un sistema que percibe su entorno con sensores, procesa la información con un controlador y actúa sobre el entorno con actuadores c) Un programa de computadora que no tiene partes físicas d) Un juguete electrónico sin sensores ni programas

PREGUNTA 2: Un robot siempre necesita estar conectado a una computadora para poder funcionar. (Verdadero / Falso) (Unidad 1)

PREGUNTA 3: Cuales son los tres bloques básicos que forman un robot? (Unidad 1)

a) Motor, batería y cable b) Sensores, controlador y actuadores c) Pantalla, teclado y bocina d) Antena, cámara y llantas

PREGUNTA 4: Los actuadores son los componentes que permiten al robot moverse o interactuar físicamente con su entorno, como los motores y los servomotores. (Verdadero / Falso) (Unidad 1)

PREGUNTA 5: Explica con tus palabras la diferencia entre un robot autónomo y un robot teleoperado, y da un ejemplo de cada uno. (Unidad 1)

PREGUNTA 6: Que es Arduino? (Unidad 2)

a) Un lenguaje de programación para teléfonos celulares b) Una plataforma de hardware y software de código abierto basada en microcontroladores, pensada para aprender electrónica y robótica c) Una marca de computadoras de escritorio d) Un tipo de sensor de distancia

PREGUNTA 7: Cual es el voltaje de operación de la tarjeta Arduino UNO? (Unidad 2)

a) 12 V b) 3.3 V c) 5 V d) 9 V

PREGUNTA 8: Los pines digitales del Arduino UNO solo pueden configurarse como entradas y nunca como salidas. (Verdadero / Falso) (Unidad 2)

PREGUNTA 9: Explica que hacen las funciones setup() y loop() en un programa de Arduino y por que ambas son necesarias. (Unidad 2)

PREGUNTA 10: Escribe un programa completo de Arduino que haga parpadear un LED conectado al pin 13, encendido durante un segundo y apagado durante un segundo, de forma repetida. (Unidad 2)

PREGUNTA 11: La ley de Ohm se expresa con la formula: (Unidad 3)

a) $V = I + R$ b) $V = I \times R$ c) $I = V + R$ d) $R = V + I$

PREGUNTA 12: Un LED rojo funciona con 2 V y 20 mA. Si lo alimentas con una batería de 9 V, calcula el valor de la resistencia que debes conectar en serie para no quemarlo. Muestra todo el procedimiento. (Unidad 3)

PREGUNTA 13: En un circuito en paralelo, la corriente total se divide entre las ramas pero el voltaje es el mismo en cada una de ellas. (Verdadero / Falso) (Unidad 3)

PREGUNTA 14: Para que sirve una resistencia en un circuito electrónico? (Unidad 3)

a) Para almacenar carga eléctrica b) Para limitar el paso de la corriente eléctrica c) Para generar voltaje por si misma d) Para amplificar señales de audio

PREGUNTA 15: Explica por que es necesario conectar una resistencia en serie con un LED y que pasaría si la omities. (Unidad 3)

PREGUNTA 16: Como mide la distancia el sensor ultrasónico HC-SR04? (Unidad 4)

a) Emitiendo luz infrarroja y midiendo su rebote b) Emitiendo un pulso de ultrasonido y midiendo el tiempo que tarda el eco en regresar c) Detectando el campo magnetico de los objetos d) Midiendo la temperatura del aire entre el sensor y el objeto

PREGUNTA 17: El HC-SR04 recibe el eco después de 12000 microsegundos. Sabiendo que el sonido viaja a 343 metros por segundo, calcula la distancia al objeto en centímetros. Muestra todo el procedimiento. (Unidad 4)

PREGUNTA 18: El sensor HC-SR04 puede medir distancias de hasta 400 cm de forma confiable en condiciones normales. (Verdadero / Falso) (Unidad 4)

PREGUNTA 19: En el HC-SR04, que función tiene cada pin de disparo y escucha? (Unidad 4)

a) TRIG envía el pulso de ultrasonido y ECHO recibe el eco b) TRIG recibe el eco y ECHO envía el pulso c) Ambos pines sirven para alimentar el sensor d) TRIG es tierra y ECHO es el voltaje de alimentación

PREGUNTA 20: Escribe un programa de Arduino que mida la distancia con un HC-SR04 (TRIG en el pin 9, ECHO en el pin 10) y la muestre en el monitor serial en centímetros cada medio segundo. (Unidad 4)

PREGUNTA 21: Que es un LDR? (Unidad 5)

a) Un sensor de temperatura digital b) Una resistencia cuyo valor cambia según la cantidad de luz que recibe c) Un tipo de motor pequeño d) Un diodo emisor de luz

PREGUNTA 22: Para que se usa comunmente un sensor infrarrojo en robótica? (Unidad 5)

a) Para medir la humedad del ambiente b) Para detectar líneas en el piso, obstáculos cercanos o la presencia de objetos c) Para generar sonidos y melodías d) Para cargar la batería del robot

PREGUNTA 23: El sensor DHT11 puede medir la temperatura y la humedad del ambiente. (Verdadero / Falso) (Unidad 5)

PREGUNTA 24: Un sensor de temperatura LM35 entrega 10 mV por cada grado Celsius y en este momento su salida es de 0.28 V. a) Cual es la temperatura en grados Celsius? b) Convierte ese valor a grados Fahrenheit. Muestra todo el procedimiento. (Unidad 5)

PREGUNTA 25: En un circuito con un LDR y una resistencia fija formando un divisor de voltaje, cuando aumenta la luz que recibe el LDR el voltaje medido en el punto medio del divisor siempre aumenta. (Verdadero / Falso) (Unidad 5)

PREGUNTA 26: Que es la modulación por ancho de pulso (PWM)? (Unidad 6)

a) Un tipo de sensor de velocidad b) Una técnica que controla la potencia entregada a un dispositivo variando el tiempo que la señal esta en alto dentro de cada ciclo c) Un protocolo de comunicación inalámbrica d) Un modelo de batería recargable

PREGUNTA 27: El módulo L298N permite controlar tanto la dirección de giro como la velocidad de un motor de corriente directa. (Verdadero / Falso) (Unidad 6)

PREGUNTA 28: En el módulo L298N, para que sirven los pines ENA y ENB? (Unidad 6)

a) Son la alimentación principal del módulo b) Habilitan cada puente H y permiten controlar la velocidad de los motores con señales PWM c) Son salidas para conectar sensores d) Son pines que nunca se utilizan

PREGUNTA 29: Es seguro conectar un motor de corriente directa directamente a un pin digital del Arduino porque el pin puede entregar toda la corriente que el motor necesita. (Verdadero / Falso) (Unidad 6)

PREGUNTA 30: Escribe un programa de Arduino que controle un motor DC con un L298N (IN1 en el pin 8, IN2 en el pin 9, ENA en el pin 10 con PWM). El motor debe girar hacia adelante a velocidad media durante 2 segundos, detenerse 1 segundo, girar hacia atrás a velocidad máxima durante 2 segundos y repetir la secuencia. (Unidad 6)

PREGUNTA 31: Como se controla la posición de un servomotor estándar? (Unidad 7)

a) Con una señal analógica de voltaje variable b) Con pulsos de entre 1 y 2 milisegundos que indican el ángulo deseado c) Conectándolo directamente a la batería d) Con corriente alterna de la red eléctrica

PREGUNTA 32: Un servomotor estándar como el SG90 puede girar de forma continua 360 grados igual que un motor DC normal. (Verdadero / Falso) (Unidad 7)

PREGUNTA 33: En la librería Servo de Arduino, que hace la instrucción `myservo.write(90)`? (Unidad 7)

a) Apaga el servomotor b) Mueve el servomotor a la posición de 90 grados c) Gira el servomotor a 90 revoluciones por minuto d) Lee la posición actual del servomotor

PREGUNTA 34: Explica la diferencia entre un motor de corriente directa y un servomotor, y menciona en que situación usarías cada uno. (Unidad 7)

PREGUNTA 35: Escribe un programa de Arduino que haga un barrido con un servomotor conectado al pin 6, moviéndolo de 0 a 180 grados y de regreso a 0, con pausas de 20 milisegundos entre cada grado. (Unidad 7)

PREGUNTA 36: Como detecta la línea un robot seguidor de línea que usa sensores infrarrojos? (Unidad 8)

a) La línea negra refleja mas luz infrarroja que el fondo blanco b) La superficie blanca refleja mas luz infrarroja que la línea negra, y el sensor distingue esa diferencia de reflexión c) El sensor mide la temperatura de la línea d) El robot usa un imán que va enterrado bajo la pista

PREGUNTA 37: Un robot evasor de obstáculos típicamente usa un sensor ultrasónico montado al frente para detectar objetos en su camino. (Verdadero / Falso) (Unidad 8)

PREGUNTA 38: En un seguidor de línea con dos sensores IR (izquierdo y derecho) sobre una línea negra con fondo blanco, si solo el sensor izquierdo detecta la línea, que debe hacer el robot? (Unidad 8)

a) Seguir avanzando en línea recta b) Girar a la izquierda para regresar a la línea c) Girar a la derecha para regresar a la línea d) Detenerse para siempre

PREGUNTA 39: El algoritmo básico de un evasor de obstáculos es: avanzar, y al detectar un objeto detenerse para siempre sin intentar rodearlo. (Verdadero / Falso) (Unidad 8)

PREGUNTA 40: Escribe un programa de Arduino para un robot seguidor de línea con dos sensores IR (izquierdo en el pin 2, derecho en el pin 3) y dos motores controlados por un L298N (motor izquierdo: IN1 en pin 8, IN2 en pin 9; motor derecho: IN3 en pin 10, IN4 en pin 11). El robot debe seguir una línea negra sobre fondo blanco. Supón que cada sensor devuelve HIGH cuando detecta la línea negra. (Unidad 8)

PREGUNTA 41: Que son los grados de libertad en un brazo robótico? (Unidad 9)

a) La cantidad de baterías que usa el brazo b) El número de movimientos independientes que el brazo puede realizar c) La velocidad máxima a la que se mueve el brazo d) El peso máximo que el brazo puede cargar

PREGUNTA 42: Un brazo robótico con mas grados de libertad puede alcanzar mas posiciones y orientaciones que uno con menos grados de libertad. (Verdadero / Falso) (Unidad 9)

PREGUNTA 43: Que es el efector final de un brazo robótico? (Unidad 9)

a) El motor que esta en la base del brazo b) La herramienta o pinza que esta en el extremo del brazo y que interactua con los objetos c) El cable de alimentación del brazo d) El programa que controla el brazo

PREGUNTA 44: La cinematica inversa calcula los ángulos de cada articulacion a partir de la posición deseada del efector final, y es exactamente el mismo calculo sencillo que la cinematica directa. (Verdadero / Falso) (Unidad 9)

PREGUNTA 45: Describe la secuencia de movimientos que debe hacer un brazo robótico de 3 grados de libertad para recoger un cubo que esta sobre una mesa y colocarlo dentro de una caja que esta a un lado. (Unidad 9)

PREGUNTA 46: En la comunicación serial, que significan TX y RX? (Unidad 10)

a) Son dos tipos de batería b) TX transmite datos y RX recibe datos c) Son pines de alimentación eléctrica d) Son dos protocolos de Bluetooth

PREGUNTA 47: Que es el módulo HC-05? (Unidad 10)

a) Un sensor de distancia por ultrasonido b) Un módulo Bluetooth que permite comunicar el Arduino de forma inalámbrica con un celular o una computadora c) Un tipo de servomotor d) Una pantalla de cristal líquido

PREGUNTA 48: Para que la comunicación serial funcione, ambos dispositivos deben estar configurados a la misma velocidad de transmisión (baudios). (Verdadero / Falso) (Unidad 10)

PREGUNTA 49: Describe tu proyecto final del curso: que robot vas a construir, que sensores y actuadores usará, y como integra lo aprendido en las 10 unidades del libro. (Unidad 10)

PREGUNTA 50: Escribe un programa de Arduino que encienda un LED conectado al pin 13 cuando reciba el carácter '1' por Bluetooth (HC-05 conectado al puerto serial a 9600 baudios) y que apague el LED cuando reciba el carácter '0'. (Unidad 10)

Respuestas y soluciones

RESPUESTA 1: La respuesta correcta es la opción b). Un robot es un sistema que percibe su entorno con sensores, procesa la información con un controlador y actúa sobre el entorno con actuadores. Esta definición es correcta porque describe el ciclo completo de funcionamiento de cualquier robot: sensor, pensar y actuar. La opción a) es incorrecta porque muchos robots son autónomos y no necesitan control remoto. La opción c) es incorrecta porque un robot siempre tiene una parte física que interactúa con el mundo real; un programa sin cuerpo es solo software. La opción d) es incorrecta porque un juguete sin sensores ni programas no puede percibir ni decidir, así que no califica como robot aunque tenga forma de máquina. Como ejercicio extra, identifica estos tres bloques en cualquier robot comercial que conozcas, desde una aspiradora robot hasta un brazo industrial: siempre encontrarás sensores que perciben, un controlador que decide y actuadores que ejecutan. Este ciclo de sensor, pensar y actuar se repite en todos los proyectos del libro, así que dominarlo te servirá en cada unidad siguiente.

RESPUESTA 2: Falso. Un robot no necesita estar conectado a una computadora para funcionar, porque lleva su propio controlador a bordo, como un Arduino o cualquier

otro microcontrolador, que ejecuta el programa de forma independiente. La computadora solo se usa durante el desarrollo para escribir y cargar el programa, pero una vez cargado el robot opera solo con su batería y su controlador. De hecho, la gracia de un robot autónomo es precisamente que toma sus propias decisiones sin ayuda externa. Solo los robots teleoperados necesitan una conexión permanente con quien los controla. La excepción son los robots teleoperados, que si necesitan una conexión permanente con su operador, ya sea por cable, radio o Bluetooth. Incluso muchos robots autónomos envían telemetría a una computadora para monitoreo, pero eso es opcional: si la conexión se pierde, el robot sigue funcionando con su programa a bordo. Por eso en tus proyectos siempre debes probar el robot desconectado de la computadora antes de darlo por terminado.

RESPUESTA 3: La respuesta correcta es la opción b). Los tres bloques básicos de un robot son los sensores, el controlador y los actuadores. Los sensores son los sentidos del robot y le permiten conocer su entorno, el controlador es el cerebro que toma decisiones con esa información, y los actuadores son los músculos que ejecutan las acciones. La opción a) es incorrecta porque motor, batería y cable son solo piezas sueltas que no describen la arquitectura funcional. La opción c) menciona periféricos de computadora que no tienen nada que ver con la estructura de un robot. La opción d) mezcla componentes opcionales que no forman el esquema básico. Un buen ejercicio es tomar el robot que armes en el curso y etiquetar cada componente con el bloque al que pertenece: el HC-SR04 y los sensores IR son sensores, el Arduino es el controlador, y los motores con el L298N son actuadores. La batería y la estructura son soporte, no bloques funcionales. Hacer este mapa mental te ayudara a diagnosticar fallas: si el robot no se mueve, el problema puede estar en cualquiera de los tres bloques.

RESPUESTA 4: Verdadero. Los actuadores son efectivamente los componentes que convierten las órdenes del controlador en movimiento o acción física sobre el entorno. Los ejemplos mas comunes en este curso son los motores de corriente directa, que hacen girar las ruedas, y los servomotores, que posicionan brazos o pinzas en un angulo preciso. Sin actuadores el robot podría sensor y pensar, pero no podría hacer nada util en el mundo físico. Por eso se les considera uno de los tres bloques fundamentales junto con los sensores y el controlador.

RESPUESTA 5: Un robot autónomo toma sus propias decisiones a partir de lo que detectan sus sensores, sin intervención humana durante su operación; un ejemplo es el robot aspiradora que recorre la casa esquivando muebles por si solo. Un robot teleoperado, en cambio, es controlado a distancia por una persona en tiempo real; un ejemplo es un dron que un piloto maneja con un control remoto o un robot de

exploración guiado por un operador. La diferencia clave esta en quien toma las decisiones: el programa a bordo en el autónomo, y el ser humano en el teleoperado. Ambos usan sensores y actuadores, pero el flujo de control es distinto. Existe además una categoría intermedia: los robots semiautonomos, que combinan ambos modos. Por ejemplo, un dron que mantiene solo su altura y posición pero el piloto decide a donde volar, o una aspiradora robot que limpia sola pero tu le indicas por la app que cuartos debe evitar. En la práctica muchos proyectos reales mezclan autonomía con teleoperacion según lo que sea mas seguro o conveniente para cada tarea.

RESPUESTA 6: La respuesta correcta es la opción b). Arduino es una plataforma de hardware y software de código abierto basada en microcontroladores, creada precisamente para que estudiantes y aficionados aprendan electrónica y robótica sin complicaciones. Incluye las tarjetas físicas y el entorno de programación gratuito donde se escribe el código. La opción a) es incorrecta porque Arduino no es un lenguaje para celulares. La opción c) es incorrecta porque no fábrica computadoras de escritorio. La opción d) es incorrecta porque Arduino es la plataforma completa, no un sensor individual. Los programas de Arduino se escriben en el Arduino IDE, un entorno gratuito donde el código se organiza en archivos llamados sketches. El lenguaje es una version simplificada de C++, así que lo que aprendas aquí te servirá para programar otros microcontroladores. Al ser código abierto, miles de personas comparten librerías y ejemplos, lo que significa que casi cualquier sensor o módulo que compres ya tendrá código de ejemplo listo para usar.

RESPUESTA 7: La respuesta correcta es la opción c). El Arduino UNO opera a 5 V, lo que significa que sus pines digitales entregan 5 voltios cuando están en estado HIGH y sus entradas están diseñadas para leer señales de ese nivel. Este dato es fundamental porque determina con que sensores y actuadores es compatible directamente la tarjeta. La opción a) de 12 V es incorrecta porque ese voltaje quemaría la tarjeta si se aplica a sus pines. La opción b) de 3.3 V corresponde a otras tarjetas como la Due o el ESP32, no al UNO. La opción d) de 9 V es un voltaje común de baterías para alimentar la tarjeta por su conector externo, pero no es su voltaje de operación interno. El Arduino UNO también tiene un pin VIN y un conector de barril que aceptan entre 7 y 12 V, porque la tarjeta trae un regulador interno que los convierte a los 5 V de operación. Sin embargo, nunca debes aplicar mas de 5 V directamente a un pin de entrada o salida, porque eso destruiria el microcontrolador. Cuando dudes del voltaje de un sensor, revisa su hoja de datos antes de conectarlo.

RESPUESTA 8: Falso. Los pines digitales del Arduino UNO pueden configurarse tanto como entradas como salidas usando la función `pinMode()`. Como entrada sirven para

leer sensores, botones o señales digitales, y como salida sirven para encender LEDs, activar relevadores o enviar señales a otros módulos. Esta flexibilidad es una de las razones por las que Arduino es tan útil en robótica. Si los pines solo fueran entradas, el Arduino podría sentir el mundo pero nunca podría actuar sobre él, y no serviría para controlar motores ni ningún actuador.

RESPUESTA 9: La función `setup()` se ejecuta una sola vez cuando el Arduino se enciende o se reinicia, y sirve para la configuración inicial: declarar pines como entradas o salidas, iniciar la comunicación serial y fijar estados iniciales. La función `loop()` se ejecuta una y otra vez en un ciclo infinito después de `setup()`, y ahí va la lógica principal del programa: leer sensores, tomar decisiones y activar actuadores. Ambas son necesarias porque todo programa embebido necesita una etapa de preparación que ocurra una vez y una etapa de trabajo repetitivo que nunca termina. Sin `setup()` no habría configuración ordenada, y sin `loop()` el programa terminaría de inmediato y el robot se quedaría sin hacer nada. Un error típico de principiante es poner instrucciones de configuración dentro de `loop()`, lo que hace que se repitan miles de veces sin necesidad, o poner la lógica principal dentro de `setup()`, lo que hace que solo se ejecute una vez. Recuerda: `setup()` prepara una vez, `loop()` trabaja para siempre. Cuando tu programa se comporte raro, revisa primero que cada instrucción esté en la función correcta.

RESPUESTA 10: El programa correcto configura el pin 13 como salida en `setup()` y alterna su estado en `loop()` con pausas de un segundo. El pin 13 es ideal porque la mayoría de las tarjetas Arduino UNO traen un LED integrado conectado a él, así que el programa funciona incluso sin componentes externos. La función `delay(1000)` detiene el programa mil milisegundos, es decir un segundo, entre cada cambio de estado. Una solución válida es la siguiente:

```
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

Este código es correcto porque HIGH enciende el LED, LOW lo apaga, y el ciclo se repite para siempre gracias a loop(). Cualquier variante que use otro pin con un LED externo también es válida siempre que configure ese pin como salida. El pin 13 es especial porque casi todas las tarjetas Arduino UNO traen un LED integrado conectado a él, marcado con la letra L en la placa. Eso significa que este programa funciona sin conectar ningún componente externo, ideal para tu primera prueba. Ten en cuenta que delay() detiene por completo el programa, así que mientras el LED está encendido el Arduino no puede hacer nada más; en la unidad 10 verás alternativas no bloqueantes con millis().

RESPUESTA 11: La respuesta correcta es la opción b). La ley de Ohm establece que el voltaje es igual a la corriente multiplicada por la resistencia, es decir $V = I \times R$. Esta fórmula es la base de toda la electrónica porque relaciona las tres magnitudes fundamentales de un circuito. La opción a) es incorrecta porque el voltaje no es la suma de corriente y resistencia; además ni siquiera coinciden las unidades. La opción c) es incorrecta porque la corriente es igual al voltaje entre la resistencia, no a su suma. La opción d) es incorrecta porque la resistencia se calcula como voltaje entre corriente, no como una suma. Un truco para no olvidar las tres formas de la ley de Ohm es el triángulo: escribe la V arriba y la I y la R abajo, tapa la magnitud que quieras calcular y lee la fórmula resultante. Tapando la V queda $I \times R$, tapando la I queda V / R , y tapando la R queda V / I . Con este triángulo puedes despejar cualquiera de las tres magnitudes en segundos sin memorizar tres fórmulas separadas.

RESPUESTA 12: Primero calculamos el voltaje que debe absorber la resistencia: 9 V de la batería menos 2 V que necesita el LED, lo que da 7 V. La corriente que debe pasar es 20 mA, que convertidos a amperios son 0.02 A. Aplicando la ley de Ohm despejada, $R = V / I$, tenemos $R = 7 / 0.02 = 350$ ohmios. En la práctica se usaría el valor comercial más cercano por arriba, como 330 o 390 ohmios, para no exceder la corriente del LED. Si omities este cálculo y conectas el LED directo a los 9 V, la corriente sería muchísimo mayor a 20 mA y el LED se quemaría al instante. Por eso este cálculo es uno de los más importantes del curso. Como verificación extra puedes calcular la potencia que disipa la resistencia con $P = V \times I$: $7 \text{ V} \times 0.02 \text{ A} = 0.14 \text{ W}$. Una resistencia común de 1/4 W (0.25 W) es suficiente porque 0.14 W está por debajo de su límite. Si la potencia calculada superara el valor nominal de la resistencia, habría que usar una de mayor potencia para que no se caliente de más. Este paso extra te evita sorpresas en circuitos reales.

RESPUESTA 13: Verdadero. En un circuito en paralelo cada rama está conectada directamente entre los mismos dos puntos de la fuente, por lo que todas las ramas reciben el mismo voltaje. La corriente total que entrega la fuente, en cambio, se reparte entre las ramas según la resistencia de cada una: la rama con menor resistencia toma

mas corriente. Esto es lo contrario de un circuito en serie, donde la corriente es la misma en todos los componentes pero el voltaje se divide entre ellos. Entender esta diferencia es clave para conectar correctamente sensores y actuadores sin dañarlos. Un ejemplo práctico: si conectas tres LEDs cada uno con su resistencia en paralelo a 5 V, cada rama recibe los 5 V completos y brilla igual que si estuviera sola, pero la fuente debe entregar la suma de las tres corrientes. En cambio, si los conectarás en serie sin rediseñar, el voltaje se repartiría y brillarian poco o nada. Por eso los módulos y sensores se conectan en paralelo a la alimentación: cada uno necesita el voltaje completo.

RESPUESTA 14: La respuesta correcta es la opción b). Una resistencia sirve para limitar el paso de la corriente eléctrica, oponiéndose a su flujo según su valor en ohmios. Gracias a esto protege componentes delicados como los LEDs, que se quemarían con demasiada corriente. La opción a) es incorrecta porque el componente que almacena carga eléctrica es el capacitor, no la resistencia. La opción c) es incorrecta porque una resistencia es un componente pasivo y no puede generar voltaje por sí misma. La opción d) es incorrecta porque amplificar señales es trabajo de los transistores o amplificadores operacionales.

RESPUESTA 15: La resistencia en serie con el LED es necesaria porque el LED por sí solo casi no limita la corriente, así que conectado directo a la fuente dejaría pasar mucha mas corriente de la que soporta. Esa corriente excesiva calienta el semiconductor del LED hasta destruirlo, y el LED se funde en cuestión de segundos o incluso al instante. La resistencia absorbe el voltaje sobrante de la fuente y fija la corriente en un valor seguro, típicamente entre 10 y 20 mA. Sin la resistencia el circuito no regula nada y el componente mas delicado, que es el LED, paga las consecuencias. Por eso la regla de oro es: ningún LED se conecta sin su resistencia limitadora. El voltaje directo del LED varía según su color: el rojo usa unos 2 V, pero el azul o el blanco pueden necesitar hasta 3.3 V. Por eso el calculo de la resistencia siempre debe hacerse con los datos del LED que tengas en la mano, revisando su hoja de datos o el empaque. Usar la resistencia calculada para un LED rojo en un LED azul daría una corriente distinta a la esperada. Cuando compres LEDs, anota sus características para no adivinar.

RESPUESTA 16: La respuesta correcta es la opción b). El HC-SR04 emite un pulso de ultrasonido por su transmisor y luego mide con su receptor el tiempo que tarda el eco en regresar después de rebotar en un objeto. Como la velocidad del sonido en el aire es conocida, el sensor o el programa calculan la distancia multiplicando ese tiempo por la velocidad y dividiendo entre dos, porque el sonido hace el viaje de ida y vuelta. La

opción a) es incorrecta porque el sensor no usa luz infrarroja sino ondas de sonido. La opción c) es incorrecta porque no detecta campos magnéticos. La opción d) es incorrecta porque no mide temperatura para calcular distancia. El HC-SR04 trabaja a 40 kHz, una frecuencia muy por encima del límite del oído humano, por eso se llama ultrasonido: no lo escuchamos aunque el sensor este disparando constantemente. Algunos animales como los perros y los murciélagos si pueden percibir estas frecuencias, algo a considerar si tu robot trabajara cerca de mascotas. El principio es el mismo que usan los murciélagos y los sonares de los submarinos para orientarse.

RESPUESTA 17: El procedimiento es el siguiente. Primero convertimos el tiempo a segundos: 12000 microsegundos son 0.012 segundos. La distancia recorrida por el sonido en ese tiempo es velocidad por tiempo: $343 \times 0.012 = 4.116$ metros. Como ese recorrido incluye la ida hasta el objeto y el regreso del eco, dividimos entre dos: $4.116 / 2 = 2.058$ metros. Convertimos a centímetros multiplicando por 100, lo que da 205.8 cm, aproximadamente 206 cm. Existe también la fórmula rápida que se usa en Arduino: distancia en cm = tiempo en microsegundos / 58, y $12000 / 58 = 206.9$ cm, que confirma el resultado. La división entre dos es el paso que mas se olvida y el error mas común en este calculo. Un detalle fino es que la velocidad del sonido cambia ligeramente con la temperatura del aire: aumenta unos 0.6 m/s por cada grado Celsius. A 20 grados es de unos 343 m/s, que es el valor usado en el calculo, pero en un día muy caluroso sería un poco mayor y la medicion tendría un pequeño error. Para los proyectos del curso este efecto es despreciable, pero en instrumentos de precisión se compensa midiendo también la temperatura ambiente.

RESPUESTA 18: Verdadero. El fabricante del HC-SR04 especifica un rango de medicion de 2 cm a 400 cm, y en condiciones normales de interior el sensor alcanza esos 4 metros sin problema. Por debajo de 2 cm el sensor no puede medir porque el eco regresa demasiado rápido y se confunde con el pulso emitido. Por encima de 400 cm el eco llega tan débil que el receptor ya no lo detecta de forma confiable. Factores como superficies muy absorbentes, ángulos muy inclinados o mucho ruido ambiental pueden reducir el alcance real, pero el valor nominal de 400 cm es correcto.

RESPUESTA 19: La respuesta correcta es la opción a). El pin TRIG es la entrada de disparo: cuando recibe un pulso de al menos 10 microsegundos, el sensor emite la rafaga de ultrasonido. El pin ECHO es la salida de escucha: se pone en alto durante exactamente el tiempo que tardó el eco en regresar, y ese tiempo es el que mide el Arduino con `pulseIn()`. La opción b) invierte las funciones y es incorrecta. La opción c) es incorrecta porque la alimentación del sensor va por los pines VCC y GND. La opción d) es incorrecta porque TRIG y ECHO son pines de señal, no de alimentación. Además

de TRIG y ECHO, el sensor tiene los pines VCC y GND para su alimentación de 5 V, que puedes tomar directo del Arduino porque el HC-SR04 consume muy poca corriente. Un error común es confundir el orden de los pines al conectarlo, así que verifica las etiquetas impresas en la placa del sensor antes de energizar. Si el sensor no responde, lo primero que hay que revisar es que VCC y GND estén bien conectados.

RESPUESTA 20: El programa debe generar el pulso de disparo en TRIG, medir con `pulseIn()` el tiempo en alto de ECHO y convertirlo a centímetros dividiendo entre 58. Además debe iniciar la comunicación serial a 9600 baudios para mostrar el resultado y esperar medio segundo entre mediciones. Una solución válida es la siguiente:

```
const int trigPin = 9;
const int echoPin = 10;

void setup() {
  Serial.begin(9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duracion = pulseIn(echoPin, HIGH);
  float distancia = duracion / 58.0;

  Serial.print("Distancia: ");
  Serial.print(distancia);
  Serial.println(" cm");
  delay(500);
}
```


El pulso de 10 microsegundos en TRIG es el disparo estándar del sensor, y la división entre 58 convierte microsegundos a centímetros. Cualquier variante equivalente que cumpla lo pedido es válida. La función `pulseIn()` tiene un tiempo de espera máximo de un segundo por defecto: si no llega ningún eco, devuelve cero en lugar de quedarse esperando para siempre. Eso evita que tu programa se congele cuando el sensor apunta al vacío. Si necesitas mediciones más rápidas o trabajar con varios sensores, existen librerías como NewPing que optimizan este proceso, pero para el curso la versión con `pulseIn()` es perfecta y muy didáctica.

RESPUESTA 21: La respuesta correcta es la opción b). Un LDR (resistor dependiente de luz) es una resistencia cuyo valor disminuye cuando recibe más luz y aumenta cuando está en oscuridad. Gracias a este comportamiento se usa para detectar niveles de iluminación, por ejemplo para que un robot sepa si es de día o de noche o para seguir una fuente de luz. La opción a) es incorrecta porque el LDR no mide temperatura; eso lo hacen sensores como el LM35 o el DHT11. La opción c) es incorrecta porque el LDR no es un motor ni produce movimiento. La opción d) es incorrecta porque el LDR no emite luz, solo reacciona a la que recibe. Las aplicaciones del LDR son muy variadas: lámparas que se encienden solas al anochecer, robots que buscan o huyen de la luz, y sistemas que ajustan el brillo de una pantalla según el ambiente. En el curso lo usarás formando un divisor de voltaje con una resistencia fija, para convertir su cambio de resistencia en un cambio de voltaje que el Arduino puede leer con una entrada analógica. Es el sensor perfecto para tu primer proyecto con entradas analógicas.

RESPUESTA 22: La respuesta correcta es la opción b). Los sensores infrarrojos se usan comúnmente en robótica para detectar líneas en el piso, obstáculos cercanos o la presencia de objetos, porque emiten luz infrarroja y miden cuanto se refleja de vuelta. En un seguidor de línea distinguen la línea negra del fondo blanco por la diferencia de reflexión, y como detectores de proximidad avisan cuando hay algo enfrente. La opción a) es incorrecta porque la humedad se mide con sensores como el DHT11. La opción c) es incorrecta porque el sensor infrarrojo no genera sonido. La opción d) es incorrecta porque ningún sensor carga baterías.

RESPUESTA 23: Verdadero. El DHT11 es un sensor digital de bajo costo que mide tanto la temperatura como la humedad relativa del ambiente, y entrega ambos valores por un solo pin de datos. Su rango típico es de 0 a 50 grados Celsius y de 20 a 90 por ciento de humedad relativa, suficiente para proyectos educativos y de domótica. Se usa con una librería que interpreta su protocolo digital, así que no necesita conversiones analógicas. Es uno de los sensores más populares para estaciones meteorológicas caseras hechas con Arduino.

RESPUESTA 24: El procedimiento es el siguiente. a) El LM35 entrega 10 mV por cada grado Celsius, así que dividimos el voltaje medido entre esa sensibilidad: 0.28 V son 280 mV, y $280 / 10 = 28$. La temperatura es 28 grados Celsius. b) Para convertir a Fahrenheit usamos la formula $F = C \times 9/5 + 32$. Sustituyendo: $28 \times 1.8 = 50.4$, y $50.4 + 32 = 82.4$. La temperatura es 82.4 grados Fahrenheit. Este ejercicio combina dos habilidades del curso: interpretar la salida de un sensor analógico según su hoja de datos y convertir entre escalas de temperatura. El error mas común es olvidar convertir los voltios a milivoltios antes de dividir. En un proyecto real con Arduino no medirías el voltaje con un multímetro, sino que usarías una entrada analógica: el convertidor del Arduino divide los 5 V en 1024 pasos, así que cada paso vale unos 4.9 mV. Si el LM35 entrega 280 mV, la lectura analógica seria de aproximadamente 57, y tu programa convertiría ese número de vuelta a temperatura. Este encadenamiento de sensor, conversión analógica y formula es el patrón básico de todos los instrumentos digitales.

RESPUESTA 25: Falso. Que el voltaje del punto medio suba o baje al aumentar la luz depende de en que posición del divisor este conectado el LDR. Si el LDR esta del lado de la alimentación (VCC), al bajar su resistencia con mas luz el punto medio sube; pero si el LDR esta del lado de tierra (GND), al bajar su resistencia el punto medio baja porque el LDR "jala" el voltaje hacia tierra. La palabra "siempre" es lo que hace falsa la afirmacion, porque el comportamiento cambia según el diseño del circuito. Por eso antes de programar umbrales hay que medir el circuito real con el multímetro o con el monitor serial.

RESPUESTA 26: La respuesta correcta es la opción b). La modulación por ancho de pulso (PWM) controla la potencia entregada a un dispositivo variando el ciclo de trabajo, es decir, el porcentaje del tiempo que la senal esta en alto dentro de cada periodo. Con un ciclo de trabajo del 50 por ciento un motor recibe en promedio la mitad de la potencia y gira mas lento que con un 100 por ciento. La opción a) es incorrecta porque el PWM no es un sensor. La opción c) es incorrecta porque no es un protocolo inalámbrico. La opción d) es incorrecta porque no es un modelo de batería. En Arduino el PWM se genera con `analogWrite()`, que acepta valores de 0 a 255: 0 es siempre apagado, 255 es siempre encendido y 127 es un ciclo de trabajo del 50 por ciento. Solo los pines marcados con el símbolo ~ en la placa pueden generar PWM por hardware. Esta técnica no solo controla motores: también sirve para atenuar el brillo de un LED o generar tonos con un buzzer, así que la usarás en muchos proyectos.

RESPUESTA 27: Verdadero. El módulo L298N contiene dos puentes H que permiten invertir la polaridad aplicada a cada motor, y con eso se controla la dirección de giro: adelante o atrás. Además, sus pines de habilitacion (ENA y ENB) aceptan señales PWM,

así que también se regula la velocidad variando el ciclo de trabajo. Esta combinación de dirección y velocidad es justo lo que necesita un robot móvil para moverse con precisión. Por eso el L298N es el controlador de motores mas usado en los proyectos de este curso. Un detalle práctico del L298N es que sus transistores internos provocan una caída de voltaje de unos 2 V, así que si lo alimentas con 9 V los motores reciben realmente unos 7 V. Además el módulo se calienta con motores grandes, por eso trae un disipador de aluminio que no debes quitar. Si tus motores giran muy lento, revisa primero el voltaje de la fuente antes de culpar al programa.

RESPUESTA 28: La respuesta correcta es la opción b). Los pines ENA y ENB habilitan cada uno de los dos puentes H del módulo: ENA controla el motor A y ENB el motor B. Cuando reciben una senal PWM, además de habilitar el puente regulan la velocidad del motor según el ciclo de trabajo. La opción a) es incorrecta porque la alimentación del módulo entra por los bornes de tornillo, no por ENA y ENB. La opción c) es incorrecta porque estos pines son entradas de control, no salidas para sensores. La opción d) es incorrecta porque son pines esenciales para variar la velocidad.

RESPUESTA 29: Falso. Un pin digital del Arduino solo puede entregar alrededor de 40 mA como máximo, mientras que un motor de corriente directa pequeño fácilmente consume varios cientos de miliamperios, sobre todo al arrancar. Conectar el motor directo al pin provocaría que el pin se sobrecargue y podría quemar el microcontrolador de forma permanente. Por eso siempre se usa un controlador intermedio como el L298N, que toma la potencia de una fuente externa y solo usa las señales del Arduino para decidir dirección y velocidad. Esta es una de las reglas de seguridad mas importantes del curso. El límite de 40 mA por pin es un máximo absoluto, y en la práctica se recomienda no pasar de 20 mA por pin ni de 200 mA en total para todo el chip. Un motor pequeño al arrancar puede pedir picos de mas de 500 mA, diez veces lo que soporta un pin. El daño no siempre es inmediato: a veces el pin queda debilitado y falla días después. Por eso la regla es tajante: los pines del Arduino solo manejan señales, nunca la potencia de los motores.

RESPUESTA 30: El programa debe configurar los tres pines como salidas y usar `digitalWrite` en IN1 e IN2 para fijar la dirección, y `analogWrite` en ENA para la velocidad con PWM. Velocidad media corresponde a un valor cercano a 127 y velocidad máxima a 255. Una solución valida es la siguiente:

```

const int IN1 = 8;
const int IN2 = 9;
const int ENA = 10;

void setup() {
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(ENA, OUTPUT);
}

void loop() {
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, 127);
  delay(2000);

  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  delay(1000);

  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  analogWrite(ENA, 255);
  delay(2000);
}

```

La dirección cambia invirtiendo IN1 e IN2, y detener el motor se logra poniendo ambos en LOW. Cualquier variante equivalente que respete pines, tiempos y velocidades es válida. Recuerda que el Arduino y la fuente que alimenta los motores deben compartir la tierra (GND) para que las señales de control tengan una referencia común. Sin esa tierra común, el L298N recibe señales flotantes y los motores se comportan de forma errática o no responden. Este es uno de los errores de cableado más frecuentes y también de los más fáciles de corregir: basta un cable de GND entre ambas fuentes.

RESPUESTA 31: La respuesta correcta es la opción b). Un servomotor estándar se controla con pulsos de entre 1 y 2 milisegundos que se repiten cada 20 milisegundos: un pulso de 1 ms lo lleva a 0 grados, uno de 1.5 ms al centro (90 grados) y uno de 2 ms a 180 grados. El circuito interno del servo compara ese pulso con la posición real de su potenciómetro y mueve el motor hasta igualarlos. La opción a) es incorrecta porque el control no es un voltaje analógico variable sino pulsos digitales de ancho preciso. La opción c) es incorrecta porque conectarlo directo a la batería lo dañaría y no controlaría su posición. La opción d) es incorrecta porque los servos de este curso trabajan con

corriente directa. El periodo de 20 milisegundos entre pulsos equivale a una frecuencia de 50 Hz, que es el estándar de los servos de radiocontrol desde hace décadas. La librería Servo de Arduino genera esta señal por hardware en segundo plano, así que tu programa puede hacer otras cosas mientras el servo mantiene su posición. Si el servo vibra o se calienta, revisa que la fuente de alimentación entregue suficiente corriente, porque un servo con poca corriente se comporta mal.

RESPUESTA 32: Falso. Un servomotor estándar como el SG90 tiene un rango limitado de giro, típicamente de 0 a 180 grados, porque su mecanismo interno incluye topes físicos y un potenciómetro que mide la posición dentro de ese rango. Si intentas forzarlo mas allá, los engranes o los topes se pueden dañar. Los servos de rotación continua son una variante modificada que si gira 360 grados, pero entonces pierden el control de posición y se comportan como un motor DC con control de velocidad. Por eso es importante distinguir el servo estándar del de rotación continua antes de usarlo en un proyecto.

RESPUESTA 33: La respuesta correcta es la opción b). La instrucción `myservo.write(90)` le ordena al servomotor moverse a la posición de 90 grados, que es el centro de su recorrido. La librería Servo se encarga de generar automáticamente los pulsos de 1.5 milisegundos necesarios sin que el programador tenga que calcularlos. La opción a) es incorrecta porque para detener un servo estándar no existe un comando de apagado; simplemente se queda en su última posición. La opción c) es incorrecta porque `write()` trabaja con ángulos, no con revoluciones por minuto. La opción d) es incorrecta porque `write()` escribe una posición; para leer se usaría `read()`.

RESPUESTA 34: Un motor de corriente directa gira de forma continua y su velocidad se controla con el voltaje o con PWM, pero no sabe en que posición angular esta; es ideal para las ruedas de un robot móvil donde lo que importa es avanzar y girar sin detenerse en un angulo exacto. Un servomotor, en cambio, se mueve a una posición angular precisa entre 0 y 180 grados y se queda ahí, gracias a su control interno de retroalimentación; es ideal para el cuello de un robot, una pinza o cualquier articulacion que deba colocarse en un angulo exacto. En resumen: usa el motor DC cuando necesites movimiento continuo y velocidad, y usa el servo cuando necesites posición exacta y torque en un angulo determinado. Otro criterio para elegir es el torque: los servos suelen ofrecer mas torque a baja velocidad gracias a su caja de engranes reductora, mientras que un motor DC pequeño gira rápido pero con poca fuerza. Si tu motor DC necesita mas fuerza, puedes anadirle una caja reductora externa, como las de los carritos robóticos. Piensa siempre en que necesita tu mecanismo: velocidad continua o posición con fuerza, y la elección será obvia.

RESPUESTA 35: El programa debe usar la librería Servo, asociar el objeto al pin 6 con attach() y recorrer los ángulos con dos ciclos for: uno de subida de 0 a 180 y otro de bajada de 180 a 0, con delay(20) en cada paso. Una solución valida es la siguiente:

```
#include <Servo.h>

Servo miservo;

void setup() {
  miservo.attach(6);
}

void loop() {
  for (int angulo = 0; angulo <= 180; angulo++) {
    miservo.write(angulo);
    delay(20);
  }
  for (int angulo = 180; angulo >= 0; angulo--) {
    miservo.write(angulo);
    delay(20);
  }
}
```

La pausa de 20 milisegundos entre grados le da tiempo físico al servo para alcanzar cada posición, así el movimiento se ve suave y continuo. Cualquier variante equivalente que haga el barrido completo en ambas direcciones es valida. La pausa de 20 milisegundos no es casual: coincide con el periodo de refresco del servo, así que cada nuevo comando llega justo cuando el servo esta listo para recibirlo. Si usarás pausas mucho mas cortas, el servo no alcanzaria cada posición y el movimiento se vería igual pero el programa seria menos eficiente. Este barrido es además la base para animar el ultrasónico de un lado a otro en el robot evasor de la unidad 8.

RESPUESTA 36: La respuesta correcta es la opción b). Los sensores infrarrojos del seguidor de línea emiten luz infrarroja y miden cuanta regresa reflejada: la superficie blanca refleja mucha luz y la línea negra absorbe casi toda. El comparador del módulo convierte esa diferencia en un 1 o un 0 digital que el Arduino puede leer fácilmente. La opción a) invierte el fenomeno físico y es incorrecta: lo negro absorbe, no refleja. La opción c) es incorrecta porque estos sensores no miden temperatura. La opción d) es incorrecta porque el sistema es optico y no necesita nada enterrado bajo la pista.

RESPUESTA 37: Verdadero. El robot evasor de obstáculos típico lleva un sensor ultrasónico HC-SR04 montado al frente, muchas veces sobre un servomotor que lo hace

"mirar" a los lados. El programa avanza mientras no detecta nada y cuando la distancia medida baja de un umbral, por ejemplo 20 cm, ejecuta la maniobra de evasión: detenerse, retroceder un poco, girar y seguir. El ultrasónico es ideal para esta tarea porque mide distancia real sin contacto y funciona con objetos de muchos materiales. Por eso la combinación de HC-SR04 mas L298N es la base del evasor que se construye en la unidad 8. Muchos diseños montan el HC-SR04 sobre un pequeño servomotor que lo hace girar como una cabeza mirando a izquierda y derecha. Así el robot no solo sabe que hay un obstáculo enfrente, sino que compara la distancia a ambos lados y elige girar hacia donde haya mas espacio libre. Esta combinación de la unidad 4 con la unidad 7 es un excelente ejemplo de como los bloques del curso se integran en robots cada vez mas capaces.

RESPUESTA 38: La respuesta correcta es la opción b). Si solo el sensor izquierdo detecta la línea negra, significa que el robot se ha desviado hacia la derecha y la línea quedó a su izquierda, así que debe girar a la izquierda para regresar sobre ella. La lógica general del seguidor es corregir siempre hacia el lado donde se detecta la línea. La opción a) es incorrecta porque seguir recto haría que el robot se saliera de la pista. La opción c) es incorrecta porque girar a la derecha lo alejaria aun mas de la línea. La opción d) es incorrecta porque detenerse para siempre no es una estrategia de seguimiento.

RESPUESTA 39: Falso. El algoritmo básico del evasor de obstáculos no termina al detectar un objeto, sino que ejecuta una secuencia de escape: detenerse, retroceder un poco, girar hacia el lado con mas espacio libre y volver a avanzar. Algunos diseños incluso montan el ultrasónico en un servo para medir a izquierda y derecha y elegir el mejor camino antes de girar. Detenerse para siempre seria un robot inútil que se quedaría atorado con el primer obstáculo. La gracia del evasor es precisamente que navega de forma autonoma rodeando todo lo que se le ponga enfrente.

RESPUESTA 40: El programa debe leer los dos sensores y aplicar la lógica de corrección: si ambos ven la línea avanza recto, si solo el izquierdo la ve gira a la izquierda, si solo el derecho la ve gira a la derecha, y si ninguno la ve puede seguir recto o detenerse según el diseño. Cada motor se controla con su par de pines del L298N. Una solución valida es la siguiente:

```
const int sensorIzq = 2;  
const int sensorDer = 3;  
const int IN1 = 8;  
const int IN2 = 9;  
const int IN3 = 10;  
const int IN4 = 11;
```

```

void setup() {
  pinMode(sensorIzq, INPUT);
  pinMode(sensorDer, INPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
}

void adelante() {
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
}

void girarIzquierda() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
}

void girarDerecha() {
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
}

void loop() {
  int izq = digitalRead(sensorIzq);
  int der = digitalRead(sensorDer);

  if (izq == HIGH && der == HIGH) {
    adelante();
  } else if (izq == HIGH && der == LOW) {
    girarIzquierda();
  } else if (izq == LOW && der == HIGH) {
    girarDerecha();
  } else {
    adelante();
  }
}

```


Girar se logra moviendo los motores en sentidos opuestos. Cualquier variante con la misma lógica de corrección hacia el lado de la línea es válida. Cuando domines este seguidor básico, el siguiente nivel es el control PID, que ajusta la corrección de forma proporcional al error en lugar de girar siempre igual. El PID hace que el robot siga la línea de forma mucho más suave y rápida, sin zigzaguear. Por ahora el control de todo o nada de este programa es suficiente, pero ten presente que la robótica real usa controladores más finos para lograr movimientos elegantes.

RESPUESTA 41: La respuesta correcta es la opción b). Los grados de libertad son el número de movimientos independientes que un brazo robótico puede realizar, y normalmente corresponden a sus articulaciones: cada articulación que puede girar o desplazarse de forma independiente suma un grado de libertad. Un brazo con 3 grados de libertad puede posicionar su pinza en más lugares que uno con 2. La opción a) es incorrecta porque la cantidad de baterías no tiene relación con la movilidad. La opción c) es incorrecta porque la velocidad es una característica de los motores, no de los grados de libertad. La opción d) es incorrecta porque el peso que carga depende de la potencia de los servos, no del número de movimientos independientes. Como referencia, el brazo humano tiene 7 grados de libertad: 3 en el hombro, 1 en el codo y 3 en la muñeca, lo que nos permite una destreza enorme. Los brazos industriales típicos tienen 6, suficientes para posicionar y orientar una herramienta en cualquier punto de su espacio de trabajo. Cuando diseñes tu brazo en el curso, cada servo que agregues suma un grado de libertad y también un reto más para el control.

RESPUESTA 42: Verdadero. Cada grado de libertad adicional le da al brazo una nueva forma independiente de moverse, lo que expande el conjunto de posiciones y orientaciones que su efector final puede alcanzar. Por ejemplo, un brazo de 2 grados de libertad solo se mueve en un plano, mientras que uno de 3 puede además girar su base y cubrir un volumen completo a su alrededor. Esta es la razón por la que los brazos industriales tienen 6 grados de libertad: así pueden colocar su herramienta en cualquier posición y con cualquier orientación dentro de su alcance. Más grados de libertad también complican el control, pero aumentan la versatilidad.

RESPUESTA 43: La respuesta correcta es la opción b). El efector final es la herramienta o pinza que esta en el extremo del brazo robótico y que interactua directamente con los objetos: puede ser una pinza de dos dedos, una ventosa, un taladro o cualquier herramienta según la tarea. Todo el brazo existe para llevar el efector final a la posición correcta con la orientación adecuada. La opción a) es incorrecta porque el motor de la base es solo la primera articulacion, no el extremo de trabajo. La opción c) es incorrecta porque el cable de alimentación no interactua con los objetos. La opción d) es incorrecta porque el programa es el controlador, no una parte física del brazo.

RESPUESTA 44: Falso. La primera parte es cierta: la cinematica inversa si calcula los ángulos de cada articulacion a partir de la posición deseada del efector final. Pero la segunda parte es falsa: no es el mismo calculo sencillo que la cinematica directa. La cinematica directa es fácil porque va de los ángulos conocidos a la posición con fórmulas trigonometricas directas. La cinematica inversa es mucho mas difícil porque va al revés, puede tener múltiples soluciones o ninguna, y muchas veces requiere métodos numéricos o geometricos complejos. Por eso en los proyectos del curso se suele evitar calcularla y en su lugar se programan las posiciones con prueba y error. En la robótica profesional la cinematica se formaliza con los parámetros de Denavit-Hartenberg, un método sistematico para describir la geometría de cualquier brazo articulado. No necesitas dominarlo para este curso, pero es bueno saber que existe porque es el lenguaje común de los ingenieros en robótica. Lo importante aquí es la intuicion: ir de ángulos a posición es directo, pero ir de posición a ángulos es un rompecabezas con varias soluciones posibles.

RESPUESTA 45: Una secuencia valida seria la siguiente. Primero, el brazo gira su base hasta quedar frente al cubo sobre la mesa. Segundo, baja el hombro y el codo para acercar la pinza al cubo y la cierra para sujetarlo con firmeza. Tercero, levanta el brazo con el cubo sujeto para separarlo de la mesa sin arrastrarlo. Cuarto, gira la base hacia el lado donde esta la caja manteniendo el cubo en el aire. Quinto, baja el brazo sobre la caja, abre la pinza para soltar el cubo dentro y sube de nuevo a una posición de reposo. Cada paso usa los 3 grados de libertad: la base para orientarse, el hombro y el codo para la altura y el alcance, y la pinza como efector final. Cualquier secuencia equivalente que recoja, traslade y suelte el cubo de forma segura es valida. En un diseño real también debes considerar la fuerza de la pinza para no aplastar el cubo ni dejarlo caer, y la trayectoria para no golpear la mesa al moverse. Los robots industriales planean rutas que evitan obstáculos y optimizan el tiempo de ciclo. Para tu proyecto basta con mover un eje a la vez y verificar visualmente cada paso antes de encadenarlos en la secuencia completa.

RESPUESTA 46: La respuesta correcta es la opción b). En la comunicación serial TX significa transmisión y RX significa recepción: el pin TX de un dispositivo envía los datos y el pin RX del otro los recibe. Por eso al conectar dos dispositivos se cruzan los cables: el TX de uno va al RX del otro y viceversa, para que lo que uno transmite el otro lo escuche. La opción a) es incorrecta porque no son tipos de batería. La opción c) es incorrecta porque la alimentación va por VCC y GND, no por TX y RX. La opción d) es incorrecta porque TX y RX son los pines físicos de la comunicación serial, no protocolos de Bluetooth. Cuando un solo puerto serial no alcanza, puedes usar la librería `SoftwareSerial` para crear puertos seriales adicionales en pines digitales comunes. Esto es útil si necesitas el puerto principal para depurar con la computadora y otro para el HC-05 al mismo tiempo. Ten en cuenta que los puertos por software son menos robustos a altas velocidades, así que reservarlos para 9600 baudios es una buena práctica.

RESPUESTA 47: La respuesta correcta es la opción b). El HC-05 es un módulo Bluetooth que se conecta al puerto serial del Arduino y permite comunicarlo de forma inalámbrica con un teléfono celular o una computadora. Con él puedes enviar comandos desde una app móvil para controlar tu robot a distancia, como encender LEDs, mover motores o dirigir un carrito. La opción a) es incorrecta porque el sensor de distancia por ultrasonido es el HC-SR04, no el HC-05. La opción c) es incorrecta porque no es un servomotor. La opción d) es incorrecta porque no es una pantalla.

RESPUESTA 48: Verdadero. La velocidad de transmisión, medida en baudios, indica cuántos símbolos por segundo se envían, y ambos dispositivos deben usar exactamente la misma para entenderse. Si el Arduino transmite a 9600 baudios pero el otro dispositivo escucha a 115200, los tiempos de cada bit no coinciden y lo que se recibe son caracteres basura sin sentido. Por eso la primera regla al usar `Serial.begin()` es poner el mismo valor en ambos extremos de la comunicación. Este es también el error más común cuando el monitor serial muestra símbolos raros: casi siempre es un desajuste de baudios. Las velocidades más comunes son 9600 baudios para módulos sencillos como el HC-05 y 115200 para comunicación rápida con la computadora. El monitor serial del Arduino IDE tiene un menú desplegable para elegir la velocidad, y debe coincidir con el valor de `Serial.begin()` de tu programa. Si ves símbolos extraños en el monitor, el noventa por ciento de las veces el problema es este desajuste.

RESPUESTA 49: Esta pregunta es abierta y se califica según la calidad de la propuesta. Un buen proyecto final debe describir que robot se va a construir, por ejemplo un carrito evasor de obstáculos controlado por Bluetooth, un brazo robótico que clasifica objetos por color o una casa domótica con sensores. Debe listar los sensores que usará, como

el HC-SR04 para distancia, los IR para línea, el LDR para luz o el DHT11 para temperatura y humedad, y los actuadores, como motores DC con L298N o servomotores. Además debe explicar como integra las 10 unidades: la estructura y propósito del robot, el Arduino como controlador, la electrónica básica del circuito, cada sensor aplicado, el control de motores, los servos si lleva articulaciones, la navegación autónoma y la comunicación inalámbrica con el HC-05. Se valora que el proyecto sea realizable con los componentes del curso y que la explicación demuestre comprensión de cada bloque. Un buen proyecto final también incluye documentación: un diagrama del circuito, fotos del armado y un video corto del robot funcionando. Presentar tu trabajo de forma ordenada demuestra que no solo armaste piezas, sino que entendiste el sistema completo. Muchos estudiantes suben sus proyectos a internet y reciben retroalimentación de la comunidad, lo que acelera muchísimo el aprendizaje.

RESPUESTA 50: El programa debe iniciar la comunicación serial a 9600 baudios, revisar si hay datos disponibles con `Serial.available()`, leer el carácter con `Serial.read()` y comparar: si es '1' enciende el LED y si es '0' lo apaga. Una solución válida es la siguiente:

```
const int ledPin = 13;

void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  if (Serial.available() > 0) {
    char comando = Serial.read();
    if (comando == '1') {
      digitalWrite(ledPin, HIGH);
    } else if (comando == '0') {
      digitalWrite(ledPin, LOW);
    }
  }
}
```

Es importante comparar con el carácter '1' entre comillas simples y no con el número 1, porque lo que llega por Bluetooth es texto. El HC-05 conectado al puerto serial entrega los datos directo a Serial, así que no se necesita ninguna librería extra. Cualquier variante equivalente que encienda y apague con esos dos caracteres es válida. El HC-05 se puede configurar con comandos AT para cambiar su nombre, su código de emparejamiento y su velocidad, lo que es

util si varios compañeros usan módulos iguales en el mismo salón. Para entrar al modo AT se mantiene presionado su botón al energizarlo. En el uso normal del curso no necesitas cambiar nada: de fábrica ya viene a 9600 baudios y listo para emparejarse con el celular.

ÍNDICE

PRÓLOGO	4
Qué es este libro	4
El robot que vas a construir	4
Materiales que necesitas	5
Seguridad: léeme antes de conectar nada	5
Cómo estudiar este libro	6
Un consejo de Esteban	7
UNIDAD 1: FUNDAMENTOS DE LA ROBÓTICA	8
Explicación detallada	10
Ejemplos paso a paso	20
Práctica	23
Resumen de la unidad	31
UNIDAD 2: PRIMEROS PASOS CON ARDUINO	33
Introducción	33
Explicación detallada	35
Ejemplos paso a paso	46
Práctica	55
Resumen de la unidad	64
UNIDAD 3: ELECTRÓNICA BÁSICA PARA ROBÓTICA	66
Introducción	66
Explicación detallada	69
Ejemplos paso a paso	79
Práctica	85
Resumen de la unidad	89
UNIDAD 4: SENSORES I - DISTANCIA ULTRASONICA	91
Introducción	91
Explicación detallada	93
Ejemplos paso a paso	101
Práctica	110
Resumen de la unidad	120
UNIDAD 5: SENSORES II - LUZ, PRESENCIA Y TEMPERATURA	122
Explicación detallada	125
Ejemplos paso a paso	133
Práctica	142
Resumen de la unidad	152

UNIDAD 6: MOTORES DC Y CONTROL DE VELOCIDAD	154
Explicación detallada	156
Ejemplos paso a paso	163
Práctica	174
Resumen de la unidad	183
UNIDAD 7: SERVOMOTORES Y MOVIMIENTO PRECISO	185
Introducción	185
Explicación detallada	187
Ejemplos paso a paso	195
Práctica	203
Resumen de la unidad	213
UNIDAD 8: COMPORTAMIENTOS AUTÓNOMOS	215
Explicación detallada	217
Ejemplos paso a paso	225
Práctica	238
Resumen de la unidad	245
UNIDAD 9: BRAZOS ROBÓTICOS	247
Explicación detallada	250
Ejemplos paso a paso	260
Práctica	269
Resumen de la unidad	279
UNIDAD 10: COMUNICACIÓN INALÁMBRICA Y PROYECTO FINAL	281
Explicación detallada	283
Ejemplos paso a paso	291
Práctica	302
Resumen de la unidad	311
EXAMEN FINAL	312
Respuestas y soluciones	317

Curso Completo de Robótica

De los fundamentos a los comportamientos autónomos: Arduino, electrónica, sensores, motores y servomotores, hasta construir un robot que se mueve solo y un proyecto final con control inalámbrico. 10 unidades con explicación detallada, ejemplos paso a paso, prácticas con solución y examen final.

Colibrí Editorial EGA

lucasegaacademy.com

Hecho en México