# Junior Robotics Kit

**STEMverse Junior Robotics Kit - Official Build Manual**

*20 Missions · From Your First Circuit to a Full Assist-Bot*

## Introduction to the Junior Robotics Kit

Welcome to your build manual for the STEMverse Junior Robotics Kit. Inside are 20 missions that take you from your very first circuit to a fully working robot. Every mission is a real, working device, reframed around a problem it actually solves: a streetlight that saves energy, a cane that helps someone navigate safely, a rover that avoids obstacles on its own. Nothing here is "just for practice", every circuit you build is something a real engineer might build too, just at a smaller scale.

Work through the missions in order. Each one reuses a skill from the mission before it, so by the time you reach Mission #20, you'll have picked up circuit wiring, sensor logic, motor control, and robot design without ever facing a giant leap. If something doesn't work on the first try, that's normal, just check your wiring against the diagram first, then check your code line by line. Debugging is half of what real engineers do all day.

### Never Seen Before: Electro-Mechanics

This kit contains various electronics modules and sensors to aid you in building real projects and real solutions. The primary basis is the Arduino, which is supported by all other components. Apart from electronics components, this kit gives you 10 bars, 4 L-brackets, and 10 bolt-and-nut pairs. You've probably seen bars, brackets, bolts, and nuts before, construction toy systems have been built around them for a century. What makes Structura different is that every part in this kit is 3D-printed, which means the kit isn't fixed to what's in the box. If you need or design a part we don't currently make, something like a longer bar, an angled connector, a completely custom bracket or casing for a robot,  we can print it and ship it as an add-on. The kit grows with what you imagine, not just with what we shipped in it.

## Full Kit Contents

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |
| USB Cable | 1 |
| Breadboard | 1 |
| Jumper Wires | 2 strips |
| 220Ω Resistor | 10 |
| 10kΩ Resistor | 10 |
| LED (Red) | 1 |
| LED (Green) | 1 |
| LED (Blue) | 1 |
| LED (Yellow) | 1 |
| RGB LED | 1 |
| Push Button | 4 |
| Buzzer | 1 |
| LDR (Light Sensor) | 1 |
| IR Sensor | 1 |

| | |
|---|---|
| Ultrasonic Sensor | 1 |
| Servo Motor (SG90) | 1 |
| DC Motor | 2 |
| Motor Driver (L298N) | 1 |
| Battery Holder | 1 |
| Wheel | 2 |
| Caster Wheel | 1 |
| Chassis Plate | 1 |
| Bolt | 10 |
| Nut | 10 |
| L Bracket | 4 |
| Bar | 10 |

## Before You Start

Do this once before your first mission. It takes about 10 minutes.

### 1. Install the Arduino IDE

The Arduino IDE is the free program you'll use to write and upload code. Go to arduino.cc/en/software on a laptop and download the installer for your operating system (Windows/Mac/Linux). Install it like any normal app.

Step: Download & install the free Arduino IDE
(the program you'll write code in)

**arduino.cc/en/software**

**[ Download Arduino IDE ]**

**Install like any normal app**

## 2. Install the USB Driver (only if needed)

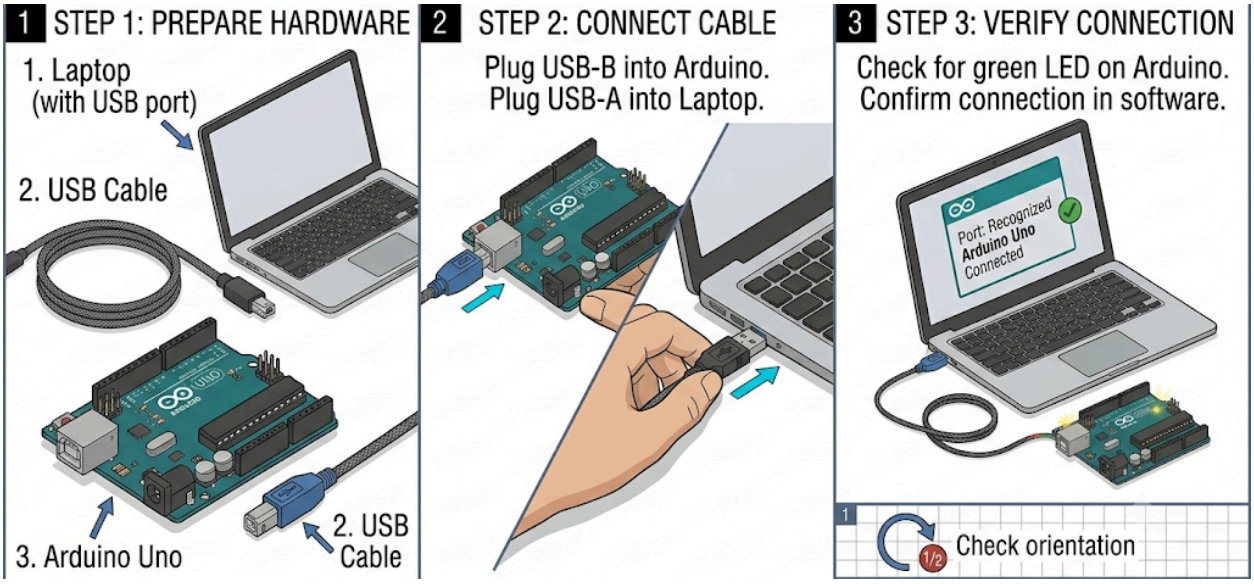
Most Arduino Uno boards use a chip called CH340 to talk to your laptop over USB. If your laptop doesn't recognize the board in Step 4 below, search online for "CH340 driver download", install it, then restart your laptop. You only need to do this once, and only if Step 4 doesn't work.

*Only needed once, if Step 'Select Port' shows nothing*

**If your laptop doesn't detect the Arduino:**
**Install the 'CH340' USB driver**
**(search: CH340 driver download)**
**Restart your laptop after installing**

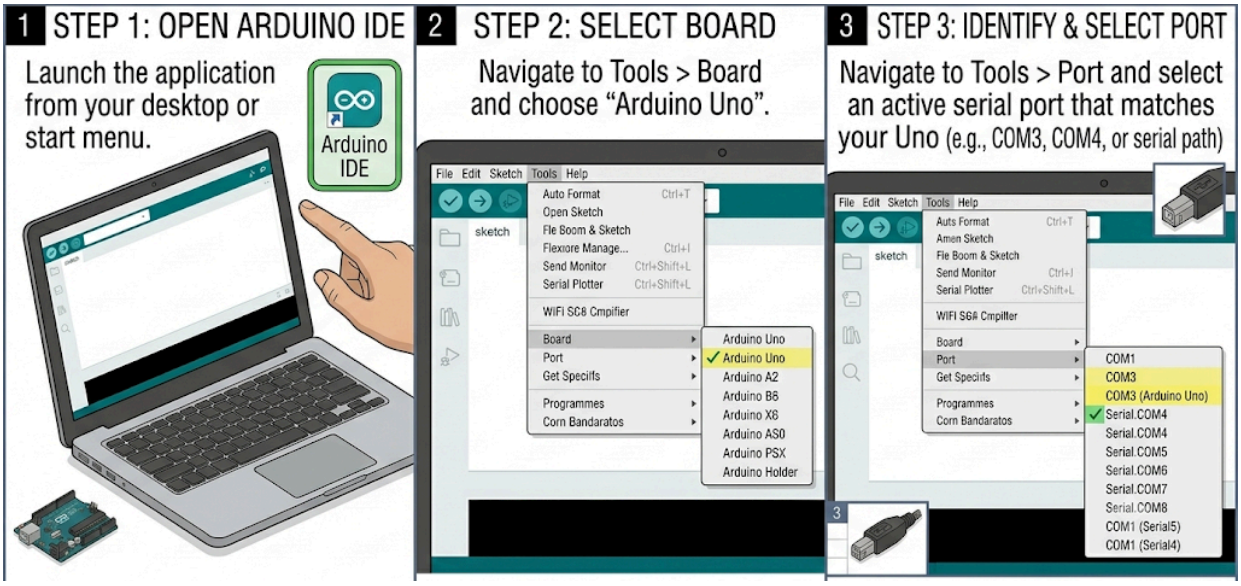
## 3. Plug In Your Arduino

Connect the USB cable from your Arduino Uno to your laptop's USB port.



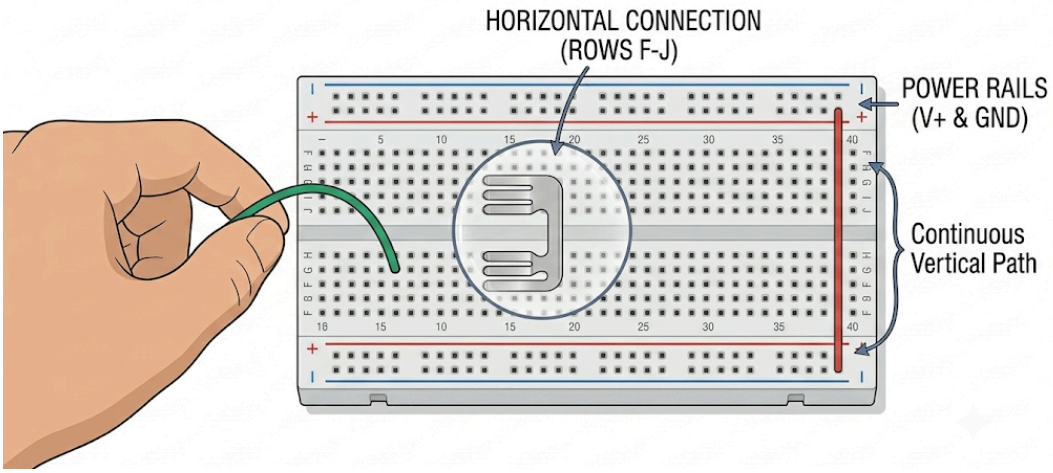

## 4. Select Your Board and Port

Open the Arduino IDE. Go to Tools > Board and choose "Arduino Uno". Then go to Tools > Port and select the port that appeared after you plugged in the USB cable.



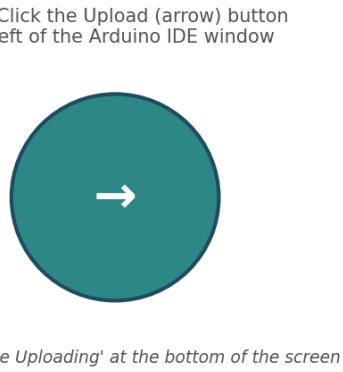

## 5. Understand Your Breadboard

A breadboard lets you connect components without soldering. Each short row of 5 holes is electrically connected. The two long rows on the edge (usually marked red + and blue −) are your power rails, used to share power/ground across the whole board.



## 6. Uploading Code - Every Mission

For every mission in this manual: type or paste the code into a new sketch, then click the Upload button (the arrow icon, top-left). Wait for “Done Uploading” at the bottom of the window before touching your circuit.

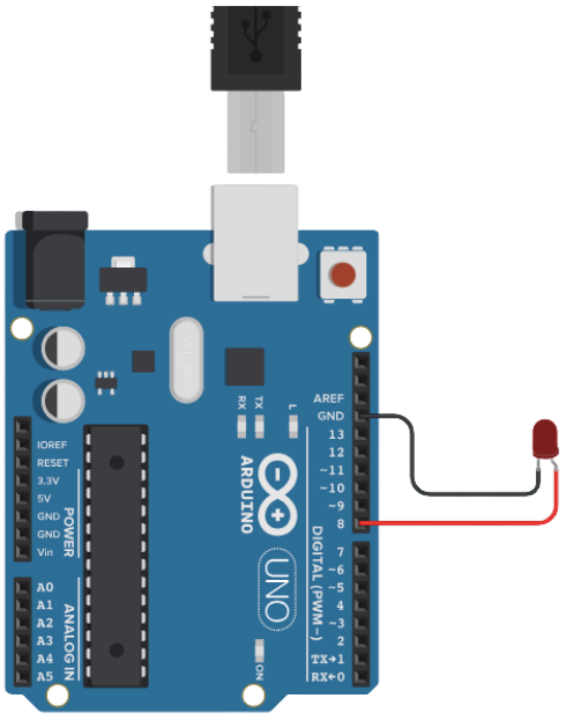



## A Note for Parents & Facilitators

Every mission below is designed to be attempted by a first-time builder, age 6 and up, with adult help for wiring. Each mission reuses skills from earlier missions, so we recommend building them in order rather than skipping ahead - Mission #9's distance-sensing code, for example, is reused directly in Missions #10, #13, #16, #19 and #20.

# Mission #1 - Medication Reminder

*"Never Miss a Dose" - Timed Health Reminder Light*




## Why This Project Matters

Missed medication doses are a real problem in elderly and patient care - this ties to SDG 3 (Good Health & Well-Being). You will build a device that lights up automatically at set intervals, just like real medical reminder devices used at home. Along the way you'll learn how a microcontroller keeps track of time on its own, without you pressing anything - a core building block used in almost every future mission.

## Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |
| USB Cable | 1 |
| Breadboard | 1 |
| Jumper Wires | 2 |
| LED (Red) | 1 |
| 220Ω Resistor | 1 |

## Step-by-Step Build

1. Complete the ‘Getting Started’ section first (install the Arduino IDE, plug in the USB cable, select your board and port).

2. Push the LED into the breadboard. Note the LED has two legs - one longer (positive) and one shorter (negative).

3. Connect a 220Ω resistor from the LED's long leg to a free breadboard row.

4. Run a jumper wire from that row to pin 8 on the Arduino.

5. Run a jumper wire from the LED's short leg to a GND (ground) pin on the Arduino.

6. Open a new sketch in the Arduino IDE and type the code below exactly as shown.

7. Click Upload and wait for ‘Done Uploading’.

8. Watch the LED - it should turn on for 3 seconds every 10 seconds. That 10-second gap represents a dose-reminder interval (in a real device this would be set to hours).

## The Code

```
const int ledPin = 8;

unsigned long reminderInterval = 10000; // 10 sec = one "dose time" (sped up
for demo)

unsigned long lastReminder = 0;

void setup() {

 pinMode(ledPin, OUTPUT);

}

void loop() {

 if (millis() - lastReminder >= reminderInterval) {

   digitalWrite(ledPin, HIGH);  // turn LED ON

   delay(3000);                 // stay on for 3 sec

   digitalWrite(ledPin, LOW);   // turn LED OFF

   lastReminder = millis();     // reset the timer

 }

}
```

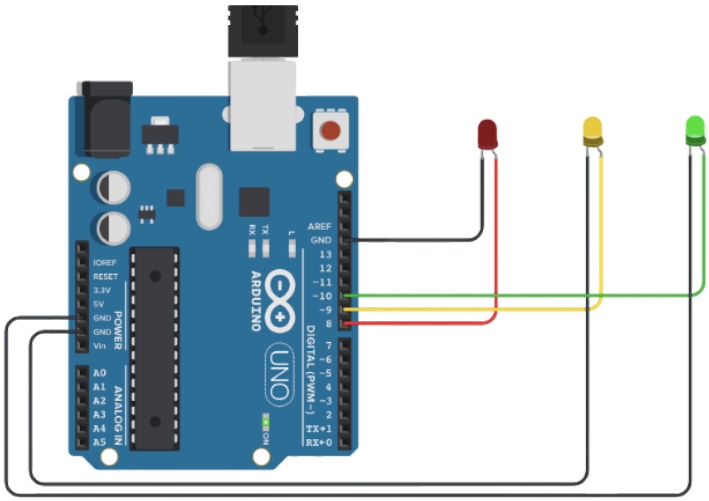
## What This Code Does

- `pinMode(ledPin, OUTPUT)` tells the Arduino “pin 8 will send power out”, not read anything in.

- `millis()` returns how many milliseconds have passed since the Arduino turned on — this is how the board tracks time without a clock chip.
- The `if` statement checks whether enough time has passed since the last reminder before doing anything - this is called non-blocking timing, and you'll reuse this exact pattern in later missions.
- `lastReminder = millis()` resets the stopwatch so the next reminder is timed from now, not from power-on.

# Mission #2 - Traffic Signal Sequencer

*Road-Safety Signal Timing Model*




## Why This Project Matters

Traffic signals are a core piece of city infrastructure (SDG 11: Sustainable Cities & Communities) - badly timed signals cause real accidents. In this mission you'll model the exact red-yellow-green timing logic that real traffic engineers design, and practice sequencing multiple outputs in the correct order.

## Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |
| USB Cable | 1 |

| | |
|---|---|
| Breadboard | 1 |
| Jumper Wires | 3 |
| LED (Red) | 1 |
| LED (Yellow) | 1 |
| LED (Green) | 1 |
| 220Ω Resistor | 3 |

## Step-by-Step Build

1. Push all 3 LEDs into the breadboard, spaced apart.
2. Wire a 220Ω resistor from each LED's long leg to its own breadboard row.
3. Connect the Red LED's row to pin 8, Yellow to pin 9, Green to pin 10.
4. Connect all 3 LEDs' short legs to a shared GND pin.
5. Upload the code below.
6. Watch the sequence: Green (go) → Yellow (wait) → Red (stop) → repeat, just like a real signal.

## The Code

```
int redPin = 8, yellowPin = 9, greenPin = 10;

void setup() {

 pinMode(redPin, OUTPUT);
```

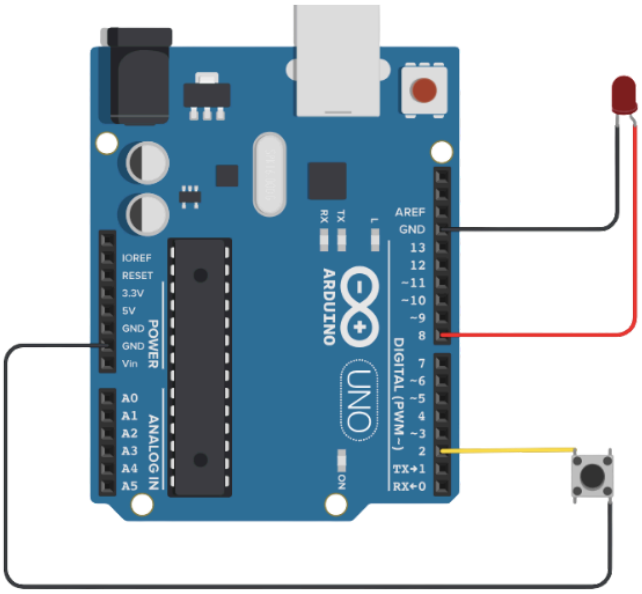
```
 pinMode(yellowPin, OUTPUT);
 pinMode(greenPin, OUTPUT);
}
void loop() {
 digitalWrite(greenPin, HIGH); delay(4000); digitalWrite(greenPin, LOW);
 digitalWrite(yellowPin, HIGH); delay(1500); digitalWrite(yellowPin, LOW);
 digitalWrite(redPin, HIGH); delay(4000); digitalWrite(redPin, LOW);
}
```

## What This Code Does

- Three separate pins are declared because each LED needs its own independent output.
- The `loop()` runs forever, so once it finishes Red it jumps straight back to Green - this is what makes a real traffic light cycle continuously.
- The `delay()` numbers directly control how long each phase lasts - try changing them and notice how it changes the whole rhythm of the signal.

## Mission #3 - Reaction Time Sensor

*Reflex & Response-Time Health Screener*




### Why This Project Matters

Reaction-time testing is a real tool doctors and sports scientists use to screen reflex decline (SDG 3). This mission introduces your first digital input - reading a button - and combines it with timing to measure a real human response.

### Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |
| USB Cable | 1 |

| | |
|---|---|
| Breadboard | 1 |
| Jumper Wires | 3 |
| LED (Red) | 1 |
| 220Ω Resistor | 1 |
| Push Button | 1 |

## Step-by-Step Build

1. Wire the LED (with resistor) to pin 8 and GND, same as Mission #1.
2. Push the button into the breadboard. Wire one side to pin 2, and the other side to GND.
3. Upload the code below and open the Serial Monitor (magnifying-glass icon, top right of the IDE).
4. Wait for the LED to light up at a random moment, then press the button as fast as you can.
5. Read your reaction time in milliseconds in the Serial Monitor. Try to beat your own score!

## The Code

```
int ledPin = 8, buttonPin = 2;

unsigned long startTime;

void setup() {

 pinMode(ledPin, OUTPUT);
```

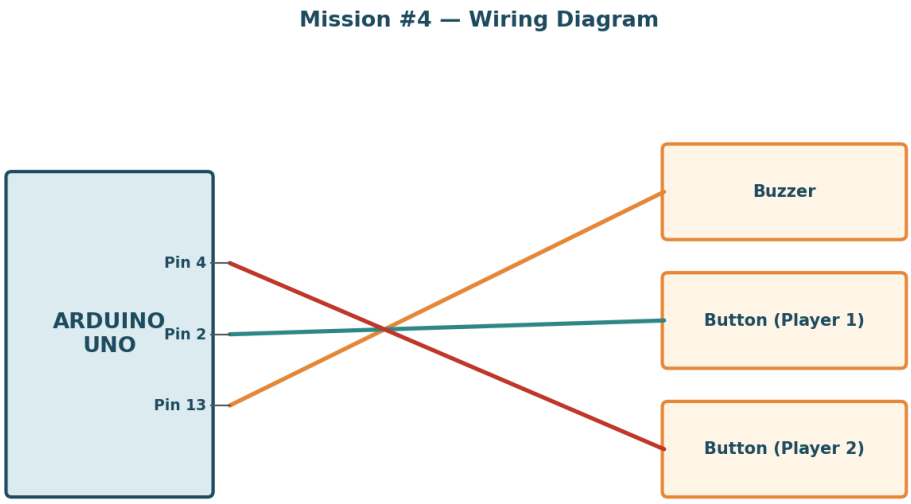
```
 pinMode(buttonPin, INPUT_PULLUP);
 Serial.begin(9600);
 randomSeed(analogRead(A5));
}
void loop() {
 delay(random(2000, 6000));      // random wait, so you can't predict it
 digitalWrite(ledPin, HIGH);
 startTime = millis();
 while (digitalRead(buttonPin) == HIGH) { }   // wait until pressed
 unsigned long reaction = millis() - startTime;
 digitalWrite(ledPin, LOW);
 Serial.print("Reaction time: ");
 Serial.print(reaction);
 Serial.println(" ms");
 delay(2000);
}
```

## What This Code Does

- `INPUT_PULLUP` means the button reads HIGH when NOT pressed, and LOW when pressed - this is the standard safe way to wire a button.
- `random(2000, 6000)` picks a random wait time each round so you can't anticipate when the LED will light up - this is what makes it a fair test.
- The `while` loop pauses the program exactly until the button is pressed, then the very next line calculates how long that took.
- `Serial.print` sends text back to your laptop screen so you can read your score.

# Mission #4 - Quiz-Buzzer System

*First-Press-Wins Classroom Quiz Buzzer*




## Why This Project Matters

This is a real tool used in classrooms and quiz competitions worldwide (SDG 4: Quality Education). You'll learn how to read two inputs at once and let the code decide who acted first — a foundational "input arbitration" concept used in games and competitive electronics.

## Bill of Materials — This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |
| USB Cable | 1 |

| | |
|---|---|
| Breadboard | 1 |
| Jumper Wires | 3 |
| Buzzer | 1 |
| Push Button | 2 |

## Step-by-Step Build

1. Wire the buzzer's positive leg to pin 13 and negative leg to GND.
2. Wire Player 1's button between pin 2 and GND.
3. Wire Player 2's button between pin 4 and GND.
4. Upload the code and open the Serial Monitor.
5. Have two players race to press their button — the buzzer sounds and the Serial Monitor announces the winner.

## The Code

```
int buzzerPin = 13, btn1 = 2, btn2 = 4;

void setup() {

 pinMode(buzzerPin, OUTPUT);

 pinMode(btn1, INPUT_PULLUP);

 pinMode(btn2, INPUT_PULLUP);

 Serial.begin(9600);
```

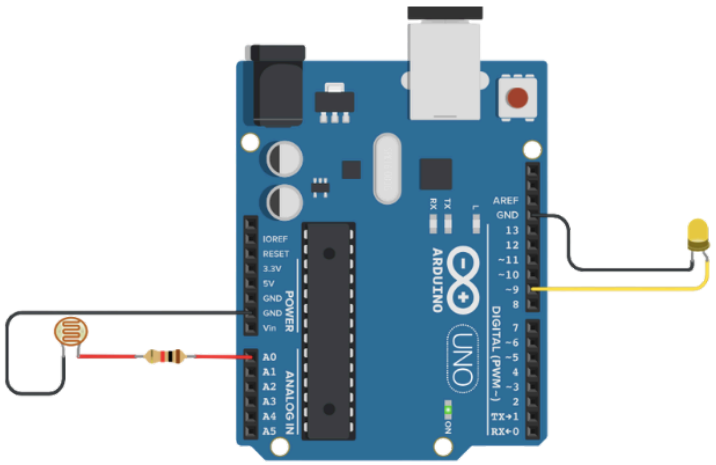
```
}

void loop() {

 if (digitalRead(btn1) == LOW) {

   tone(buzzerPin, 1000, 500);

   Serial.println("Player 1 buzzed in first!");

   delay(2000);

 } else if (digitalRead(btn2) == LOW) {

   tone(buzzerPin, 1500, 500);

   Serial.println("Player 2 buzzed in first!");

   delay(2000);

 }

}
```

## What This Code Does

- `tone(pin, frequency, duration)` plays a sound on the buzzer at a chosen pitch — 1000Hz vs 1500Hz gives each player a distinct sound.
- Using `if / else if` means only ONE branch can run per loop — whichever button is checked and found pressed first “wins” that instant.
- The `delay(2000)` after a win pauses the game for 2 seconds so the buzz doesn't repeat instantly.

## Mission #5 — Energy-Saving Streetlight

*Automatic Dusk-to-Dawn Street Lamp*




### Why This Project Matters

Municipal streetlights that turn on automatically at dusk save huge amounts of electricity (SDG 7: Affordable & Clean Energy). This mission introduces your first analog sensor reading - turning a light level into a number your code can react to.

### Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |
| USB Cable | 1 |
| Breadboard | 1 |

| | |
|---|---|
| Jumper Wires | 3 |
| LDR (Light Sensor) | 1 |
| 10kΩ Resistor | 1 |
| LED (Yellow) | 1 |
| 220Ω Resistor | 1 |

## Step-by-Step Build

1. Build an LDR voltage-divider: connect the LDR from 5V to pin A0, then a 10kΩ resistor from pin A0 to GND.
2. Wire the Yellow LED (with 220Ω resistor) to pin 9 and GND.
3. Upload the code below.
4. Cover the LDR with your hand to simulate nighttime - the LED should turn on.
5. Uncover it - the LED should turn off, just like a real streetlight sensing daylight.

## The Code

```
int ledPin = 9, ldrPin = A0;

int threshold = 500;

void setup() {

 pinMode(ledPin, OUTPUT);

}
```

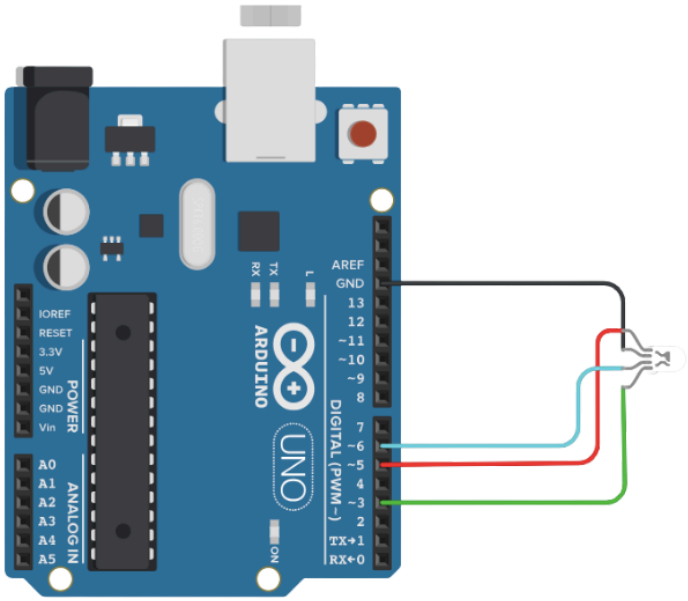
```
void loop() {
 int lightLevel = analogRead(ldrPin);
 if (lightLevel < threshold) {
   digitalWrite(ledPin, HIGH);   // dark -> lamp ON
 } else {
   digitalWrite(ledPin, LOW);    // bright -> lamp OFF
 }
 delay(200);
}
```

## What This Code Does

- `analogRead()` gives a number from 0 to 1023 representing how much light is hitting the LDR - more light usually gives a higher number.
- The `threshold` value is the cutoff point that decides “dark enough to turn the light on” - try changing it if your room is unusually bright or dim.
- This exact on/off-by-threshold pattern is the basis for almost every automatic sensor mission later in this kit.

# Mission #6 - Dimmable Lamp

*Smooth-Fade Mood/Reading Lamp*




## Why This Project Matters

Dimmable LED lighting reduces energy use compared to always-on bulbs (SDG 7). This mission introduces analog output (PWM) - instead of just ON/OFF, you'll control brightness itself.

## Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |

| | |
|---|---|
| USB Cable | 1 |
| Breadboard | 1 |
| Jumper Wires | 3 |
| RGB LED | 1 |
| 220Ω Resistor | 3 |

## Step-by-Step Build

1. Push the RGB LED into the breadboard. It has 4 legs - the longest is usually the common ground.
2. Wire the Red leg (through a 220Ω resistor) to pin 5, Green to pin 6, #003c71 to pin 3.
3. Wire the common (longest) leg to GND.
4. Upload the code below and watch the lamp smoothly fade from off to full brightness and back.

## The Code

```
int ledPin = 9, ldrPin = A0;

int threshold = 500;

void setup() {

 pinMode(ledPin, OUTPUT);

}

void loop() {
```

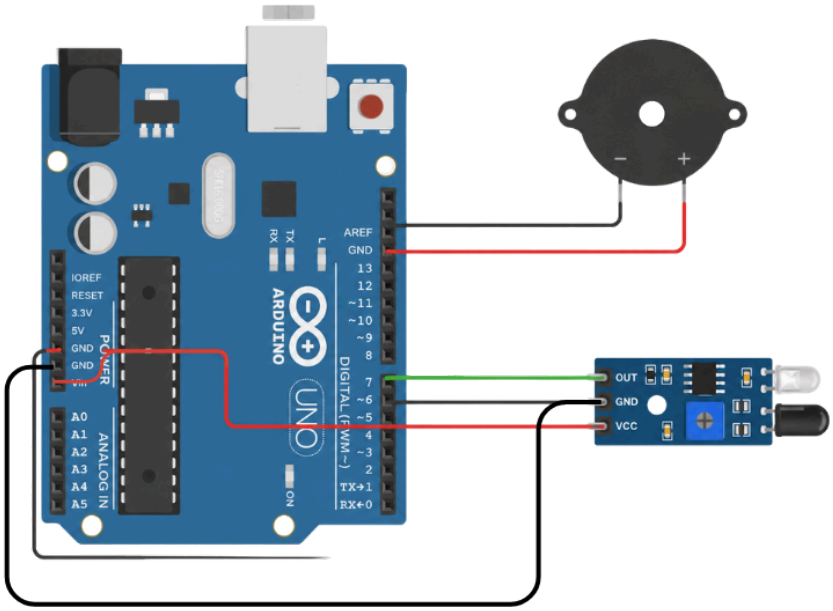
```
  int lightLevel = analogRead(ldrPin);

  if (lightLevel < threshold) {

    digitalWrite(ledPin, HIGH);    // dark -> lamp ON

  } else {

    digitalWrite(ledPin, LOW);     // bright -> lamp OFF

  }

  delay(200);

}
```

## What This Code Does

- `analogWrite()` sends a value from 0 (off) to 255 (full brightness) - this is called PWM, and it's how the Arduino fakes an “in-between” voltage.
- The first `for` loop counts up from 0 to 255, fading the lamp brighter one step at a time.
- The second `for` loop counts back down, fading it dim again - together they create a smooth breathing effect.

# Mission #7 - Touchless Hand-Wash Alert

*Hygiene Reminder Station*




## Why This Project Matters

Hand hygiene is one of the simplest, highest-impact public health interventions there is (SDG 3 and SDG 6: Clean Water & Sanitation). You'll learn to read a digital sensor module (IR) instead of a simple button - the same signal type used in many real hygiene-station products.

## Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |

| | |
|---|---|
| USB Cable | 1 |
| Breadboard | 1 |
| Jumper Wires | 2 |
| IR Sensor | 1 |
| Buzzer | 1 |

## Step-by-Step Build

1. Connect the IR sensor's signal (OUT) pin to Arduino pin 7, and its VCC/GND to 5V/GND.
2. Connect the buzzer's positive leg to pin 13 and negative to GND.
3. Upload the code below.
4. Wave your hand in front of the IR sensor - the buzzer should chirp as a wash reminder.
5. Note: some IR modules read LOW when they detect an object, others read HIGH - if it doesn't react, flip `LOW` to `HIGH` in the code.

## The Code

```
int irPin = 7, buzzerPin = 13;

void setup() {

 pinMode(irPin, INPUT);

 pinMode(buzzerPin, OUTPUT);

}
```

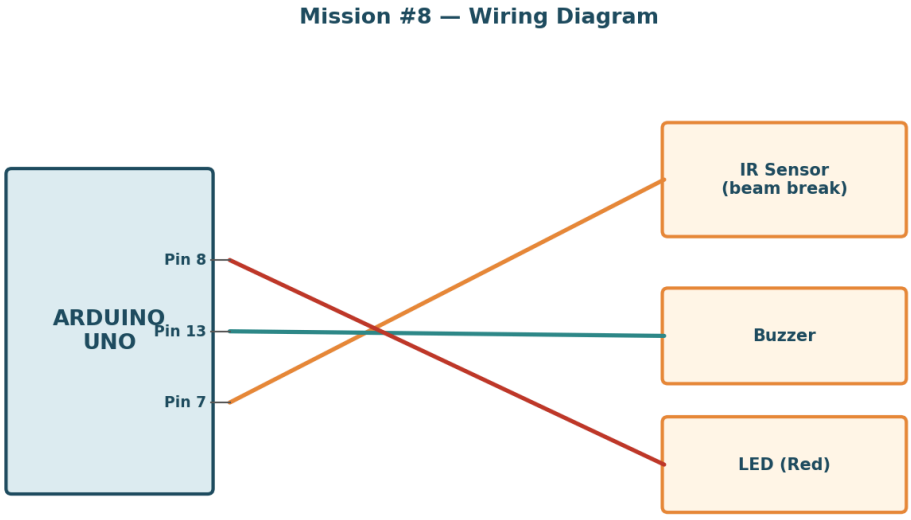
```
void loop() {
 if (digitalRead(irPin) == LOW) {    // hand detected
   tone(buzzerPin, 1000, 300);
   delay(1000);
 }
}
```

## What This Code Does

- IR sensor modules output a simple digital signal - LOW or HIGH - depending on whether they detect an object, which is why we can read it just like a button.
- The `delay(1000)` after a detection stops the buzzer from chirping continuously while your hand is still in front of it.

# Mission #8 - Perimeter Intrusion Alarm

*Doorway/Entry Security Alert*




## Why This Project Matters

Simple beam-break sensors are the basis of real entry-level security systems used in homes and shops (SDG 11: safer communities). This mission combines a sensor with two outputs at once (sound + light) to build a full alarm response.

## Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |

| | |
|---|---|
| USB Cable | 1 |
| Breadboard | 1 |
| Jumper Wires | 3 |
| IR Sensor | 1 |
| Buzzer | 1 |
| LED (Red) | 1 |
| 220Ω Resistor | 1 |

## Step-by-Step Build

1. Wire the IR sensor to pin 7 (same as Mission #7).
2. Wire the buzzer to pin 13 and the Red LED (with resistor) to pin 8.
3. Position the IR sensor across a “doorway” (a gap between two boxes works fine).
4. Upload the code and walk through the gap to trigger the alarm.

## The Code

```
int irPin = 7, buzzerPin = 13, ledPin = 8;

void setup() {

 pinMode(irPin, INPUT);

 pinMode(buzzerPin, OUTPUT);
```

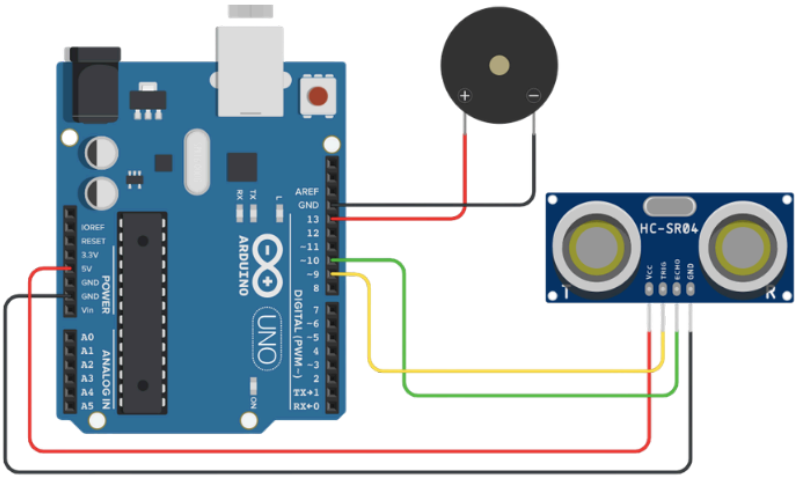
```
 pinMode(ledPin, OUTPUT);
}
void loop() {
 if (digitalRead(irPin) == LOW) {
   digitalWrite(ledPin, HIGH);
   tone(buzzerPin, 2000);
 } else {
   digitalWrite(ledPin, LOW);
   noTone(buzzerPin);
 }
}
```

## What This Code Does

- This mission reuses Mission #7's sensor-reading logic but now drives TWO outputs together instead of one - the same `if/else` shape, just more actions inside it.
- `noTone()` explicitly stops any sound - without it, `tone()` would keep playing even after the person has passed.

# Mission #9 - Smart-Parking Distance Sensor

*Reverse-Parking Proximity Alert*




## Why This Project Matters

This is the exact sensing principle used in real car reverse-parking sensors (SDG 11: safer, smarter transport infrastructure). You'll take your first analog-style distance reading using an ultrasonic sensor, which works like sonar.

## Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |
| USB Cable | 1 |

| | |
|---|---|
| Breadboard | 1 |
| Jumper Wires | 4 |
| Ultrasonic Sensor | 1 |
| Buzzer | 1 |

## Step-by-Step Build

1. Connect the ultrasonic sensor's Trig pin to Arduino pin 9, and Echo pin to pin 10.
2. Connect its VCC to 5V and GND to GND.
3. Wire the buzzer to pin 13.
4. Upload the code below.
5. Move your hand slowly toward the sensor - the buzzer should beep faster the closer you get, just like a car's parking sensor.

## The Code

```
const int trigPin = 9, echoPin = 10, buzzerPin = 13;

void setup() {

 pinMode(trigPin, OUTPUT);

 pinMode(echoPin, INPUT);

 pinMode(buzzerPin, OUTPUT);

}
```

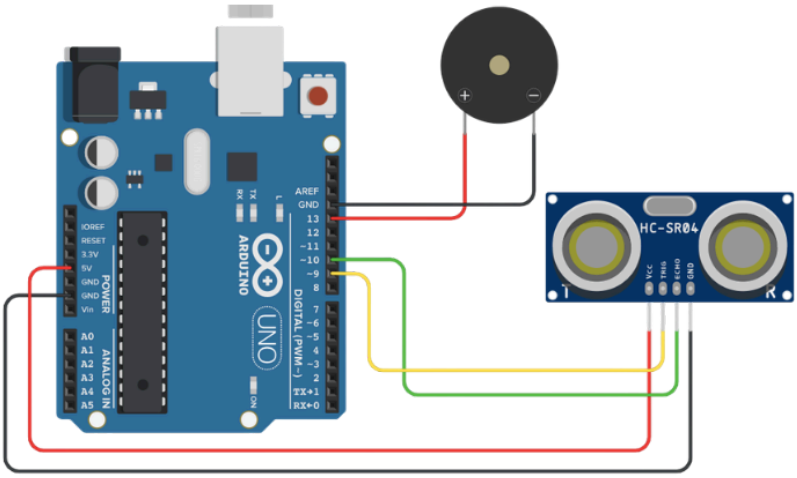
```
long readDistanceCM() {
 digitalWrite(trigPin, LOW); delayMicroseconds(2);
 digitalWrite(trigPin, HIGH); delayMicroseconds(10);
 digitalWrite(trigPin, LOW);
 long duration = pulseIn(echoPin, HIGH);
 return duration * 0.034 / 2;
}
void loop() {
 long distance = readDistanceCM();
 if (distance < 30) {
   int beepGap = map(distance, 0, 30, 50, 500);
   tone(buzzerPin, 1000, 50);
   delay(beepGap);
 } else {
   delay(300);
 }
}
```

## What This Code Does

- The ultrasonic sensor sends out a sound pulse (`Trig`) and measures how long it takes to bounce back (`Echo`) - that time is converted into a distance in centimeters.
- `map()` takes the distance and rescales it into a beep-gap time - close objects get a short gap (fast beeping), far objects get a long gap (slow beeping).
- This distance-reading function (`readDistanceCM`) is reused as-is in Missions #10, #13, #16 and #19 - once you understand it here, you understand it everywhere.

## Mission #10 - Assistive White-Cane Obstacle Alert

*Navigation Aid for the Visually Impaired*




### Why This Project Matters

This mission adapts real assistive-navigation technology (SDG 3 and SDG 10: Reduced Inequalities) - the same distance-to-feedback concept used in Mission #9, now reframed as a wearable aid that could genuinely help someone navigate safely.

### Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |
| USB Cable | 1 |
| Breadboard | 1 |

| | |
|---|---|
| Jumper Wires | 4 |
| Ultrasonic Sensor | 1 |
| Buzzer | 1 |

## Step-by-Step Build

1. Wire the ultrasonic sensor and buzzer exactly as in Mission #9.
2. Mount the sensor at the end of a stick or rod (or just hold it out in front of you) to simulate a white cane.
3. Upload the same code as Mission #9 - no changes needed, since the concept transfers directly.
4. Walk toward an obstacle and notice the buzzer beeping faster as you approach, giving early warning before contact.

## The Code

```
// Same code as Mission #9 - reused here for a wearable/hand-held use case.

const int trigPin = 9, echoPin = 10, buzzerPin = 13;

void setup() {

 pinMode(trigPin, OUTPUT);

 pinMode(echoPin, INPUT);

 pinMode(buzzerPin, OUTPUT);

}
```

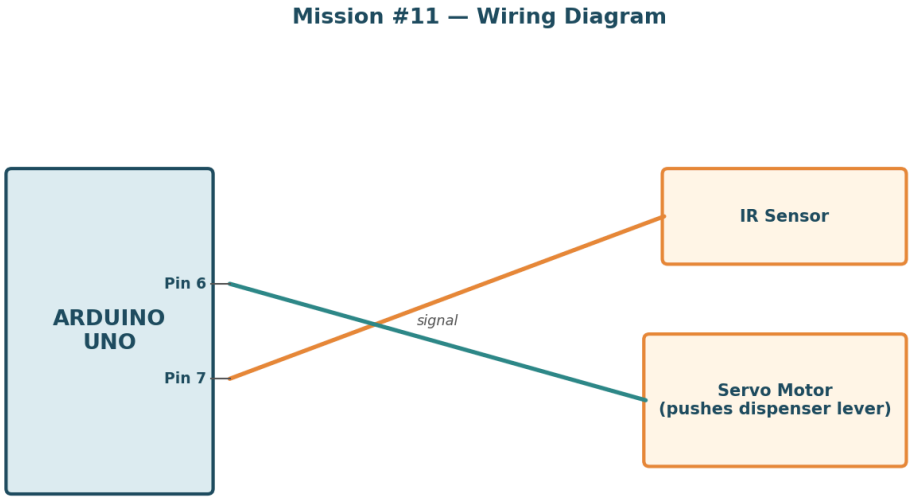
```
long readDistanceCM() {
 digitalWrite(trigPin, LOW); delayMicroseconds(2);
 digitalWrite(trigPin, HIGH); delayMicroseconds(10);
 digitalWrite(trigPin, LOW);
 long duration = pulseIn(echoPin, HIGH);
 return duration * 0.034 / 2;
}
void loop() {
 long distance = readDistanceCM();
 if (distance < 100) {
   int beepGap = map(distance, 0, 100, 50, 600);
   tone(buzzerPin, 1200, 50);
   delay(beepGap);
 } else {
   delay(300);
 }
}
```

## What This Code Does

- Only the detection range changed (100cm instead of 30cm) - a cane needs to warn much earlier than a parking sensor does.
- This is a good example of how the SAME sensor logic can serve two completely different real purposes just by changing the numbers, not the concept.

# Mission #11 - Touchless Hand-Sanitizer Dispenser

*Hygiene Station — Contactless Dispenser*




## Why This Project Matters

Touchless dispensers reduce surface contact and germ spread (SDG 3, SDG 6). This is your first mission combining a sensor with a servo motor - your first physical actuator that moves to a specific angle.

## Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |

| | |
|---|---|
| USB Cable | 1 |
| Breadboard | 1 |
| Jumper Wires | 3 |
| IR Sensor | 1 |
| Servo Motor (SG90) | 1 |

## Step-by-Step Build

1. Wire the IR sensor to pin 7 as in Mission #7.
2. Connect the Servo's signal wire to pin 6, and its power/ground wires to 5V/GND.
3. Mount the servo's arm so it can push down on a soft bottle's pump when it rotates.
4. Upload the code below.
5. Place your hand under the sensor — the servo should push the pump, then return to rest.

## The Code

```
#include <Servo.h>

Servo dispenserServo;

int irPin = 7;

void setup() {

 dispenserServo.attach(6);

 dispenserServo.write(0);
```

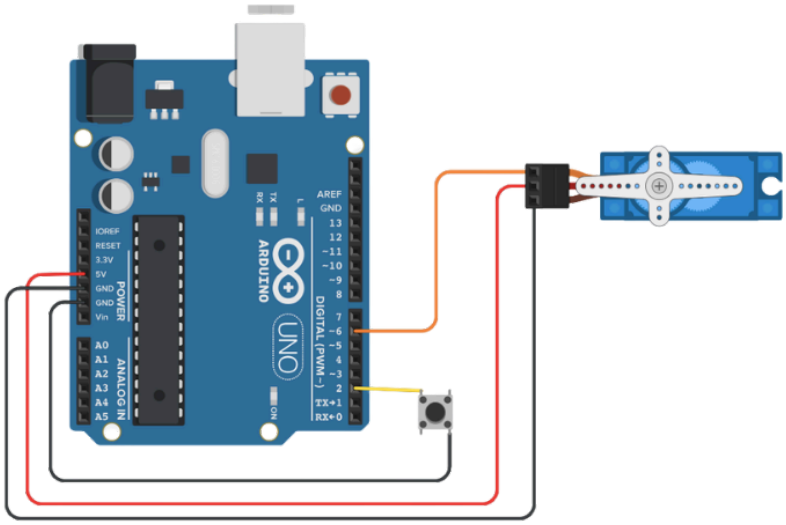
```
 pinMode(irPin, INPUT);
}
void loop() {
 if (digitalRead(irPin) == LOW) {
   dispenserServo.write(90);   // push the pump
   delay(500);
   dispenserServo.write(0);   // return to rest
   delay(1000);
 }
}
```

## What This Code Does

- `#include <Servo.h>` loads the built-in library that knows how to talk to servo motors - you'll use this in every servo mission from here on.
- `.attach(6)` tells the code which pin the servo's signal wire is connected to.
- `.write(angle)` moves the servo to a specific angle in degrees (0–180) - `90` here represents “push” and `0` represents “rest”.

## Mission #12 - Medicine / Pet-Feeder Dispenser

*Timed Dose or Feeding Dispenser*




### Why This Project Matters

Automatic dispensers help with medication adherence for the elderly and consistent feeding for pets (SDG 3). You'll reuse the servo skill from Mission #11, but now trigger it from a button instead of a sensor.

### Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |
| USB Cable | 1 |
| Breadboard | 1 |

| | |
|---|---|
| Jumper Wires | 3 |
| Push Button | 1 |
| Servo Motor (SG90) | 1 |

## Step-by-Step Build

1. Wire the push button between pin 2 and GND, using `INPUT_PULLUP` (same pattern as Mission #3).
2. Wire the servo's signal wire to pin 6.
3. Position the servo arm under a small container of food/pills so a 90° turn tips it slightly.
4. Upload the code and press the button to dispense one dose.

## The Code

```
#include <Servo.h>

Servo feederServo;

int buttonPin = 2;

void setup() {

 feederServo.attach(6);

 feederServo.write(0);

 pinMode(buttonPin, INPUT_PULLUP);

}
```

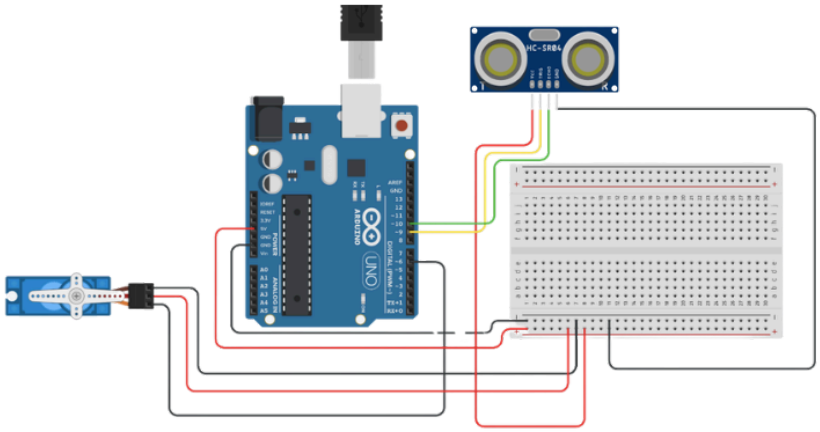
```
void loop() {
 if (digitalRead(buttonPin) == LOW) {
   feederServo.write(90);
   delay(700);
   feederServo.write(0);
   delay(1000);
 }
}
```

## What This Code Does

- This is Mission #11's servo logic, but the trigger changed from an IR sensor to a button - notice how little of the code had to change.
- The `delay(1000)` at the end prevents one button press from accidentally triggering multiple dispenses in a row.

# Mission #13 - Automatic Gate / Barrier Arm

*Smart Parking Barrier*



## Why This Project Matters

Automatic barriers are standard in smart-parking and gated facilities (SDG 11). This mission combines two skills you already have - ultrasonic distance sensing (Mission #9) and servo actuation (Mission #11) - into one automated system.

## Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |
| USB Cable | 1 |
| Breadboard | 1 |
| Jumper Wires | 4 |

| | |
|---|---|
| Ultrasonic Sensor | 1 |
| Servo Motor (SG90) | 1 |

## Step-by-Step Build

1. Wire the ultrasonic sensor exactly as in Mission #9 (Trig=9, Echo=10).
2. Wire the servo's signal wire to pin 6.
3. Attach a straw or thin strip to the servo arm to act as the barrier arm.
4. Upload the code and bring an object close to the sensor — the barrier should lift.

## The Code

```
#include <Servo.h>

Servo gateServo;

const int trigPin = 9, echoPin = 10;

long readDistanceCM() {

 digitalWrite(trigPin, LOW); delayMicroseconds(2);

 digitalWrite(trigPin, HIGH); delayMicroseconds(10);

 digitalWrite(trigPin, LOW);

 long duration = pulseIn(echoPin, HIGH);

 return duration * 0.034 / 2;

}
```

```
void setup() {
 gateServo.attach(6);
 gateServo.write(0);
 pinMode(trigPin, OUTPUT);
 pinMode(echoPin, INPUT);
}
void loop() {
 long d = readDistanceCM();
 if (d < 15) {
   gateServo.write(90);   // barrier up
 } else {
   gateServo.write(0);    // barrier down
 }
 delay(200);
}
```

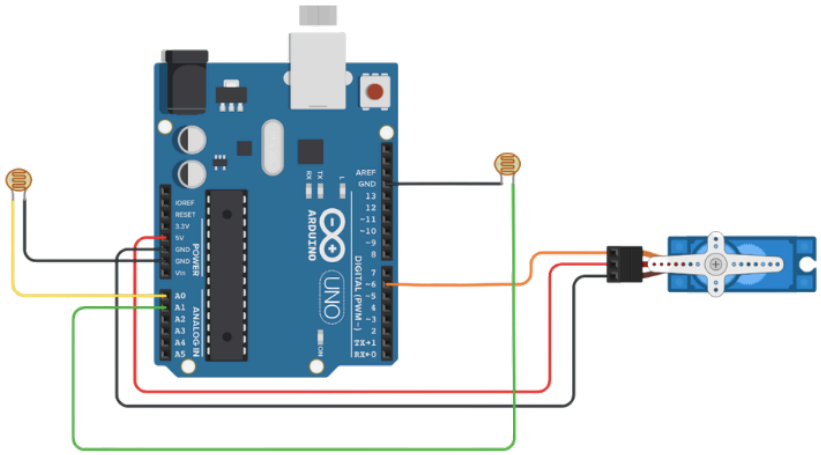
## What This Code Does

- Notice this mission's code is really just Mission #9's distance function PLUS Mission #11's servo control - combining two known skills into one new device.

- The barrier stays up only while an object is within 15cm; move away and it lowers automatically.

# Mission #14 - Solar-Tracking Mini Panel

*Sun-Following Panel Mount*



## Why This Project Matters

Real solar farms use sun-tracking mounts to increase energy capture by up to 25% (SDG 7: Clean Energy, SDG 13: Climate Action). This mission introduces comparing two sensor readings at once, which is a more advanced logic step than anything used so far.

## Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |
| USB Cable | 1 |
| Breadboard | 1 |

| | |
|---|---|
| Jumper Wires | 4 |
| LDR (Light Sensor) | 2 |
| 10kΩ Resistor | 2 |
| Servo Motor (SG90) | 1 |

## Step-by-Step Build

1. Build two LDR voltage-dividers (like Mission #5), one on pin A0 (“left”) and one on pin A1 (“right”).
2. Wire the servo's signal wire to pin 6, and mount a small piece of cardboard (the “panel”) on the servo arm.
3. Upload the code below.
4. Shine a torch/phone light more on one LDR than the other — the panel should slowly turn toward the brighter side.

## The Code

```
#include <Servo.h>

Servo panelServo;

int ldrLeft = A0, ldrRight = A1;

int pos = 90;

void setup() {

 panelServo.attach(6);
```

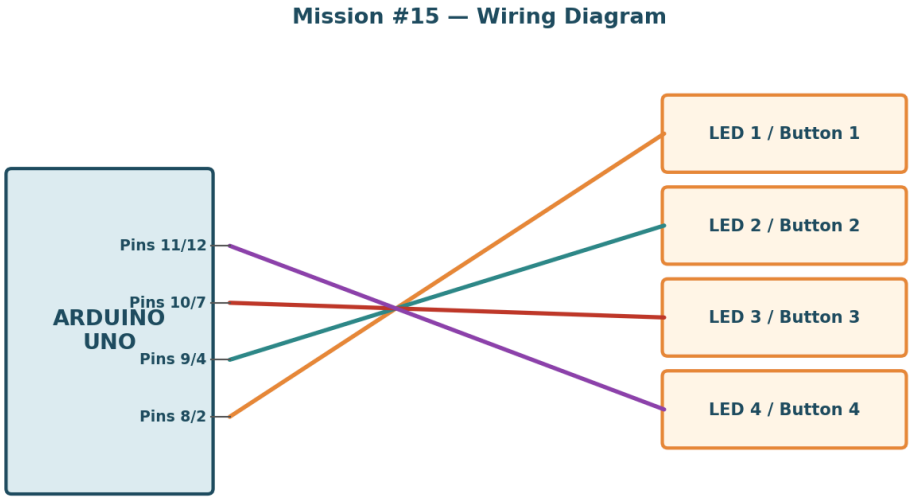
```
 panelServo.write(pos);
}
void loop() {
 int left = analogRead(ldrLeft);
 int right = analogRead(ldrRight);
 if (abs(left - right) > 50) {
   if (left > right && pos < 170) pos++;
   else if (right > left && pos > 10) pos--;
   panelServo.write(pos);
 }
 delay(50);
}
```

## What This Code Does

- `abs(left - right)` measures how different the two light readings are - small differences are ignored so the panel doesn't jitter constantly.
- The `if / else if` decides which direction is brighter and nudges `pos` by 1 degree at a time toward it - this gradual stepping (rather than jumping straight there) is what makes the motion smooth.

# Mission #15 - Cognitive Memory-Recall Trainer

*Simon-Says Style Memory & Focus Game*




## Why This Project Matters

Sequence-recall games like this are used in real cognitive therapy and early memory-decline screening, including for Alzheimer's-related care (SDG 3). This mission introduces your first "state machine" - code that remembers a growing sequence and checks the player against it.

## Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |

| | |
|---|---|
| USB Cable | 1 |
| Breadboard | 1 |
| Jumper Wires | 8 |
| LED (Red) | 1 |
| LED (Green) | 1 |
| LED (Blue) | 1 |
| LED (Yellow) | 1 |
| 220Ω Resistor | 4 |
| Push Button | 4 |

## Step-by-Step Build

1. Wire all 4 LEDs (with resistors) to pins 8, 9, 10, 11.
2. Wire all 4 buttons (with INPUT_PULLUP) to pins 2, 4, 7, 12 — one button next to each matching LED.
3. Upload the code below.
4. Watch the LED sequence play, then press the buttons back in the same order.
5. Each round the sequence grows by one — see how long you can remember!

## The Code

```
int leds[4] = {8, 9, 10, 11};
```

```
int buttons[4] = {2, 4, 7, 12};

int sequence[100];

int level = 0;

void setup() {

 for (int i = 0; i < 4; i++) {

   pinMode(leds[i], OUTPUT);

   pinMode(buttons[i], INPUT_PULLUP);

 }

 randomSeed(analogRead(A5));

}

void playSequence() {

 for (int i = 0; i <= level; i++) {

   digitalWrite(leds[sequence[i]], HIGH);

   delay(400);

   digitalWrite(leds[sequence[i]], LOW);

   delay(200);

 }
```

```
}

bool getPlayerInput() {

 for (int i = 0; i <= level; i++) {

   int pressed = -1;

   while (pressed == -1) {

     for (int b = 0; b < 4; b++) {

       if (digitalRead(buttons[b]) == LOW) {

         pressed = b;

         digitalWrite(leds[b], HIGH); delay(200); digitalWrite(leds[b],
LOW);

       }

     }

   }

   if (pressed != sequence[i]) return false;

 }

 return true;

}

void loop() {
```

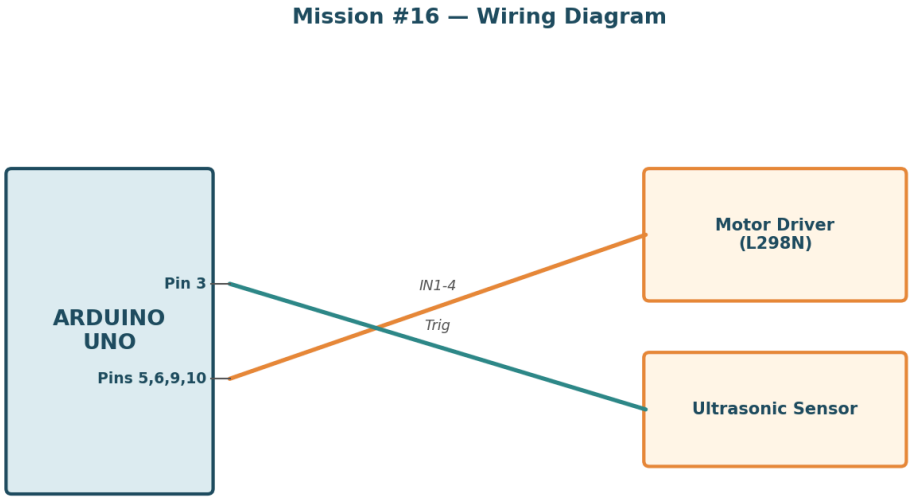
```
  sequence[level] = random(0, 4);
  playSequence();
  if (!getPlayerInput()) {
    level = 0;      // wrong answer -> restart
    delay(1000);
  } else {
    level++;        // correct -> sequence gets longer
    delay(500);
  }
}
```

## What This Code Does

- The `sequence[]` array is the game's memory - each round adds one more random step to it.
- `playSequence()` lights up every step in the sequence so far, in order, so the player can watch and memorize it.
- `getPlayerInput()` waits for a button press at each step and compares it to what the sequence expects - one wrong press ends the round.
- This up-front code is longer than earlier missions, but it's built entirely from ideas you already know: arrays of pins, `for` loops, and button reading.

# Mission #16 - Obstacle-Avoiding Assistive Rover

*Autonomous Navigation Rover (assistive-mobility concept)*




*Motor Driver output wires connect to the 2 DC motors on the chassis*

## Why This Project Matters

This is the base logic behind real assistive robotic wheelchairs and warehouse delivery robots (SDG 3, SDG 10). This is your first mission driving actual DC motors through a motor driver - real robotics, not just a stationary circuit.

## Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |

| | |
|---|---|
| USB Cable | 1 |
| Breadboard | 1 |
| Jumper Wires | 6 |
| Ultrasonic Sensor | 1 |
| DC Motor | 2 |
| Motor Driver (L298N) | 1 |
| Chassis Plate | 1 |
| Wheel | 2 |
| Caster Wheel | 1 |
| Battery Holder | 1 |
| Bolt | 4 |
| Nut | 4 |

## Step-by-Step Build

1. Bolt the chassis plate together with the L-brackets, then bolt the caster wheel to the front-underside and the 2 DC motors to the rear, using the bolts/nuts.
2. Push the 2 wheels onto the DC motor shafts.
3. Mount the motor driver (L298N) on the chassis. Connect each DC motor's two wires to OUT1/OUT2 and OUT3/OUT4 on the driver.

4. Wire the driver's IN1, IN2, IN3, IN4 pins to Arduino pins 5, 6, 9, 10. Leave ENA/ENB jumpers ON (full speed).
5. Mount the ultrasonic sensor facing forward. Wire Trig to pin 3, Echo to pin 4.
6. Connect the battery holder to the motor driver's power input (not the Arduino's USB — motors need their own power).
7. Upload the code below.
8. Place the rover on the floor and put your hand in front of it - it should stop, turn, then keep moving forward.

## The Code

```
const int IN1 = 5, IN2 = 6, IN3 = 9, IN4 = 10;

const int trigPin = 3, echoPin = 4;

long readDistanceCM() {

 digitalWrite(trigPin, LOW); delayMicroseconds(2);

 digitalWrite(trigPin, HIGH); delayMicroseconds(10);

 digitalWrite(trigPin, LOW);

 long duration = pulseIn(echoPin, HIGH);

 return duration * 0.034 / 2;

}

void forward()    { digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW); }
```

```
void stopMotors() { digitalWrite(IN1, LOW);  digitalWrite(IN2, LOW);
digitalWrite(IN3, LOW);  digitalWrite(IN4, LOW); }

void turnRight()  { digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
digitalWrite(IN3, LOW);  digitalWrite(IN4, HIGH); }

void setup() {

 pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT); pinMode(IN3, OUTPUT);
pinMode(IN4, OUTPUT);

 pinMode(trigPin, OUTPUT); pinMode(echoPin, INPUT);

}

void loop() {

 long distance = readDistanceCM();

 if (distance > 20) {

   forward();

 } else {

   stopMotors();

   delay(200);

   turnRight();

   delay(400);
```

```
  }

}
```

## What This Code Does

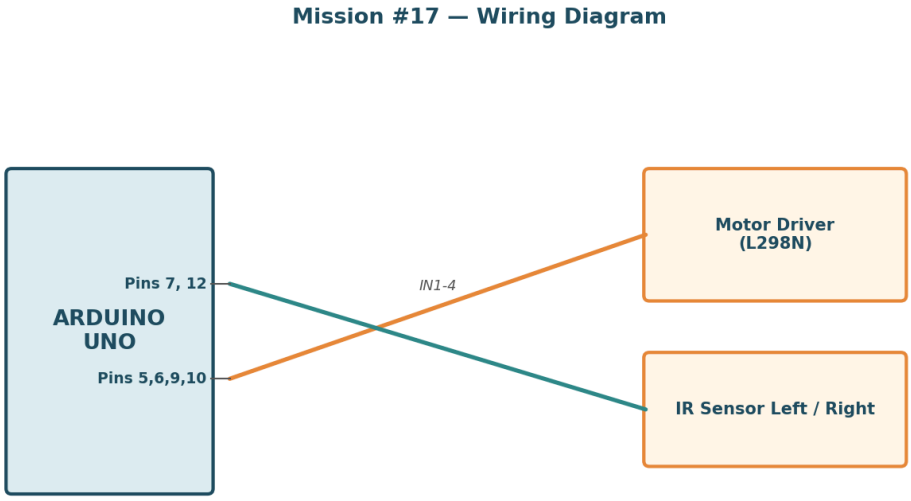
- Each motor direction is a small function (`forward`, `stopMotors`, `turnRight`) that sets 4 pins at once - grouping repeated actions into a function like this keeps `loop()` short and readable.

- The rover reuses the exact `readDistanceCM()` function from Mission #9 - this is the payoff of learning it early.

- The logic is simple: keep going forward while the path is clear (>20cm); if something's close, stop, turn briefly, then try forward again.

# Mission #17 - Line-Tracing Hospital-Delivery Cart

*Autonomous Delivery Cart (hospital/warehouse concept)*




*IR sensors mounted facing down at the front of the chassis, above the line*

## Why This Project Matters

Autonomous delivery carts that follow a fixed path are already used in real hospitals and warehouses to move supplies without staff time (SDG 3, SDG 9: Industry & Infrastructure). This mission introduces reading two sensors at once to make a steering decision.

## Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |

| | |
|---|---|
| USB Cable | 1 |
| Breadboard | 1 |
| Jumper Wires | 6 |
| IR Sensor | 2 |
| DC Motor | 2 |
| Motor Driver (L298N) | 1 |
| Chassis Plate | 1 |
| Wheel | 2 |
| Caster Wheel | 1 |
| Battery Holder | 1 |

## Step-by-Step Build

1. Reuse the same rover base as Mission #16 (chassis, motors, wheels, driver, battery).
2. Mount 2 IR sensors facing down at the front of the chassis, a few cm apart, just above a dark tape line on the floor.
3. Wire the left IR sensor to pin 7, and the right IR sensor to pin 12.
4. Upload the code below.
5. Lay a strip of dark tape on a light floor in a curved path, place the rover on it, and watch it follow the line.

## The Code

```
const int IN1 = 5, IN2 = 6, IN3 = 9, IN4 = 10;

const int irLeft = 7, irRight = 12;

void forward()    { digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW); }

void stopMotors() { digitalWrite(IN1, LOW);  digitalWrite(IN2, LOW);
digitalWrite(IN3, LOW);  digitalWrite(IN4, LOW); }

void turnLeft()   { digitalWrite(IN1, LOW);  digitalWrite(IN2, LOW);
digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW); }

void turnRight()  { digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
digitalWrite(IN3, LOW);  digitalWrite(IN4, LOW); }

void setup() {

 pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT); pinMode(IN3, OUTPUT);
pinMode(IN4, OUTPUT);

 pinMode(irLeft, INPUT); pinMode(irRight, INPUT);

}

void loop() {

 int left = digitalRead(irLeft);

 int right = digitalRead(irRight);

 if (left == LOW && right == LOW) forward();
```

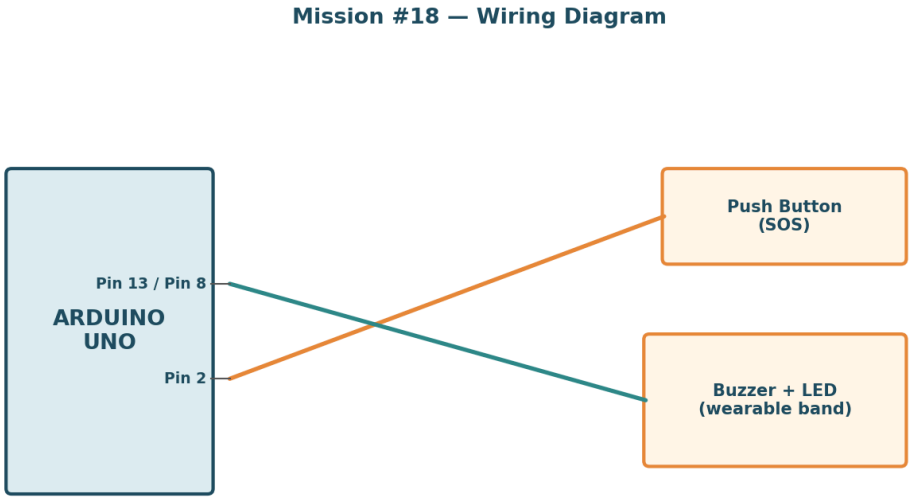
```
  else if (left == HIGH && right == LOW) turnLeft();
  else if (left == LOW && right == HIGH) turnRight();
  else stopMotors();
 }
```

## What This Code Does

- Most IR line-sensor modules read LOW over a dark line and HIGH over a light floor - if your rover behaves backwards, swap `LOW`/`HIGH` in the code.
- The 4 possible combinations of (left, right) readings map to 4 clear decisions: both-on-line = go straight, one side drifting off = steer back toward the line.
- This introduces reacting to TWO sensors together, which is a step up from Mission #16's single ultrasonic sensor.

# Mission #18 - Wearable Fall / SOS Alert

*Emergency Panic-Button Wearable*




## Why This Project Matters

Wearable panic buttons are real products used in elderly care to summon help quickly (SDG 3). This mission is intentionally simple by design — a wearable safety device should never be complicated to use, which is itself an important design lesson.

## Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |

| | |
|---|---|
| USB Cable | 1 |
| Breadboard | 1 |
| Jumper Wires | 3 |
| Push Button | 1 |
| Buzzer | 1 |
| LED (Red) | 1 |
| 220Ω Resistor | 1 |

## Step-by-Step Build

1. Wire the button between pin 2 and GND (INPUT_PULLUP).
2. Wire the buzzer to pin 13 and the Red LED (with resistor) to pin 8.
3. Tape or strap the button somewhere easy to press (wrist, table edge) to simulate a wearable.
4. Upload the code and press the button to trigger the alert pattern.

## The Code

```
int buttonPin = 2, buzzerPin = 13, ledPin = 8;

void setup() {

 pinMode(buttonPin, INPUT_PULLUP);

 pinMode(buzzerPin, OUTPUT);
```

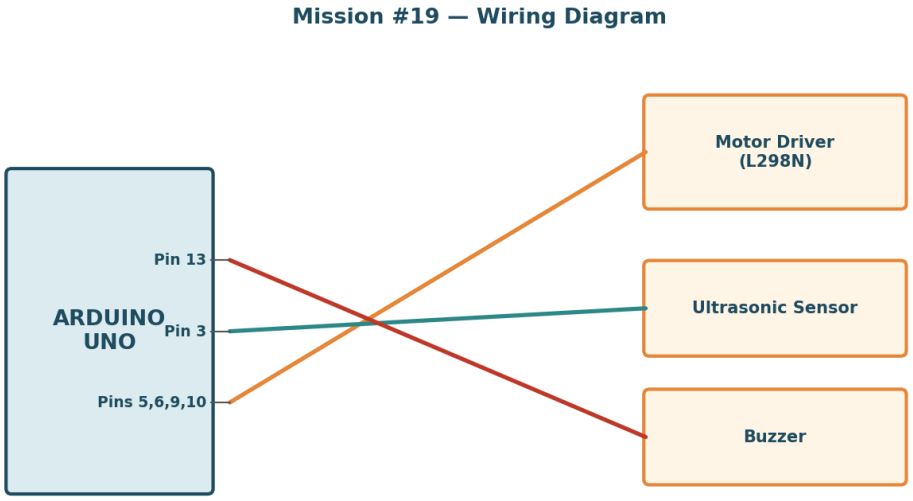
```
 pinMode(ledPin, OUTPUT);
}
void loop() {
 if (digitalRead(buttonPin) == LOW) {
   for (int i = 0; i < 5; i++) {
     digitalWrite(ledPin, HIGH); tone(buzzerPin, 2000); delay(200);
     digitalWrite(ledPin, LOW);  noTone(buzzerPin);      delay(200);
   }
 }
}
```

## What This Code Does

- The `for` loop repeats the flash-and-beep pattern exactly 5 times - a distinct, unmistakable alert pattern rather than a single beep that could be missed.
- Good assistive design is often about simplicity: one button, one clear unmistakable response - resist the urge to add more features here.

# Mission #19 - Autonomous Security Patrol Bot

*Perimeter Patrol & Alert Robot*




*Same rover base as Mission #16, with buzzer added for alerts*

## Why This Project Matters

Autonomous patrol robots are used in real large facilities to extend limited security staff coverage (SDG 11, SDG 16: Peace & Strong Institutions). This mission combines Mission #16's navigation with an active alarm response, showing how the same rover base can be repurposed for a different mission.

## Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |

| | |
|---|---|
| USB Cable | 1 |
| Breadboard | 1 |
| Jumper Wires | 7 |
| Ultrasonic Sensor | 1 |
| DC Motor | 2 |
| Motor Driver (L298N) | 1 |
| Buzzer | 1 |
| Chassis Plate | 1 |
| Wheel | 2 |
| Caster Wheel | 1 |
| Battery Holder | 1 |

## Step-by-Step Build

1. Reuse the exact rover build from Mission #16.
2. Add the buzzer to pin 13, positive leg to pin 13 and negative to GND.
3. Upload the code below.
4. Let the rover patrol an open area, when it detects an obstacle it should stop and sound an alert instead of just turning away.

## The Code

```
const int IN1 = 5, IN2 = 6, IN3 = 9, IN4 = 10;

const int trigPin = 3, echoPin = 4, buzzerPin = 13;

long readDistanceCM() {

 digitalWrite(trigPin, LOW); delayMicroseconds(2);

 digitalWrite(trigPin, HIGH); delayMicroseconds(10);

 digitalWrite(trigPin, LOW);

 long duration = pulseIn(echoPin, HIGH);

 return duration * 0.034 / 2;

}

void forward()    { digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW); }

void stopMotors() { digitalWrite(IN1, LOW);  digitalWrite(IN2, LOW);
digitalWrite(IN3, LOW);  digitalWrite(IN4, LOW); }

void setup() {

 pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT); pinMode(IN3, OUTPUT);
pinMode(IN4, OUTPUT);

 pinMode(trigPin, OUTPUT); pinMode(echoPin, INPUT);

 pinMode(buzzerPin, OUTPUT);
```

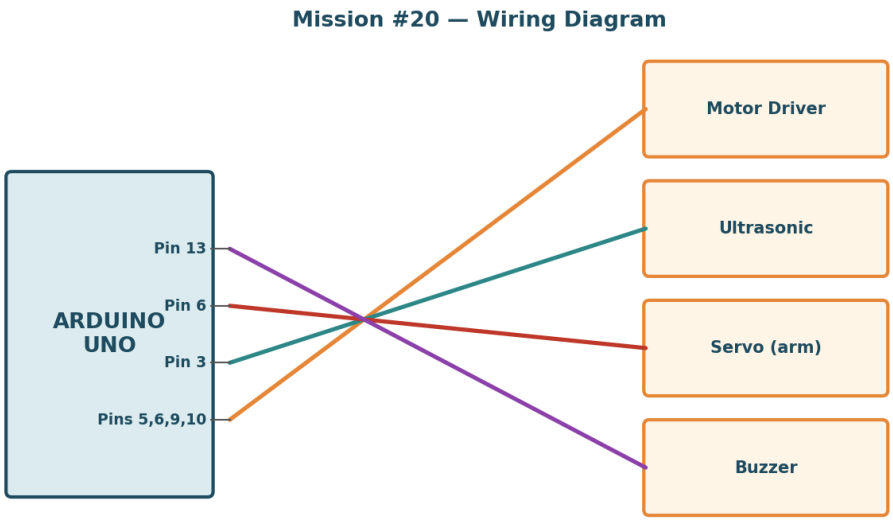
```
}

void loop() {

 long distance = readDistanceCM();

 if (distance > 20) {

   forward();

   noTone(buzzerPin);

 } else {

   stopMotors();

   tone(buzzerPin, 2000);

   delay(500);

 }

}
```

## What This Code Does

- Compare this to Mission #16 - the movement logic is almost identical, but the “react to an obstacle” branch now sounds an alarm instead of turning, because a patrol bot's job is to alert, not to solve the obstacle itself.
- This is a good example of reusing a robotics platform for a different mission just by changing what happens in one branch of the code.

# Mission #20 - Assist-Bot

*Capstone: Personal Home Assistive/Security Robot*




*Full integration build — combine all subsystems from previous missions on one chassis*

## Why This Project Matters

This capstone combines every subsystem you've built across the kit - sensing, sound, movement, and actuation - into one integrated assistive or security robot (SDG 3, SDG 10, SDG 11). This is the mission where you move from following instructions to designing your own solution, the final and most important skill in engineering.

## Bill of Materials - This Mission

| Component | Quantity |
|---|---|
| Arduino Uno | 1 |

| | |
|---|---|
| USB Cable | 1 |
| Breadboard | 1 |
| Jumper Wires | 8 |
| Ultrasonic Sensor | 1 |
| DC Motor | 2 |
| Motor Driver (L298N) | 1 |
| Servo Motor (SG90) | 1 |
| Buzzer | 1 |
| Chassis Plate | 1 |
| Wheel | 2 |
| Caster Wheel | 1 |
| Battery Holder | 1 |

## Step-by-Step Build

1. Start from the Mission #16/19 rover base (chassis, motors, driver, ultrasonic, battery).
2. Add a servo (pin 11) as a small ‘arm’ or flap on the front of the chassis.
3. Add a buzzer to pin 13 for alerts.
4. Upload the starter code below — it stops and “reaches out” with the arm plus alerts when something is close, otherwise it patrols forward.

5. Now personalize it: choose a real use case (elderly companion, home security, warehouse helper) and change the behaviour, sounds, or arm action to match your story.

6. Present your finished Assist-Bot and its use case to the class or at your next STEMverse exhibition.

## The Code

```
#include <Servo.h>

Servo armServo;

const int IN1 = 5, IN2 = 6, IN3 = 9, IN4 = 10;

const int trigPin = 3, echoPin = 4, buzzerPin = 13;

long readDistanceCM() {

 digitalWrite(trigPin, LOW); delayMicroseconds(2);

 digitalWrite(trigPin, HIGH); delayMicroseconds(10);

 digitalWrite(trigPin, LOW);

 long duration = pulseIn(echoPin, HIGH);

 return duration * 0.034 / 2;

}

void forward()    { digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW); }

void stopMotors() { digitalWrite(IN1, LOW);  digitalWrite(IN2, LOW);
digitalWrite(IN3, LOW);  digitalWrite(IN4, LOW); }
```

```
void setup() {
 pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT); pinMode(IN3, OUTPUT);
pinMode(IN4, OUTPUT);
 pinMode(trigPin, OUTPUT); pinMode(echoPin, INPUT);
 pinMode(buzzerPin, OUTPUT);
 armServo.attach(11);
 armServo.write(0);
}
void loop() {
 long distance = readDistanceCM();
 if (distance < 15) {
   stopMotors();
   tone(buzzerPin, 2000, 300);
   armServo.write(90);
   delay(500);
   armServo.write(0);
 } else {
   forward();
```

```
  }

}
```

## What This Code Does

- This starter code is deliberately a combination of things you've already built in Missions #9, #11 and #16 - nothing here is new syntax, only new combinations.

- The real assignment is what comes after: decide what your Assist-Bot is FOR, then change the numbers, add a mission, or bring in an extra sensor from earlier missions to make it truly yours.