# **Dawson College**

# **Electrical Engineering Technology Department**

# **Introduction to Internet of Things**

## **Project Name:**

## Smart Pet Feeder Alert System

## **Team Members:**

## Carlos Student (ID: 1234567), Bobby Student (ID: 7654321)

## May 13, 2025

## 2. Project Description

Meet Bobby, a playful cat whose owner often rushes out the door in the morning, juggling work, emails, and morning traffic. Like many pet owners, they do their best—but between meetings and errands, they sometimes forget one simple but important task: refilling Bobby’s food bowl.

To address this common oversight, we designed a **Smart Pet Feeder Alert System** that uses an **ultrasonic distance sensor** to monitor the food level in a pet’s bowl. When the system detects that the bowl is empty, it automatically activates an **LED** and **buzzer** to notify the owner.

It promotes responsible pet care, minimizes stress for the animal, and creates a simple, automated check-in system for food levels.

**How it works:**

* The **ultrasonic sensor** continuously measures the distance to the food surface.
* If the measured distance exceeds a defined threshold (indicating the bowl is low or empty), the **LED blinks** and the **buzzer sounds** to alert the user.
* A **manual override push button** is available to temporarily silence the buzzer without disabling the alert system entirely.

**Final assembly diagram:**

Use any drawing tool of your preference. This is meant to have students think about practical implications for their design. The drawing below was done on PowerPoint. Use the diagram to help with your project description.

A black background with a black background and a black wall

AI-generated content may be incorrect.

Figure 1 - Project diagram.

## 3. Circuit Diagram

**Inputs:**

* Ultrasonic sensor (HC-SR04)
* Push button (Pull-Up)

**Outputs:**

* LED (alert)
* Buzzer (auditory alert)

Below is a description of the hardware connections, which are also shown graphically using Fritzing (or a similar CAD tool). Note how schematic is **neat, labelled and colour-coded,** so the instructor can follow the circuit at a glance.

Table 1 - Circuit Connections

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbol** | **Part** | **Arduino Uno Pin** | **Notes** |
| **U1** | HC-SR04 Ultrasonic Sensor | Trig → D11 (green) Echo → D10 (orange) | - Vcc to 5 V, GND to GND.  - Keep Echo wire short to reduce noise. |
| **S1** | Momentary **push button** (override) | One leg → D4 (blue) Opposite leg → GND (black) | Pin D4 is configured with INPUT\_PULLUP, so **no external resistor** is required. |
| **D1** | 5 mm Red **LED** (alert) | Anode → 330 Ω → D6 (yellow) Cathode → GND (black) | 330 Ω limits current ≈ 10–15 mA. Use any value 220–470 Ω. |
| **BZ1** | Piezo **Buzzer** | Positive → D5 (cyan) Negative → GND (black) | Active buzzer preferred. |
| — | +5 V Rail | (red) | Feed Vcc for HC-SR04 and LED resistor. Do **not** power devices from the 3.3 V rail. |
| — | GND Rail | (black) | Tie all GND pins together; run one solid wire back to Arduino GND. |

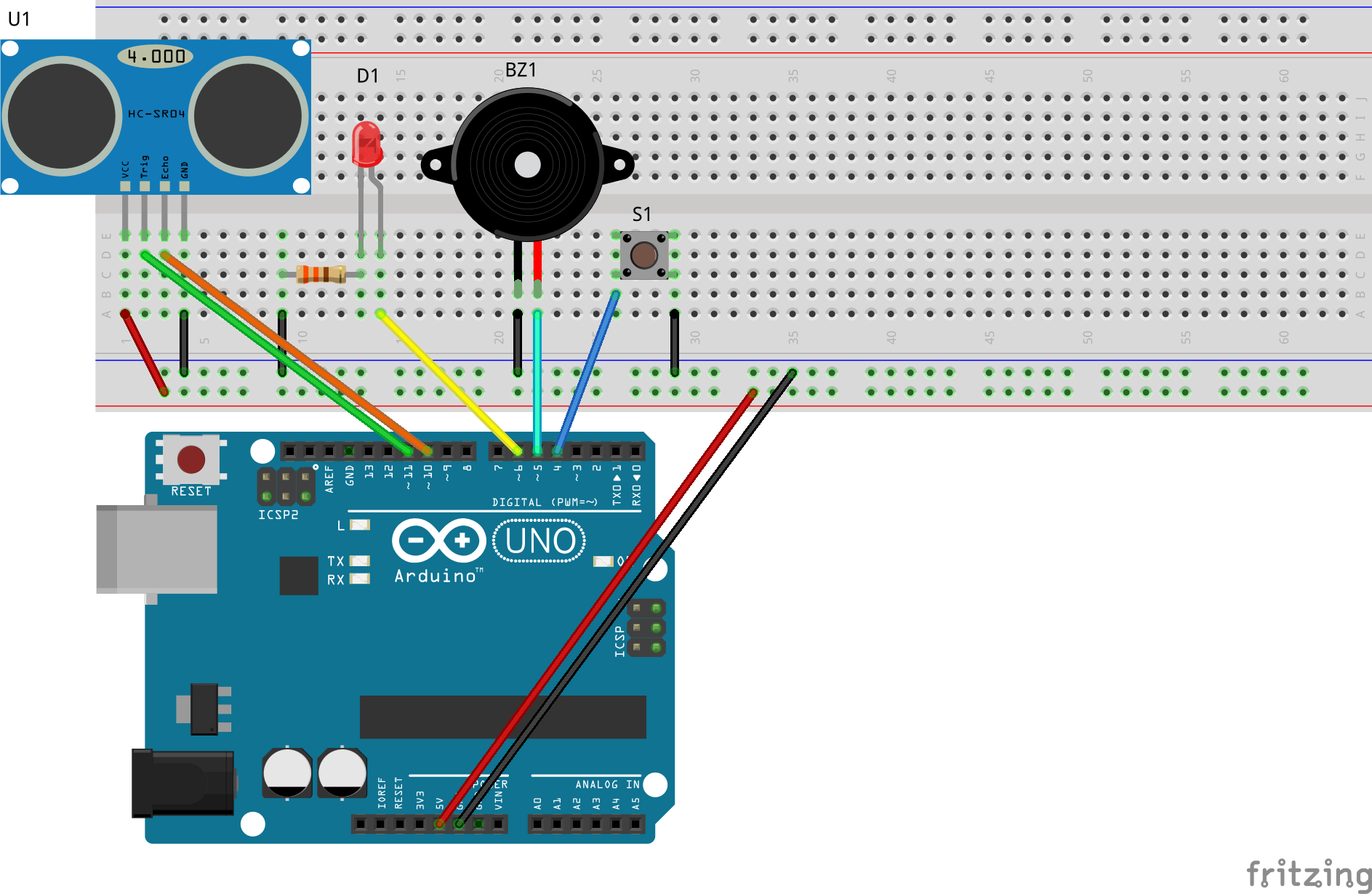


Figure 2 - Circuit Connections Diagram. Image from Fritzing, educational purpose only.

**Notes about the circuit assembly:**

1. Notice that the push button (S1) has no pull-up or pull-down resistor. This was intentional as the Arduino Uno allows the pins to be internally connected to pull-up resistors by setting:
   1. pinMode(buttonPin, INPUT\_PULLUP);   // internal pull-up
2. The active buzzer (BZ1) is chosen for its ease of use, only requiring setting it HIGH or LOW to active or deactivate it, respectively.
   1. digitalWrite(buzzerPin, HIGH);
   2. digitalWrite(buzzerPin, LOW);

## 4. Code Documentation

**4.1 Library Used**

#include <NewPing.h> // Handles HC-SR04 timing and distance math

*NewPing* library fires the trigger, waits with an accurate timeout, and returns the result in cm (or 0 cm if no echo). That keeps *void* *loop()* short and avoids manual pulse‐width math.

**4.2 Global Constants & Pins**

Below is the list of constant variables used in the code. As constants they do not change during the running of the code, and as global they can be used inside and outside functions.

Table 2 - Global constants and Pins.

|  |  |  |
| --- | --- | --- |
| **Constant / Pin** | **Purpose** | **Typical Value** |
| *maxDistance* | Rejects echoes beyond bowl range | 30 cm |
| *threshold* | Depth where the bowl is *called* “empty” | 10 cm (adjust per bowl) |
| *triggerPin / echoPin* | HC-SR04 sensor pin | D9 / D10 |
| *ledPin* | LED pin | D6 |
| *buzzerPin* | Buzzer pin alert | D5 |
| *buttonPin* | Push-button pin | D4 (INPUT\_PULLUP) |

**4.3 Function Descriptions and Responsibilities**

Table 3 - Function descriptions and Responsibilities.

|  |  |
| --- | --- |
| **Functions** | **Descriptions & Responsibilities** |
| long checkDistance() | • Wait 50 ms for sensor stability • Call sonar.ping\_cm() (returns 0 cm if out of range) • Print reading to Serial for debugging • Return distance |
| void triggerAlert(bool alert) | • If **alert == true** → LED ON, buzzer ON • Else → LED OFF, stop buzzer • Centralizes all output control |
| void handleButton(bool) | • Read push-button with explicit if (LOW = pressed) • Perform *rising-edge* detection (debounce ready) • If pressed *while* bowl empty → set alertSilenced = true |
| void loop() | • Calls the three functions • Maintains alertSilenced tracking • Non-blocking 200 ms pause |

**4.4 Code Explanation**

While the code can be found on our [GitHub](https://github.com/cmendes-dc), the explanation of how the code works is shown below.

1. **Distance Measurement** – Every 200 ms the loop calls *checkDistance(),* which fires the HC-SR04 with the *NewPing* library and returns the bowl-to-food distance in centimetres. A 0 cm reading means “no echo”—it’s ignored.
2. **Empty-bowl Decision** – An explicit if sets *bowlEmpty* to true when the measured distance is greater than 10 cm (adjustable threshold) and not zero, indicating that the kibble surface has dropped below the sensor’s sight-line.
3. **Push-button Override** – *handleButton()* reads the active-LOW push-button, performs rising-edge detection, and—if the bowl is empty—toggles *alertSilenced = true*. That mutes further alerts until food is replenished.
4. **Alert Control** – triggerAlert() drives the LED and an active buzzer. When the bowl is empty and alerts have not been silenced, both outputs are driven HIGH; otherwise they remain LOW.
5. **Self-Re-Arm** – As soon as fresh food is detected (bowlEmpty == false), the code clears alertSilenced, re-arming the system for the next empty-bowl event.

Together these steps create an autonomous monitor: it warns the owner when the bowl is empty, lets them mute the alarm with one press, and automatically resets once the bowl is refilled.

## 5. Ethics, Privacy, or Security Disclaimer

We reflected on the ethical implications of our project. While the system does not collect personal data, it is critical to ensure that users do not rely entirely on automation for pet care. This device is a support tool, not a replacement for responsible pet ownership. Future versions that include connectivity must secure data using encrypted transmission.

## 6. References

* Elegoo HC-SR04 documentation: <https://www.elegoo.com/>
* Arduino NewPing Library: <https://github.com/microflo/NewPing>
* Arduino Tutorial: Ultrasonic Sensor HC-SR04 - <https://randomnerdtutorials.com/>
* Fritzing - <https://fritzing.org/>