SMART MULTI-PET FEEDING SYSTEM

This project presents a prototype that uses RFID technology for pet identification and automated, portion-controlled feeding in separate bowls to ensure pet health and hygiene. Integrated with ThingSpeak Cloud, it enables real-time monitoring, live video streaming via web browser, and sends email alerts for efficient multi-pet management.

Group Members

Besaam Ansar Chaudhry	19I-0895
Shahmeer Ali Mirza	19I-0860
Syed Shafqat Hussain Qamar	20I-2461

Project Supervisor

Dr. Ataul Aziz Ikram

Department of Electrical Engineering

National University of Computer and Emerging Sciences, Islamabad

2025

Developer's Submission

"This report is being submitted to the Department of Electrical Engineering of the National University of Computer and Emerging Sciences in partial fulfillment of the requirements for the degree of BS in Electrical Engineering"

Developer's Declaration

"We take full responsibility for the project work conducted during the Final Year Project (FYP) titled **"Smart Multi-Pet Feeding System"**. The project work presented in the FYP report is completed by us with no significant help from any other person, although any minor assistance received has been duly acknowledged. We have written the entire FYP report ourselves. Additionally, we confirm that this FYP (or any substantially similar project work) has not been presented to any other degree-awarding institution within Pakistan or abroad.

We acknowledge that the Department of Electrical Engineering at the National University of Computer and Emerging Sciences has strict academic standards. We assert that all material used from other sources is properly referenced. Furthermore, the work presented in the report is our own, and we have properly cited the relevant work of others, clearly distinguishing it from our own contributions.

We understand that if we are found to have violated any academic integrity policies in our FYP report, even after graduation, the University reserves the right to withdraw our BS degree. Additionally, the University may publish our names on its website, maintaining a record of students who violated these policies in their FYP reports."

Besaam Ansar	Shahmeer Ali	Shafqat Hussain
BS(EE) 2019-0895	BS(EE) 2019-0860	BS(EE) 2020-2461

_____

Certified by Supervisor

_____

Verified by Plagiarism Cell Officer

Dated: _____

Abstract

Feeding pets in multi-animal households presents ongoing challenges such as inconsistent meal times, food sharing, overfeeding, and hygiene concerns. Traditional feeders operate on fixed schedules and lack the ability to distinguish between pets, leading to food wastage and poor nutritional management. While some advanced systems use machine learning and computer vision for pet recognition, they are often costly, require extensive datasets, and can suffer from misidentification—particularly among visually similar pets.

To address these limitations, this project proposes a cost-effective Smart Multi-Pet feeding system that utilizes Radio Frequency Identification (RFID) for accurate pet recognition. Each pet is equipped with a unique RFID tag, allowing the system to detect and dispense food only when the intended pet is present. The system is integrated with load cells, servo motors, and cloud connectivity to automate feeding processes while ensuring data tracking and remote access through a dedicated web interface.

This solution minimizes food waste, prevents cross-feeding, and promotes sanitary feeding conditions. Its affordability and scalability make it accessible to a wider range of users compared to AI-based alternatives. The proposed system provides an efficient, reliable, and intelligent approach to pet care management, enhancing both pet well-being and owner convenience in multi-pet environments.

Acknowledgements

We would like to express our sincere gratitude to our supervisors, Dr. Ataul Aziz Ikram and Engr. Saad Younus, for their invaluable guidance, comments, and suggestions throughout the course of this project. We are particularly grateful to Dr. Ataul Aziz Ikram for providing constant motivation and emphasizing the importance of hard work and teamwork. We also extend our heartfelt thanks to Engr. Saad Younus for assisting in verifying the results of our device and offering guidance on how to achieve better outcomes.

Table of Contents

Table of Content

List of Figures

Chapter 1

Introduction

1.1 Background

As companion animals are becoming more common worldwide, better ways to take care of pets are needed. Good overall health and nutrition in pets, especially in houses with more than one pet, depend on proper and prompt feeding. Many pets suffer because owners often cannot keep their feeding schedules consistent because of their busy lives. Since the amount of food is timed to be delivered, even when the pet is not around, food may go stale and spoil, increasing the risk of contamination. The inflexible programming for pet activities reduces how well both food is delivered and the cleanliness of the machine.

Current research reveals that while 60% of household pets are not given regular diets, nearly half the people with multiple pets find it harder to care for them.

Nearly three-quarters of existing automated feeders that use AI or machine learning face high costs because the equipment and software must be thoroughly up to date and well trained. Therefore, these limits stop most customers from experiencing these innovations.

Techniques from machine learning and computer vision have been used to help with immediate pet identification and giving them a tailored meal. Although it is possible with these methods to see pets as having breed or certain features, they need access to a large data set and require lots of computation. Pet appearance differences should be maintained to avoid pets being misidentified and lowering the system's efficiency.

1.2 Motivation

Manual pet feeding can be inconsistent and inefficient, especially in multi-pet homes. Most automated feeders on the market lack the ability to identify individual pets, leading to misfeeding. As pet care becomes more personalized, there is growing demand for intelligent, automated solutions.

Attaching RFID tags to pets' collars provides a simple and reliable method for identification. When integrated with IoT, this enables selective, automated feeding based on each pet's presence, ensuring accurate portions and hygiene.

The market reflects this need. The global pet tech market was valued at $6.9 billion in 2021 and is projected to reach over $20 billion by 2028, growing at a CAGR of 20%. Smart feeders are among the fastest-growing segments, driven by rising pet ownership, increasing awareness of pet health, and demand for connected solutions among working owners.

1.3 Problem Statement

"Smart Multi-Pet Feeding Management for Enhanced Pet Care"

The Pet feeding systems as they exist today create various problems for people:

1. Manual Feeding Demands Constant Attention:

Filling up bowls for animals by hand is not easy and this chore becomes even harder for people with several pets and no time to spare.

2. Wrong Pet, Wasted Food: The Feeding Dilemma:

Timers on feeders are set by a fixed time, without considering each pet's needs which makes pets eat the same meal at the same time. If a pet's meal is distributed while it is not home, it often remains untouched and can be unsafe for the pets that find it, causing waste.

3. Unreliable and Slow Visual Recognition Systems:

Visual feeders for pets need large data and are affected by look-alike pets, they have high delays and often make errors.

4. Cost-Intensive Limits Adoption:

Since they depend heavily on AI and ML such systems are too expensive for most people to buy.

1.4 Literature Review

A study introduced a system combining IoT and machine learning to feed pets automatically and identify them. The use of detailed algorithms and a large number of training images makes it difficult for this technology to function in real time [1].

In another study, an IoT feeder was made which could be controlled from a phone app by parents, though it did not have a good way to tell apart pets. If a pet does not eat its meal when scheduled, the food is likely to expire. This system is also not effective when there're multiple pets in the home [2].

Introducing sensors meant food could be monitored and the dispensing could be controlled, but no effort was made to automatically prevent pets from eating each other's food or to offer personalized bowls for individual pets [3].

This technology for animal tracking is valuable because of its accuracy, rapid use for identification and readiness for integration with smart feeders [4].

Almost 60% of pet owners in households with more than one pet say feeding their pets causes them difficulty [5].

At the same time, a review of expenses reveals that over 70% of current AI-assisted feeding equipment is difficult for many people to afford since it is so advanced and has many maintenance needs [6].

1.5 Report Outline

This report is organized into the following chapters:

Chapter 2 presents a thorough analysis of the proposed solution, including a detailed examination of the block diagram, a process flow chart, and the integration of sensors with the processing module.

Chapter 3 presents the results from testing the device on various subjects to improve its efficiency. It also discusses the conclusions derived from these results and provides recommendations for future enhancements to the device.

Chapter 2

Solution Design & Implementation

This chapter includes the complete design and implementation of the proposed device.
Section 2.1 describes the project's block diagram
Section 2.2 presents the flowchart
Section 2.3 describes the Mathematical formulas,
Section 2.4 demonstrates the wokwi circuit simulation,
Section 2.5 covers the software implementation,
Section 2.6 illustrates the Printed Circuit Board Design,
Section 2.7 Hardware Implementation:

2.1 Block Diagram

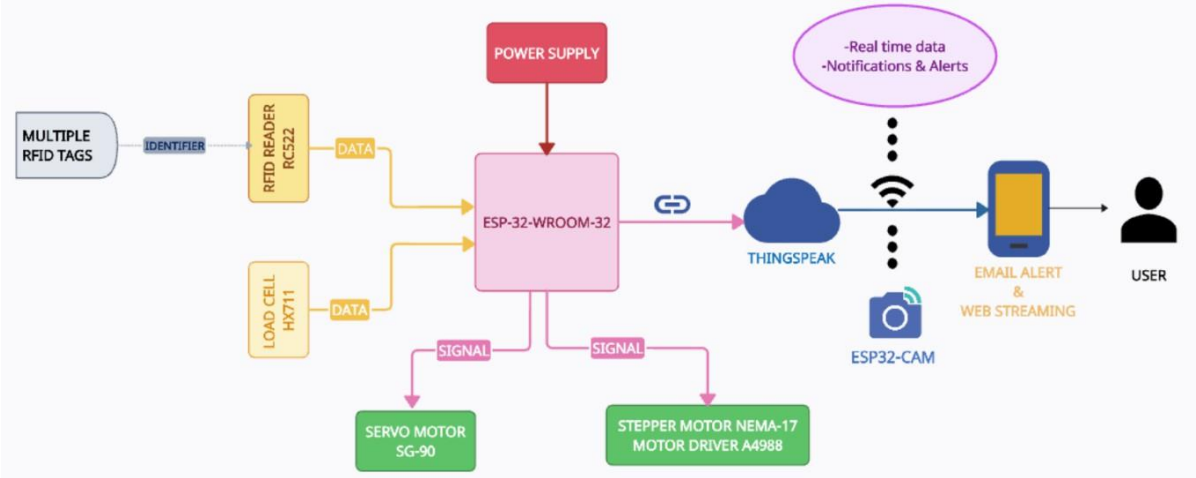


Figure 2.1 shows the block diagram of project. The details of each block with specifications are discussed below:

Figure 2.1 Block Diagram

i. Microcontroller: ESP32-Devkit V1

The ESP32-DevKit V1 acts as the central controller of the system and is based on the ESP32-WROOM-32D module. It contains a dual-core 32-bit LX6 processor running up to 240 megahertz, with 520 kilobytes of SRAM and 4 megabytes of flash memory. It supports Wi-Fi and Bluetooth for wireless communication, and provides multiple I/O interfaces including SPI, I2C, UART, ADC, DAC, and PWM.

- Operating Voltage: 3.3 **V**
- Input Supply Voltage: 5 **V**
- Current Consumption: ≈160 m**A** during Wi-Fi transmission

ii. RFID Reader RC522:

. The RC522 RFID reader identifies pets through RFID tags. It operates at 13.56 megahertz and uses SPI communication with the ESP32. It captures the tag's unique ID and sends it to the micro-controller for recognition.

- Operating Voltage: 3.3 **V**
- Current Consumption: 13 to 26 m**A**
- Detection Range: Up to 10 cm

iii. RFID Tag:

- Type: Passive RFID tag
- Memory: 64 to 256 bytes for storing unique ID and basic data
- Size: Small and lightweight, suitable for pet collars
- Communication Protocol: ISO 14443 or ISO 15693 standards
- Power Source: Powered by the RFID reader's electromagnetic field
- Durability: Water-resistant and durable for everyday use on pets

iv. Load Cell with HX711 Amplifier:

The HX711 is a precision 24-bit analog-to-digital converter (ADC). It acts as both an amplifier and ADC, amplifying the small analog signal from the load cell and converting it into digital data readable by the ESP32. This raw data is then processed using calibration formulas to determine the actual remaining food weight in the container.

- Operating Voltage: 2.6 to 5.5 **V**
- System Voltage: 5 **V**
- Current Consumption: 1.5 m**A**

v. Servo Motor SG90:

The SG90 servo motor powered by external 5V receives a Pulse width Modulation signal from the ESP32 and is activated to open and close the dispenser flap to release food with precise timing delays, ensuring accurate portion control.

- Operating Voltage: 4.8 to 6 **V**
- Power Supply: 5 **V**
- Current Consumption: 100 to 250 m**A**
- Angle Movement: ~1ms pulse = 0°, ~1.5ms pulse = 90°, ~2ms pulse = 180°

vi. Stepper Motor NEMA17:

A NEMA-17 stepper motor is responsible for the accurate placement of feeding bowls, allowing fine and reliable positioning. It is driven using an A4988 motor driver, which receives step and direction signals from the ESP32 to control the motor's precise movement.

- Operating Voltage: 12 **V**
- Rated Current: Around 1.2 to 2.0 $\boldsymbol{A/\emptyset}$
- Step Angle: 1.8° per step = 200 steps per revolution
- Holding Torque: 40 to 50 **N**·cm
- Phase Resistance: 1.5 to 2.8 ohms
- Phase Inductance: 2.5 to 4.0 m**H**
- Number of Phases: Bipolar

vii. Motor Driver A4988:

- Logic Voltage (VDD): 3 to 5.5 **V**
- Motor Voltage (VMOT): 8 to 35 **V**dc
- Output Current: Up to 2 **A** per coil
- Current Control: Adjustable via potentiometer
- Protection Features: Overcurrent, overtemperature, and undervoltage lockout
- Operating Temperature: -20°C to +85 **°C**

viii. Data Processing Unit:

The micro-controller handles all incoming data in real time. It performs data normalization noise filtering, timestamping of events, and calibration of inputs from all connected components. This pre-processing ensures high accuracy and consistency before the data is transmitted to the cloud and mobile application.

ix. ThingSpeak Cloud Platform:

ThingSpeak serves as the cloud backend for real-time data monitoring and storage. The system uploads the following data:

- Pet Identification
- Remaining food level in the container

x. User Notification and Web Streaming:

When a pet approaches the system, the RC522 RFID reader detects and ESP32 verifies its tag.

- The stepper motor rotates to position the correct feeding bowl
- The servo motor activates to dispense the required amount of food

After feeding each pet, the load cell measures the remaining food level. If the weight falls below the minimum threshold, indicating the container is empty, the system pauses further operations. Users receive timely alerts through the Email about the empty container. Once the food level is replenished above the threshold, the system automatically resumes operation.

Additionally, the ESP32-CAM provides live video streaming via a web browser, allowing the user to monitor the feeder surroundings and observe the pet's condition in real time, enhancing both visibility and care.

2.2 Flow Chart

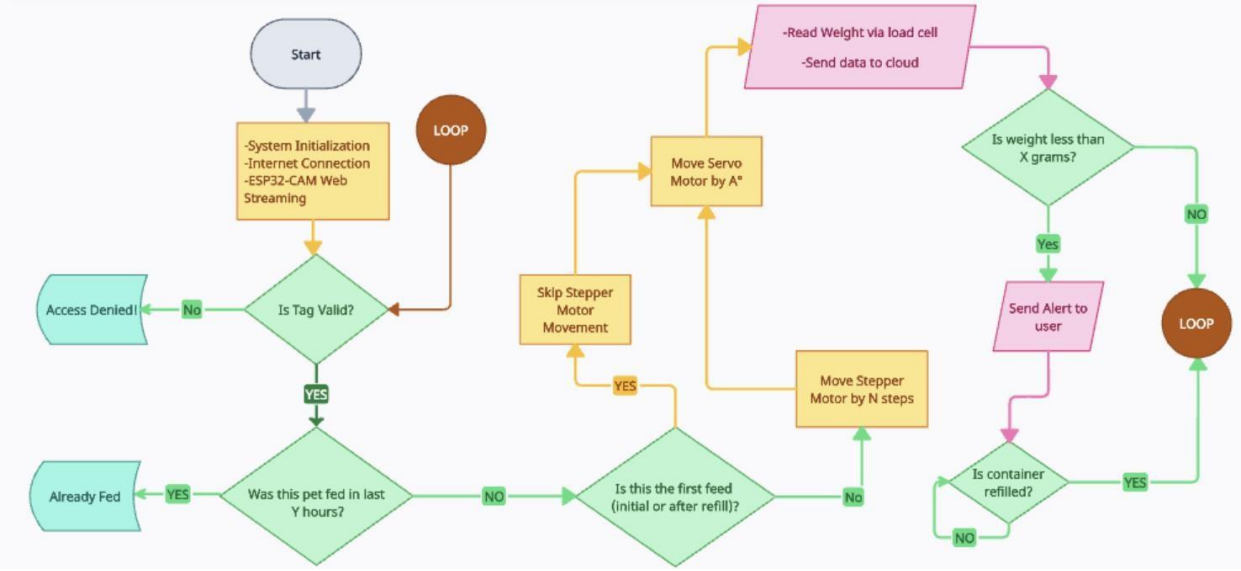


Figure 2.2 shows the flow chart of the project. The details are given below.

Figure 2.2 Flow Chart.

i. ESP-32 Initialization and Internet Connection:

The ESP32 microcontroller powers up, initializes the RFID reader, servo motor, load cell, stepper motor, connects to Wi-Fi and starts live video streaming via web browser through ESP32-CAM.

ii. Pet Detection:

When a pet approaches, the RFID reader scans the RFID tag and triggers the ESP32.

- If the tag is invalid, the system returns to sleep mode.
- If valid, the pet is identified as PET(N) and the system continues.

iii. Check PET(N) Feed Log:

- If already fed in last Y hours, a message is sent to the Blynk app "PET(N) has already been fed".
- If not fed, it proceeds to the initial feed check.

iv. Power-On/Refill Feed Check:

The system checks whether this feeding cycle is the initial one following system startup or food container refill.

- If this is the initial feeding cycle, only the servo motor is activated to open the flap and dispense food directly.
- If this is a subsequent feeding cycle, the stepper motor first rotates to position a clean bowl, then the servo motor is activated to dispense food.

v. Servo-Actuated Flap Mechanism:

- The servo motor rotates $(\boldsymbol{\theta} \rightarrow \emptyset)$ to open the flap of the container to dispense food into the bowl.
- After dispensing, the servo motor rotates $(\emptyset \rightarrow \boldsymbol{\theta})$ to close the flap.

vi. ESP32 → ThingSpeak Sync:

After the pet is fed, the ESP32 transmits following data to ThingSpeak:

- The ESP32 sends the PET(N) data.
- The ESP32 transmits the measured weight of the food in the container.

vii. Food Level Verification:

The system checks the condition;

$$\boldsymbol{Xremaining < Xthreshold}$$

- If food is sufficient, the system returns to detect new valid RFID tag.
- If food is low, it proceeds to notify the user.

viii. Data Trigger Low Food Alert:

The system sends an email alert to the user through ThingSpeak:

- "Low food level – please refill the container".

ix. Monitoring Refill:

The system keeps checking for a change in food weight $\boldsymbol{Xfilled > Xthreshold}$.

- Monitoring stays active until the container is refilled.
- Once the container is refilled, the initial-feed flag is reset and the system is ready to scan RFID tag.

2.3 Mathematical Formulas

i. Formula for weight measurement of feed in dispenser:

- $W\sigma = \frac{R - R\sigma}{C}$
- $C = \frac{R\alpha - R\sigma}{W}$

Symbol	Name	Description
R	Raw reading	current raw HX711 reading
Rσ	Offset value	offset reading
Rα	Calibration reading	raw reading with known calibration weight
W	Initial Weight	known calibration weight
C	Calibration factor	calibration factor
$W\sigma$	Current weight	calculated weight at any moment

ii. Formula to rotate the stepper motor:

- $\alpha = \frac{360^\circ}{n}$
- $\mathbb{R} = \alpha \cdot S\sigma \cdot \mu$

Symbol	Name	Description
α	rotation angle	Angle° rotated for each partition
n	number of partitions	Total number of plate sections
μ	micro stepping factor	Factor based on micro stepping (1,2,4,8,16, ...)
$S\sigma$	steps per revolution	Motor steps for a full 360° rotation
$\mathbb{R}$	required steps	Total number of steps to rotate α degrees

iii. Formula to rotate servo motor:

- $\Delta\theta = \emptyset - \theta$
- $\mathrm{T}\alpha = \frac{\Delta\theta}{\omega}$
- $\mathrm{T}\gamma = 2 \cdot \mathrm{T}\alpha + T\beta$

Symbol	Name	Description
θ	Initial angle	Servo angle when the feeder door is closed
$\emptyset$	Feeding angle	Servo angle when the feeder door is open
$\Delta\theta$	Angular displacement	Difference between open and close angles
ω	Angular speed	Speed at which servo motor changes its angle
$\mathrm{T}\alpha$	Movement time	Time taken by the servo motor to move from θ to $\emptyset$
$\mathrm{T}\beta$	Hold time	Duration the feeder remains open at $\emptyset$
$\mathrm{T}\gamma$	Operational time	Total time for open, hold and close sequence

2.4 Wokwi Circuit Simulation

This setup allows real-time simulation of the circuit, making it easy to test code, debug components, and visualize system behavior before actual hardware implementation.

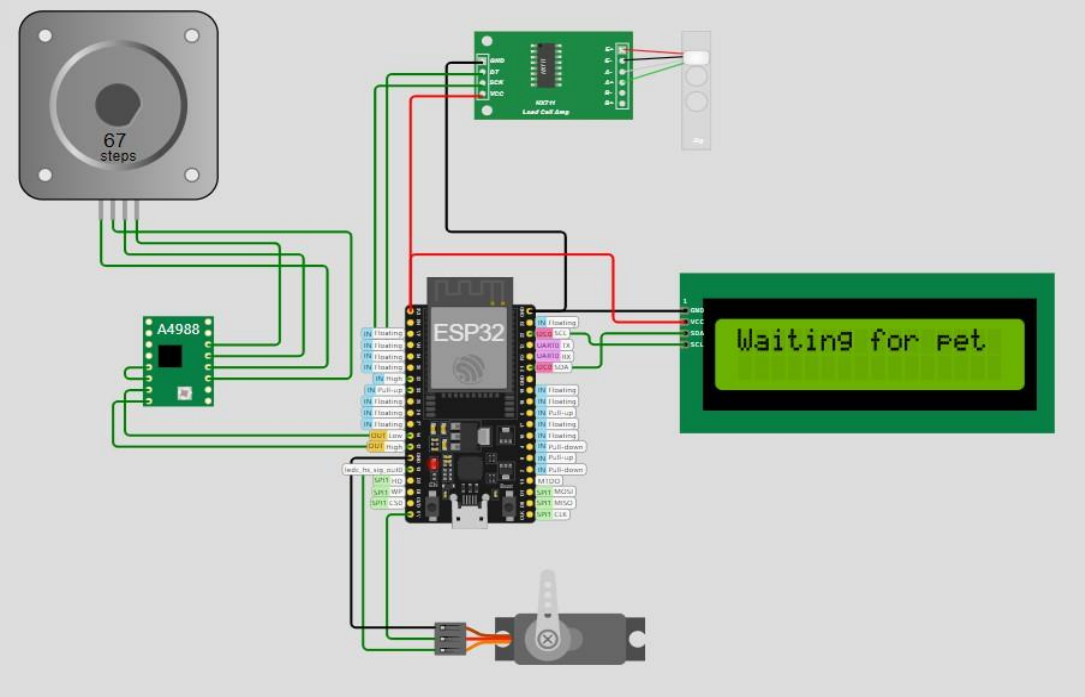


Figure 2.3 Wokwi Circuit Diagram

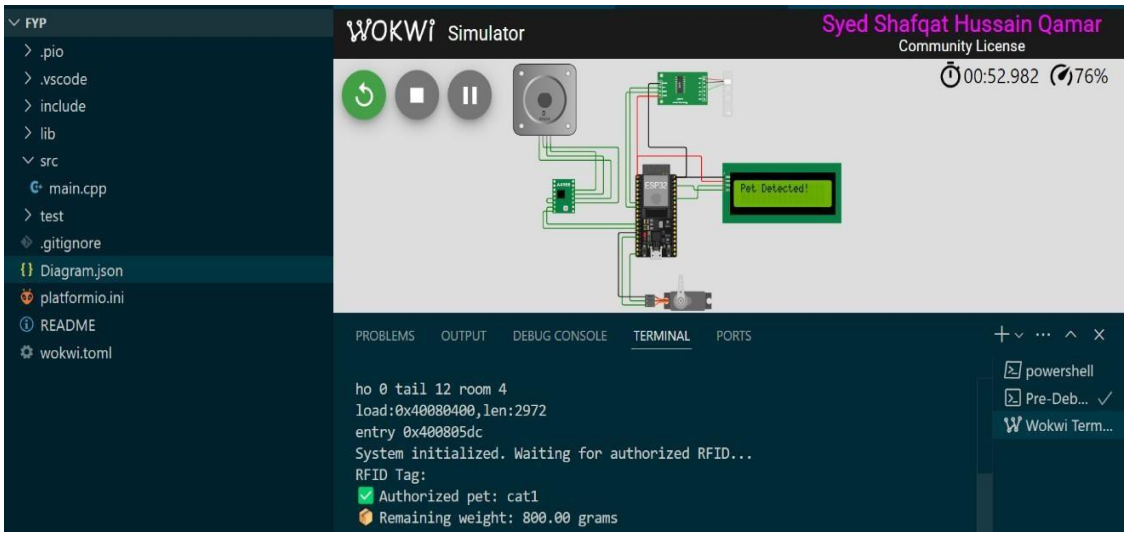


Figure 2.4 Wokwi Circuit Simulation

2.5 Software Implementation

i. Arduino IDE (ESP32-Devkit V1):

- **RFID Detection:**

The RC522 RFID module is connected to the ESP32, which identifies the pet when a valid RFID tag is detected. The ESP32 receives data from the RC522.

- **Bowl Positioning:**

The NEMA17 stepper motor moves the bowl to the proper position for each pet, ensuring a clean bowl before dispensing food.

- **Food Dispensing:**

After positioning, the SG90 servo motor is activated to dispense food with precise timing delays to ensure accurate portion control.

- **Weight Measurement:**

The HX711 load cell reading is converted from raw data to the actual remaining food weight using calibration formulas.

- **Data Transmission:**

Data is transmitted immediately after each feeding event, ensuring timely updates. The ESP32 sends this data to ThingSpeak.

- **Threshold and Alert:**

The system checks if the remaining weight is below the threshold. If yes, it sends an alert via Blynk and pauses further dispensing.

ii. ESP32-CAM (OV2640):

The ESP32-CAM delivers a live video stream to user, allowing real-time monitoring through any web browser. Simply connect to local network and view at ***http***://**192. 168. 43. 240**.

iii. ThingSpeak and Google Colab:

The ESP32 sends sensor data to ThingSpeak in JSON format via HTTP whenever a pet is fed. This includes the remaining weight in the container after food is dispensed. The data is then retrieved from ThingSpeak into Google Colab, where it is processed using Python for data analysis and visualization. This enables real-time monitoring, refill alerts, and insights into feeding patterns through visual dashboards.

- New Pet Spotted! – Instantly detects and logs each new pet visit.
- Real-Time Pet Tracker – Keeps a live count of feeding sessions.
- Smart Feed Monitor – Tracks food weight with precision after every dispense.

iv. Email Alert:

If $\boldsymbol{Xremaining < Xthreshold}$ MATLAB Analysis script on ThingSpeak sends an email alert to a Gmail account via ThingSpeak's email service, and the system pauses dispensing until $\boldsymbol{Xfilled > Xthreshold}$

2.6 Printed Circuit Board Design

- **Altium Designer Software:**

A 2-layer, 85mm × 85mm PCB is designed using Altium Designer for the **Smart Multi-Pet Feeding System.** The design minimizes power consumption and circuit complexity while ensuring optimal component placement. The PCB Editor is used as the main workspace, the Library Editor helps create custom footprints, and the Components Panel enables easy manual placement for a high-performance end product.

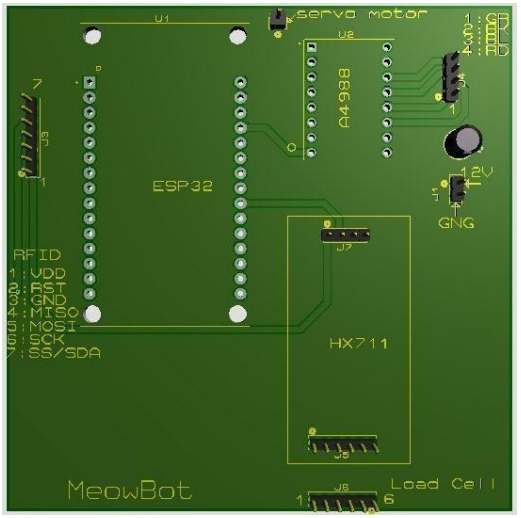


Figure 2.5 Component Layer

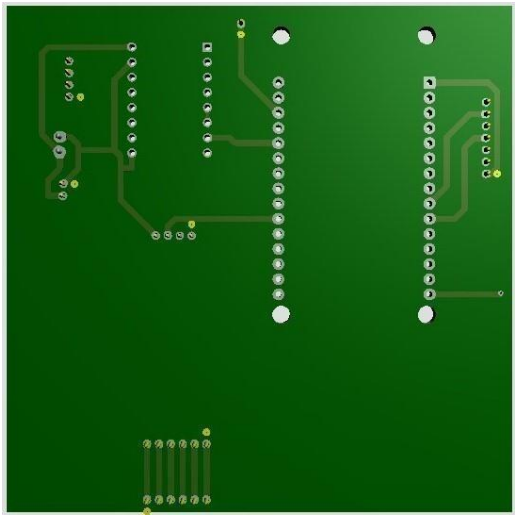
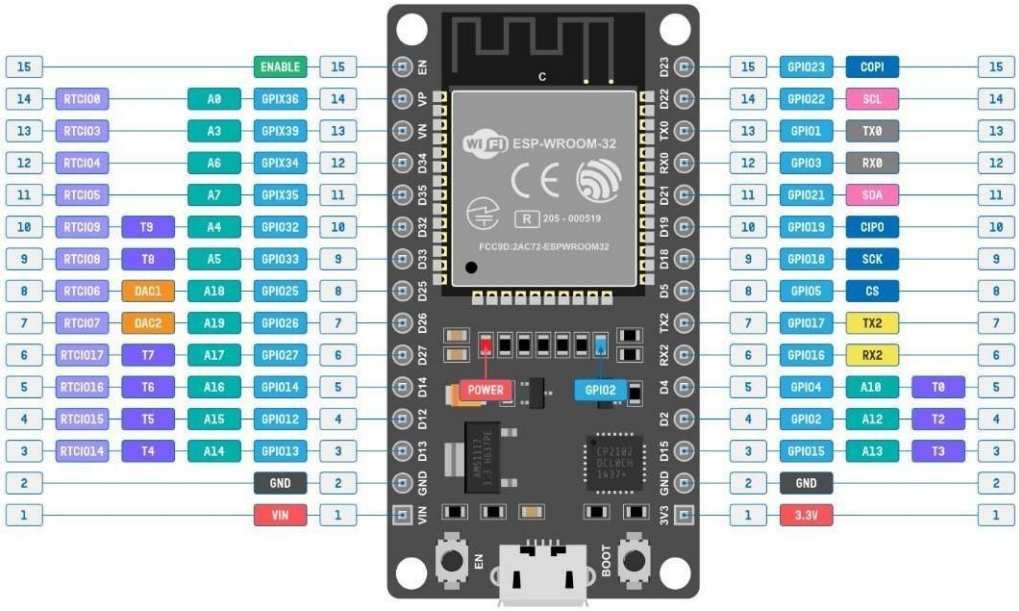


Figure 2.6 Soldering Layer

2.7 Hardware Implementation

i. ESP32-DevKit V1:



- GPIO pins **34**, **35**, **36** and **39** are input only.
- **TX0** and **RX0** (**Serial0**) are used for serial programming.
- **TX2** and **RX2** can be accessed as **Serial2**.
- Default SPI is VSPI. Both VSPI and HSPI pins can be set to any GPIO pins.
- All GPIO pins support PWM and interrupts.
- Builtin LED is connected to **GPIO2**.
- Some GPIO pins are used for interfacing flash memory and thus are not shown.

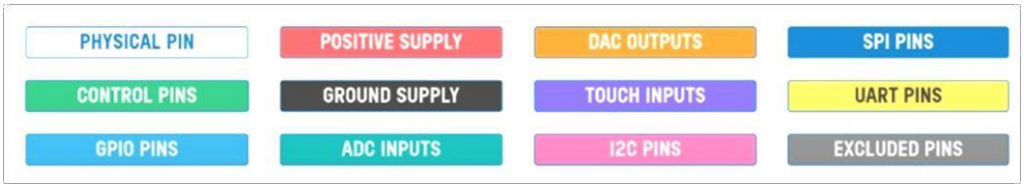
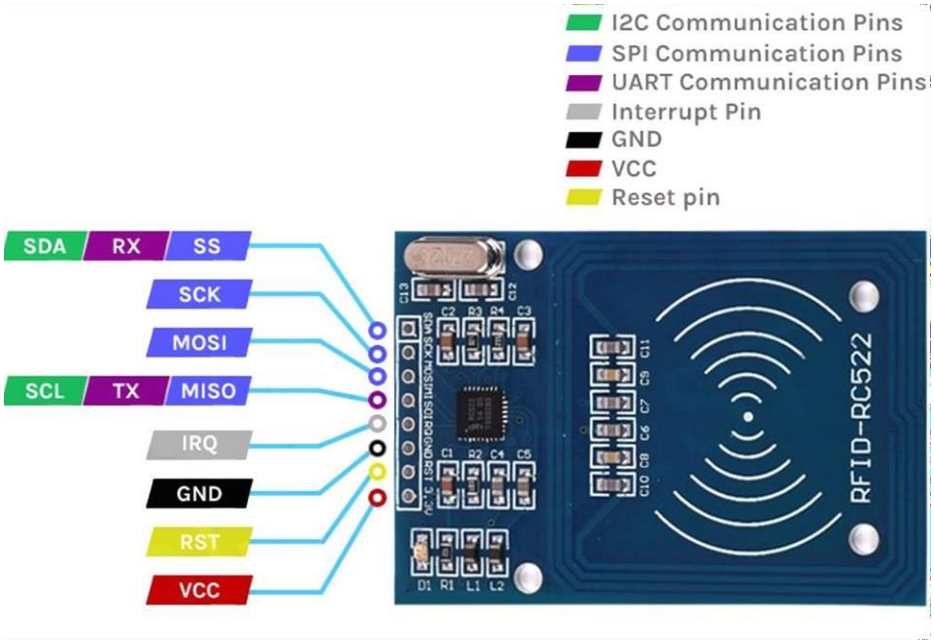


Figure 2.7 ESP32-DevKit V1

ii. RFID Module:

- **RC522 Reader**

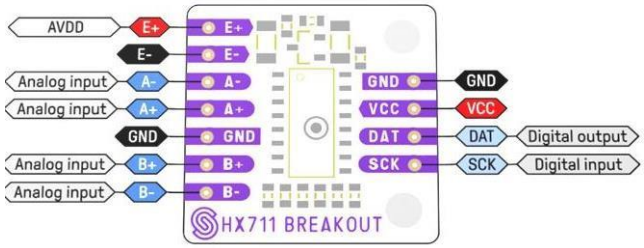


RFID RC522 Pins	ESP32 Pins
VCC	3.3V
GND	GND
MOSI	23
MISO	19
SCK	18

- **Tags**

Figure 2.8 RFID MODULE

iii. HX711 Load Cell:



Load Cell	HX711	HX711	ESP32
RED	E+	GND	GND
BLACK	E-	DAT	32
WHITE	A-	SCK	33
GREEN	A+	Vcc	3.3V

Figure 2.9 HX711 Load Cell

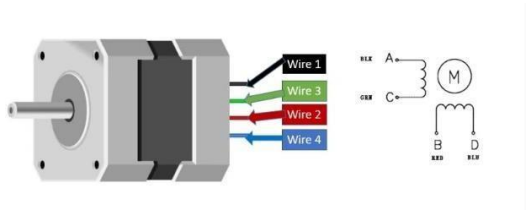
iv. SG90 Servo Motor:

Servo Motor	ESP32
VCC	5V
GND	GND
Signal	12

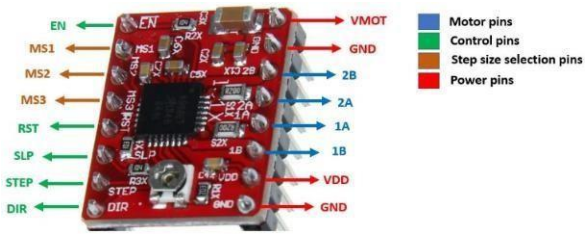
Figure 2.10 SG90 Servo Motor

v. Stepper Control Unit:

- **Stepper Motor NEMA17**



- **Motor Driver A4988**



NEMA17	A4988
BLACK	1B
GREEN	1A
BLUE	2A
RED	2B

A4988	ESP32
DIR	12
STEP	14

MS1	MS2	MS3	Micro-step Resolution
LOW	LOW	LOW	FULL STEP
HIGH	LOW	LOW	1/2 STEP
LOW	HIGH	LOW	1/4 STEP
HIGH	HIGH	LOW	1/8 STEP
HIGH	HIGH	HIGH	1/16 STEP

Figure 2.11 NEMA17 Stepper Motor with A4988 Motor Driver

vi. ESP32-CAM OV2640:

Figure 2.12 ESP32-CAM OV2640

Chapter 3

Result and Recommendations

This chapter includes the results and recommendations of the proposed device.

3.1 Pet Detection Indicator

The pet detection system uses collar-mounted RFID tags to identify individual pets through RC522 Reader Module.

The detection status is visually represented on the Thingspeak Cloud dashboard:

- No Pet Detected: If no RFID tag is in range of the RFID reader, the Thingspeak dashboard displays a light, faded circle, indicating no pet presence (Figure 3.1).

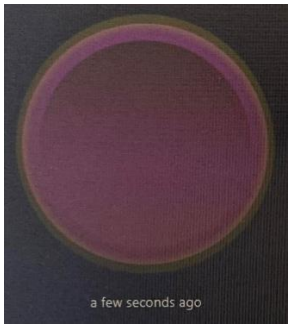


Figure 3.1 No RFID Tag Detected

- Pet Detected: When a pet with a registered collar RFID tag comes into range, the RFID reader successfully detects it. The Thingspeak dashboard then shows a dark, bold circle, confirming pet presence (Figure 3.2).

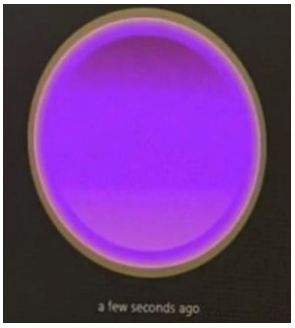


Figure 3.2 RFID Tag Detected

3.2 Count of Pets Fed

The number of pets fed is dynamically tracked and updated to the Thingspeak Cloud based on successful RFID identification and food dispensing events.

- When no pet has been fed yet, N = 0.

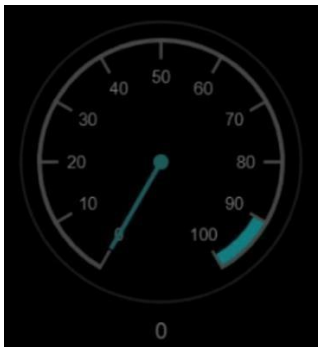


Figure 3.3 Number of Pets Fed: N=0

- As pets are fed one after another, the value updates to N = a, where a represents the total count of pets fed.

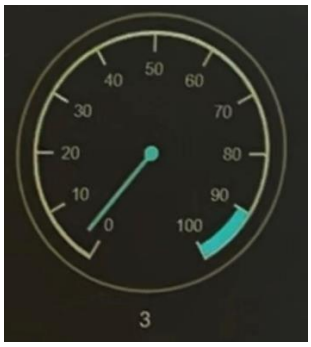


Figure 3.4 Number of Pets Fed: N=3

3.3 Real-Time Food Container Weight Measurement

The load cell sensor accurately measures the food container weight and updates the cloud dashboard in real time:

Figure 3.5 A spike in weight corresponds to the refilling of food.

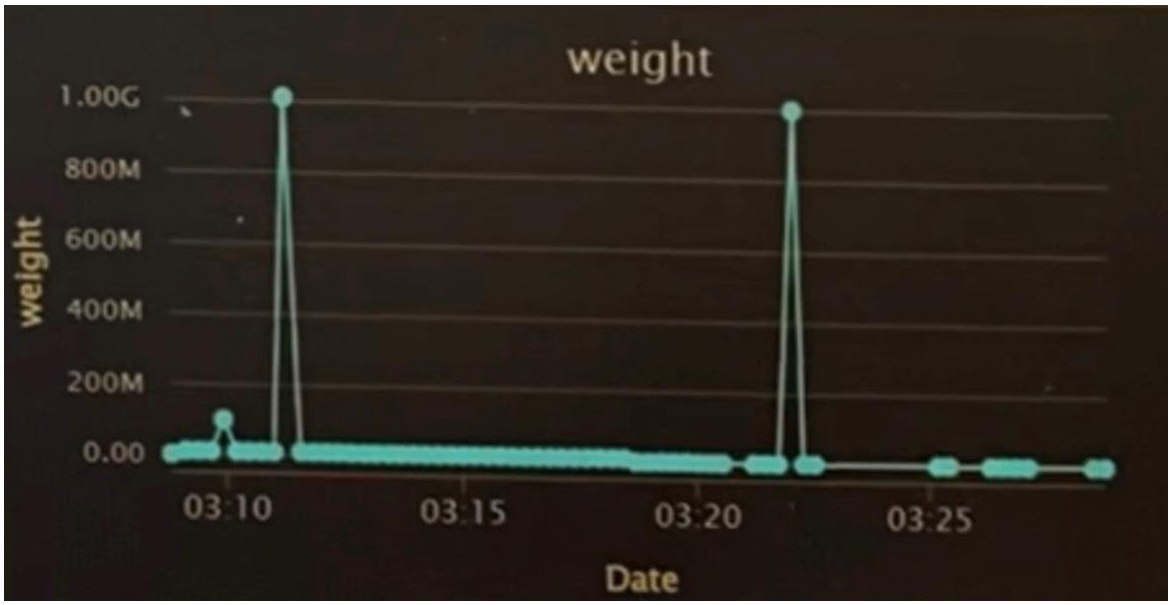


Figure 3.6 Gradual drops signify the feeding process as food is consumed by pets.

3.4 Email Alert

Thingspeak is configured to send an email via Gmail when the food container weight falls below a defined threshold. This alert, based on load cell data, notifies the user to refill the container, ensuring timely feeding without manual monitoring.

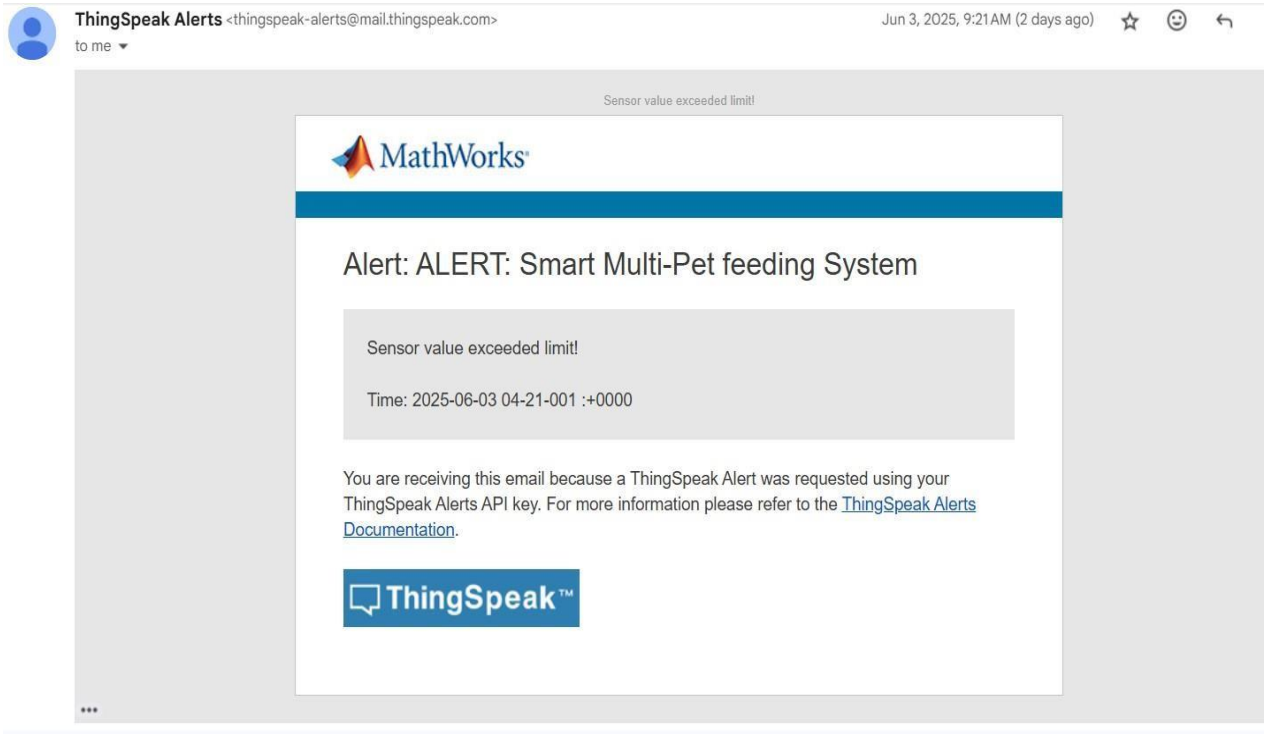


Figure 3.7 Low Food Level Alert Notification on Email

3.5 Data Visualization on Google Colab

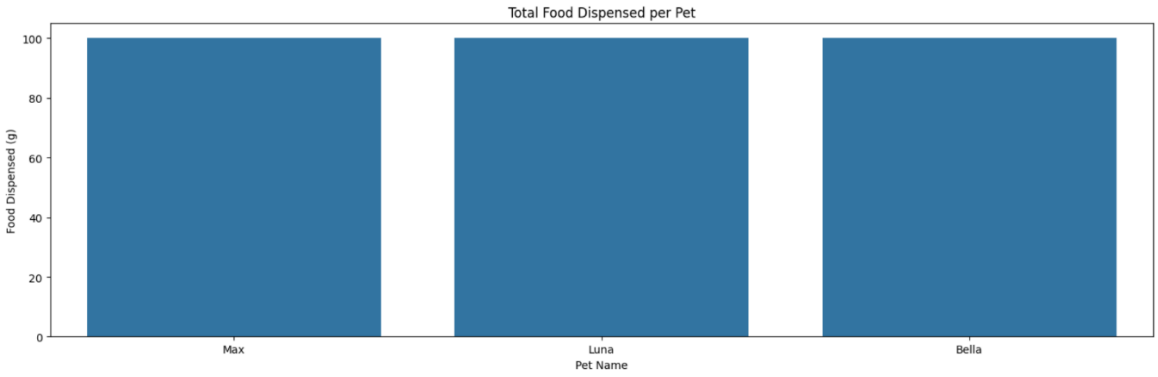


Figure 3.8 Total Food Dispense Per Pet

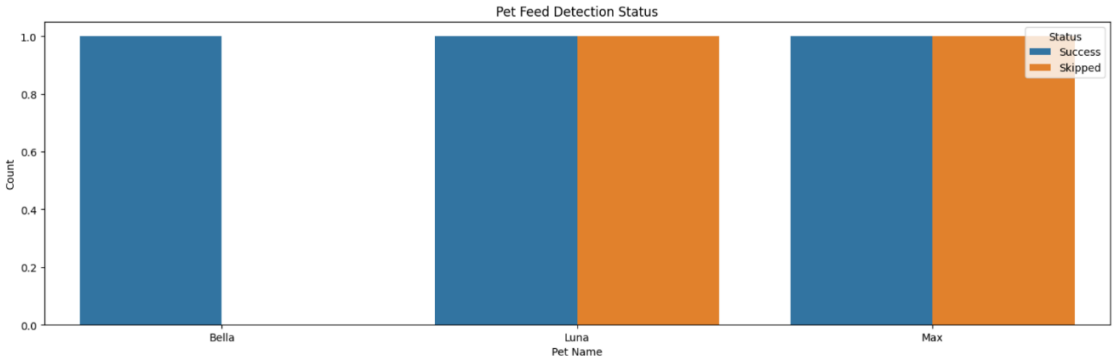


Figure 3.9 Pet Feed Detection Status

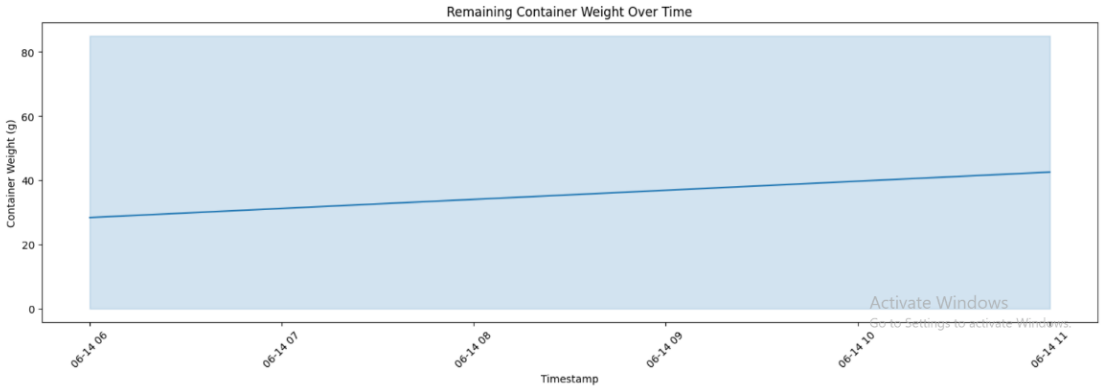


Figure 3.9 Remaining Container Weight

3.6 Recommendations / Future Work

The results obtained from the implementation of the Smart Multi-Pet Feeding System, the following recommendations are proposed to enhance system performance and ensure sustained benefits:

1. **RFID-Based Sleep Mode Trigger**:
    - Utilize the IRQ pin of the RFID module to trigger interrupts and place the system in sleep mode when idle, effectively reducing power consumption.
2. **Stepper Motor Power Management**:
    - Configure the A4988 motor driver to enter sleep mode when inactive, minimizing energy usage during non-operational periods.
3. **PIR Sensor for Camera Wake-Up**:
    - A PIR motion sensor (HC-SR501) is used to detect activity near the feeder. When motion is detected, it wakes the OV264 from sleep mode to enable video monitoring. This approach helps reduce power consumption by keeping the camera off when no motion is present.
4. **Efficient Power Conversion**:
    - Use a DC-DC buck converter to step down the 12V adapter input to 5V, efficiently powering the ESP32, ESP32-CAM, and servo motor with reduced heat loss and improved voltage regulation.
5. **Enhanced System Security**:
    - Strengthen security through data encryption and multi-factor authentication, safeguarding user data and preventing unauthorized system access.
6. **Mobile Application**:
    - A mobile application is recommended to allow users to view real-time pet camera streaming, remotely turn the feeder on or off, receive instant alerts (e.g., low food level, pet detection), and track feeding history through an intuitive interface.
7. **Intelligent Battery Backup System**:
    - Integrate a Li-ion battery backup with a TP4056 charging module in parallel with the 12V adapter. Use an LT4412 controller for automatic and seamless switching to battery power during outages, ensuring uninterrupted system operation.

8. **Multi-Container System**:

    - Enhance pet care with a system designed using multiple containers, each holding a different type of feed tailored for companion animals, including canines and felines. The system identifies each pet and dispenses the appropriate feed from the designated container based on species, breed, or age—ensuring personalized and nutritionally appropriate feeding.

Appendix-A: Code Required for Hardware and Software

Arduino IDE Code

ESP32-DevKit V1:

```
//------------------------------------ WiFi and HTTP ------------------------------------
#include <WiFi.h>
#include <HTTPClient.h>

const char* WIFI_NAME = "Infinix";
const char* WIFI_PASS = "123456789a";

const char* THINGSPEAK_API = "http://api.thingspeak.com/update";
const String WRITE_API_KEY = "OUR9PR007SGPVQVO";

//------------------------------------ Libraries ------------------------------------
#include <ESP32Servo.h>
#include "HX711.h"
#include <AccelStepper.h>
#include <SPI.h>
#include <MFRC522.h>

//------------------------------------ Pin & Peripheral Setup ------------------------------------
const int SERVO_PIN = 13;
const int LOADCELL_DT = 32;
const int LOADCELL_SCK = 33;

const int MOTOR_STEP = 14;
const int MOTOR_DIR = 12;

const int RFID_SS_PIN = 5;
const int RFID_RST_PIN = 0;

//------------------------------------ Global Variables ------------------------------------
bool rfidEventTriggered = false;
float pseudoSensor = 0.0;
long currentWeight = 0;

bool hasRotated = false;
Servo foodDispenser;

HX711 weightModule;
AccelStepper foodStepper(1, MOTOR_STEP, MOTOR_DIR);
MFRC522 tagScanner(RFID_SS_PIN, RFID_RST_PIN);
MFRC522::MIFARE_Key masterKey;
byte lastTagUID[4];

//------------------------------------ Timers ------------------------------------
unsigned long lastRFIDReadTime = 0;
unsigned long RFID_TIMEOUT_MS = 70000;
//------------------------------------ Tasks ------------------------------------
TaskHandle_t StepperControlTask;
TaskHandle_t IoTUplinkTask;
```

```
//---------------------------------------- Stepper Task ----------------------------------------
void stepperMotorLoop(void* parameter) {
  while (true) {
    foodStepper.run();
    vTaskDelay(20);
  }
}

//---------------------------------------- WiFi Upload Task ----------------------------------------
void uploadToThingSpeak(void* parameter) {
  WiFi.begin(WIFI_NAME, WIFI_PASS);
  Serial.print("Connecting to WiFi");

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi Connected!");

  while (true) {
    if (WiFi.status() == WL_CONNECTED) {
      HTTPClient httpClient;

      if (currentWeight > 10000) {
        currentWeight = 1000;
      }

      String httpRequest = String(THINGSPEAK_API) + "?api_key=" + WRITE_API_KEY +
        "&field1=" + String(rfidEventTriggered ? 1 : 0) +
        "&field2=" + String(pseudoSensor) +
        "&field3=" + String(currentWeight, 2);

      httpClient.begin(httpRequest);
      int response = httpClient.GET();

      if (response > 0) {
        Serial.print("📡 Data uploaded. Response: ");
        Serial.println(response);
      } else {
        Serial.print("❌ Failed upload. Error code: ");
        Serial.println(response);
      }
```

Appendix-A: Code Required for Hardware and Software

```
httpClient.end();
      rfidEventTriggered = false;
    } else {
      Serial.println("🔁 Attempting WiFi reconnect...");
      WiFi.begin(WIFI_NAME, WIFI_PASS);
    }

    if (pseudoSensor > 3) {
      pseudoSensor = 0;
    }

    vTaskDelay(15000 / portTICK_PERIOD_MS);
  }
}

//---------------------------------------- Setup ----------------------------------------
void setup() {
  Serial.begin(9600);
  foodDispenser.attach(SERVO_PIN);

  weightModule.begin(LOADCELL_DT, LOADCELL_SCK);
  weightModule.set_scale(75.8095);
  weightModule.tare();

  foodStepper.setMaxSpeed(500);
  foodStepper.setAcceleration(250);
  foodStepper.setSpeed(100);
  foodStepper.moveTo(0);

  SPI.begin();
  tagScanner.PCD_Init();

  for (byte i = 0; i < 6; i++) {
    masterKey.keyByte[i] = 0xFF;
  }
xTaskCreatePinnedToCore(
    stepperMotorLoop,
    "StepperTask",
    2048,
    NULL,
    1,
    &StepperControlTask,
    0
  );

  xTaskCreatePinnedToCore(
    uploadToThingSpeak,
    "UploaderTask",
    4096,
    NULL,
    1,
    &IoTUplinkTask,
    1
  );
}
```

Appendix-A: Code Required for Hardware and Software

```
//--------------------------------------- Loop ----------------------------------------
void loop() {
  currentWeight = weightModule.get_units();
  Serial.print("Current Load: ");
  Serial.println(currentWeight, 1);

  foodDispenser.write(0); // Default position

  if ((millis() - lastRFIDReadTime) > RFID_TIMEOUT_MS) {
    lastTagUID[1] = 0x00;
  }

  if (!tagScanner.PICC_IsNewCardPresent() || !tagScanner.PICC_ReadCardSerial()) return;

  MFRC522::PICC_Type tagType = tagScanner.PICC_GetType(tagScanner.uid.sak);
  if (!(tagType == MFRC522::PICC_TYPE_MIFARE_1K || tagType == MFRC522::PICC_TYPE_MIFARE_4K))
{
    Serial.println("⛔ Unsupported tag type.");
    return;
  }

  if (memcmp(tagScanner.uid.uidByte, lastTagUID, 4) != 0) {
    lastRFIDReadTime = millis();
    Serial.println("🎫 New RFID Detected!");
    pseudoSensor++;
    rfidEventTriggered = true;

    if (hasRotated && foodStepper.distanceToGo() == 0) {
      foodStepper.moveTo(foodStepper.currentPosition() + 66);
      delay(3000);
      foodDispenser.write(80);
      delay(5000);
    } else {
      foodDispenser.write(80);
      delay(5000);
    }

    hasRotated = true;
    memcpy(lastTagUID, tagScanner.uid.uidByte, 4);
  } else {
    Serial.println("⚠️ Tag already processed.");
    rfidEventTriggered = false;
  }

  tagScanner.PICC_HaltA();
  tagScanner.PCD_StopCrypto1();
}
```

Arduino IDE Code

ESP32-CAM

```
  #include "esp_camera.h"
#include <WiFi.h>
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
#include "fb_gfx.h"
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
#include "esp_http_server.h"

// Replace with your WiFi network details
const char* wifiSSID = "Infinix";
const char* wifiPassword = "123456789a";

#define MULTIPART_BOUNDARY "123456789000000000000987654321"

// Select the AI Thinker ESP32-CAM board configuration
#define CAM_MODEL_AI_THINKER

#if defined(CAM_MODEL_AI_THINKER)
  #define CAM_PWDN_GPIO     32
  #define CAM_RESET_GPIO    -1
  #define CAM_XCLK_GPIO      0
  #define CAM_SIOD_GPIO     26
  #define CAM_SIOC_GPIO     27
  #define CAM_Y9_GPIO       35
  #define CAM_Y8_GPIO       34
  #define CAM_Y7_GPIO       39
  #define CAM_Y6_GPIO       36
  #define CAM_Y5_GPIO       21
  #define CAM_Y4_GPIO       19
  #define CAM_Y3_GPIO       18
  #define CAM_Y2_GPIO        5
  #define CAM_VSYNC_GPIO    25
  #define CAM_HREF_GPIO     23
  #define CAM_PCLK_GPIO     22
#else
  #error "Camera model not defined"
#endif

static const char* CONTENT_TYPE_STREAM = "multipart/x-mixed-replace;boundary="
MULTIPART_BOUNDARY;
static const char* BOUNDARY = "\r\n--" MULTIPART_BOUNDARY "\r\n";
static const char* HEADER_JPEG = "Content-Type: image/jpeg\r\nContent-Length: %u\r\n\r\n";

httpd_handle_t cameraServer = NULL;
```

```
static esp_err_t streamCallback(httpd_req_t *request) {
  camera_fb_t *frame = NULL;
  esp_err_t result = ESP_OK;
  size_t jpegLen = 0;
  uint8_t *jpegBuf = NULL;
  char buffer[64];

  result = httpd_resp_set_type(request, CONTENT_TYPE_STREAM);
  if (result != ESP_OK) return result;

  while (true) {
    frame = esp_camera_fb_get();
    if (!frame) {
      Serial.println("Camera capture failed");
      result = ESP_FAIL;
      break;
    }

    if (frame->width > 400) {
      if (frame->format != PIXFORMAT_JPEG) {
        bool jpegCreated = frame2jpg(frame, 80, &jpegBuf, &jpegLen);
        esp_camera_fb_return(frame);
        frame = NULL;
        if (!jpegCreated) {
          Serial.println("JPEG compression failed");
          result = ESP_FAIL;
          break;
        }
      } else {
        jpegLen = frame->len;
        jpegBuf = frame->buf;
      }
    }

    size_t headerLen = snprintf(buffer, sizeof(buffer), HEADER_JPEG, jpegLen);
    result = httpd_resp_send_chunk(request, buffer, headerLen);
    if (result == ESP_OK) result = httpd_resp_send_chunk(request, (const char *)jpegBuf,
jpegLen);
    if (result == ESP_OK) result = httpd_resp_send_chunk(request, BOUNDARY,
strlen(BOUNDARY));

    if (frame) esp_camera_fb_return(frame);
    else if (jpegBuf) free(jpegBuf);

    if (result != ESP_OK) break;
  }
  return result;
}
```

```
  void initiateCameraStream() {
  httpd_config_t config = HTTPD_DEFAULT_CONFIG();
  config.server_port = 80;

  httpd_uri_t uriHandler = {
    .uri       = "/",
    .method    = HTTP_GET,
    .handler   = streamCallback,
    .user_ctx  = NULL
  };

  if (httpd_start(&cameraServer, &config) == ESP_OK) {
    httpd_register_uri_handler(cameraServer, &uriHandler);
  }
}

void setup() {
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
  Serial.begin(115200);

  camera_config_t camCfg;
  camCfg.ledc_channel = LEDC_CHANNEL_0;
  camCfg.ledc_timer = LEDC_TIMER_0;
  camCfg.pin_d0 = CAM_Y2_GPIO;
  camCfg.pin_d1 = CAM_Y3_GPIO;
  camCfg.pin_d2 = CAM_Y4_GPIO;
  camCfg.pin_d3 = CAM_Y5_GPIO;
  camCfg.pin_d4 = CAM_Y6_GPIO;
  camCfg.pin_d5 = CAM_Y7_GPIO;
  camCfg.pin_d6 = CAM_Y8_GPIO;
  camCfg.pin_d7 = CAM_Y9_GPIO;
  camCfg.pin_xclk = CAM_XCLK_GPIO;
  camCfg.pin_pclk = CAM_PCLK_GPIO;
  camCfg.pin_vsync = CAM_VSYNC_GPIO;
  camCfg.pin_href = CAM_HREF_GPIO;
  camCfg.pin_sccb_sda = CAM_SIOD_GPIO;
  camCfg.pin_sccb_scl = CAM_SIOC_GPIO;
  camCfg.pin_pwdn = CAM_PWDN_GPIO;
  camCfg.pin_reset = CAM_RESET_GPIO;
  camCfg.xclk_freq_hz = 20000000;
  camCfg.pixel_format = PIXFORMAT_JPEG;

  if (psramFound()) {
    camCfg.frame_size = FRAMESIZE_UXGA;
    camCfg.jpeg_quality = 10;
    camCfg.fb_count = 2;
  } else {
    camCfg.frame_size = FRAMESIZE_SVGA;
    camCfg.jpeg_quality = 12;
    camCfg.fb_count = 1;
  }
```

Appendix-A: Code Required for Hardware and Software

```
if (esp_camera_init(&camCfg) != ESP_OK) {
    Serial.println("Failed to initialize camera");
    return;
  }

  WiFi.begin(wifiSSID, wifiPassword);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("Connected to WiFi");
  Serial.print("Stream URL: http://");
  Serial.println(WiFi.localIP());

  initiateCameraStream();
}

void loop() {
  delay(1);
}
```

Wokwi Setup

Wokwi.toml

```
[wokwi]
version = 1
firmware = ".pio/build/esp32doit-devkit-v1/firmware.bin"
elf = ".pio/build/esp32doit-devkit-v1/firmware.elf"
```

Platformio.ini

```
[env:esp32doit-devkit-v1]
platform = espressif32
board = esp32doit-devkit-v1
framework = arduino
lib_deps =
    madhephaestus/ESP32Servo@^3.0.6
    bogde/HX711@^0.7.5
    marcoschwartz/LiquidCrystal I2C@^1.1.4
```

Diagram.json

```
{
  "version": 1,
  "author": "Uri Shaked",
  "editor": "wokwi",
  "parts": [
    { "type": "board-esp32-devkit-c-v4", "id": "esp", "top": 28.8, "left": -52.76, "attrs":
{} },
    {
      "type": "wokwi-hx711",
      "id": "cell1",
      "top": -179.8,
      "left": 2.6,
      "attrs": { "type": "5kg" }
    },
    { "type": "wokwi-a4988", "id": "drv1", "top": 62.4, "left": -264, "attrs": {} },
    {
      "type": "wokwi-stepper-motor",
      "id": "stepper1",
      "top": -197.99,
      "left": -364.37,
      "attrs": { "size": "17" }
    },
    { "type": "wokwi-servo", "id": "servo1", "top": 276.4, "left": -28.8, "attrs": {} },
```

```
    {
      "type": "wokwi-lcd1602",
      "id": "lcd2",
      "top": 25.6,
      "left": 168.8,

      "attrs": { "pins": "i2c" }
    }
  ],



  "connections": [
    [ "esp:TX", "$serialMonitor:RX", "", [] ],
    [ "esp:RX", "$serialMonitor:TX", "", [] ],
    [ "stepper1:A-", "drv1:1A", "green", [ "v38.4", "h182.4", "v86.4" ] ],
    [ "stepper1:A+", "drv1:1B", "green", [ "v9.6", "h192", "v115.2" ] ],
    [ "stepper1:B+", "drv1:2A", "green", [ "v28.8", "h144", "v76.8" ] ],
    [ "stepper1:B-", "drv1:2B", "green", [ "v19.2", "h115.19", "v86.4", "h-19.2" ] ],
    [ "drv1:STEP", "esp:14", "green", [ "h-19.2", "v38.4" ] ],
    [ "drv1:DIR", "esp:12", "green", [ "h-28.8", "v38.4" ] ],
    [ "drv1:RESET", "drv1:SLEEP", "green", [ "h-19.2", "v9.6" ] ],
    [ "servo1:PWM", "esp:13", "green", [ "h-57.6", "v-153.4" ] ],
    [ "servo1:V+", "esp:5V", "green", [ "h-38.4", "v-105.5" ] ],
    [ "servo1:GND", "esp:GND.1", "black", [ "h-67.2", "v-144" ] ],
    [ "cell1:GND", "esp:GND.2", "black", [ "h-28.8", "v143.7", "h96", "v67.2" ] ],
[ "cell1:GND", "esp:GND.2", "black", [ "h-28.8", "v143.7", "h96", "v67.2" ] ],
    [ "cell1:DT", "esp:32", "green", [ "h-76.8", "v258.7" ] ],
    [ "cell1:SCK", "esp:33", "green", [ "h-86.4", "v258.4" ] ],
    [ "cell1:VCC", "esp:3V3", "red", [ "h0" ] ],
    [ "lcd2:GND", "esp:GND.2", "black", [ "h-96" ] ],
    [ "lcd2:VCC", "esp:3V3", "red", [ "h-57.6", "v-57.5", "h-172.8" ] ],
    [ "lcd2:SCL", "esp:22", "green", [ "h-76.8", "v-9.3" ] ],
    [ "lcd2:SDA", "esp:21", "green", [ "h-67.2", "v29" ] ]
  ],
  "dependencies": {}
}
```

**** Arduino IDE code for ESP32-DevKit V1 remains the same when running Wokwi simulations in Visual Studio Code.***

Google Colab

```
# STEP 1: Upload Your Local CSV Data
from google.colab import files
uploaded = files.upload()
# STEP 2: Import Required Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import requests

# STEP 3: Load Local CSV Data
csv_df = pd.read_csv('SMART_MULTIPET_FEEDING_SYSTEM.csv')
csv_df.replace("-", np.nan, inplace=True)
csv_df['servo_delay (s)'] = pd.to_numeric(csv_df['servo_delay (s)'], errors='coerce')
csv_df['stepper_steps'] = pd.to_numeric(csv_df['stepper_steps'], errors='coerce')
csv_df['bowl_no'] = pd.to_numeric(csv_df['bowl_no'], errors='coerce')
csv_df['timestamp'] = pd.to_datetime(csv_df['timestamp'])
# STEP 4: Fetch Live Weight Data from ThingSpeak
# Replace with your real ThingSpeak channel ID and field number
channel_id = 'YOUR_CHANNEL_ID'
field = 1
url = f"https://api.thingspeak.com/channels/{channel_id}/fields/{field}.json?results=100"
response = requests.get(url)
if response.status_code == 200:
    data = response.json()
    feeds = data['feeds']

    ts_df = pd.DataFrame(feeds)
    ts_df['created_at'] = pd.to_datetime(ts_df['created_at'])
    ts_df['field1'] = pd.to_numeric(ts_df['field1'], errors='coerce')
    print("☑ Live weight data loaded from ThingSpeak:")
    print(ts_df.tail())
# STEP 5: Refill Alert Logic
    latest_weight = ts_df['field1'].dropna().iloc[-1]
    if latest_weight < 20:
        print(f"⚠ ALERT: Current container weight is {latest_weight}g. Please refill!")
    else:
        print(f"☑ Container weight is healthy: {latest_weight}g")
 # STEP 6: Plot Live Weight Over Time
    plt.figure(figsize=(12, 5))
    plt.plot(ts_df['created_at'], ts_df['field1'], marker='o', color='teal')
    plt.title("Live Container Weight from ThingSpeak")
    plt.xlabel("Timestamp")
    plt.ylabel("Weight (g)")
    plt.xticks(rotation=45)
    plt.grid(True)
    plt.tight_layout()
    plt.show()
else:
    print("Failed to retrieve data from ThingSpeak:", response.status_code)
# STEP 7: (Optional) Summary of Your Local CSV Data
print("\nFeeding Summary by Pet:")
print(csv_df.groupby(['pet_name', 'status']).size().unstack(fill_value=0))
```

Bibliography

[1] M. A. Hannan, M. F. M. Basar, and A. Hussain, "Smart Pet Feeder Using IoT and Machine Learning," Journal of Physics: Conference Series, vol. 1797, no. 1, p. 012018, 2021.

[2] A. M. a. A. G. Sanda, "Automatic Pet Feeder with IoT Integration," Politehnica University of Bucharest, Bucharest, 2021.

[3] M. S. W. Rahman and S. S. U. Hasan, "Design and Implementation of IoT-based Smart Pet Feeder," International Journal of Mechanical and Design Engineering Systems, vol. 1, no. 2, pp. 25–30, 2020.

[4] E. Pereira et al., "RFID Technology for Animal Tracking: A Survey," 2023.

[5] "Pet Care Survey," American Pet Products Association, 2021.

[6] "The Cost of AI-Based Pet Feeders: Market Analysis," PetTech Insights, 2022