MINI PROJECT
Project Title: Smart Pet Feeder

Team Members:

Edwin Joel J – 22ITA37

Hareesh Kumar S – 22ITA38

Hareshvar S P – 22ITA39

Hariharan K – 22ITA40

Janadeeswaran P – 22ITA41

Smart Pet Feeder Project

1. Abstract

The Smart Pet Feeder project aims to design and implement an automated pet feeding system using the NodeMCU ESP8266 microcontroller, a servo motor, a 16x2 LCD with an I2C module, and a web-based interface. This system allows pet owners to remotely dispense food by clicking a "Feed" button on a web application, which triggers the servo to open a feeder mechanism for a specified duration. The LCD displays real-time status updates, such as "Feeding…" and "Feed Complete," enhancing user interaction. The project integrates hardware and software components to achieve seamless communication over Wi-Fi, addressing challenges like power management, I2C communication, and network reliability. Through iterative testing and debugging, the system was successfully developed, offering a cost-effective, scalable solution for pet care automation. This report details the methodology, hardware setup, software implementation, and results, concluding with insights into its practical applications and potential enhancements.

2. Introduction

In modern households, pet care often requires balancing busy schedules with the responsibility of feeding pets on time. Manual feeding can be inconvenient, especially for owners who are away from home for extended periods. The advent of Internet of Things (IoT) technology has enabled the development of smart devices that automate routine tasks, improving convenience and efficiency. The Smart Pet Feeder project leverages IoT principles to create a remotely controlled feeding system, allowing pet owners to ensure their pets are fed without physical presence.

The system uses the NodeMCU ESP8266, a low-cost Wi-Fi-enabled microcontroller, as its core, paired with a servo motor to dispense food and a 16x2 LCD to provide visual feedback. A web application serves as the user interface, sending commands to the NodeMCU over a local Wi-Fi network. This project was inspired by the need for a practical, affordable solution that combines hardware prototyping with software development, making it accessible to hobbyists and pet enthusiasts alike.

The objectives include designing a reliable mechanism to dispense food, establishing stable Wi-Fi communication, and ensuring user-friendly feedback through the LCD. Challenges encountered during development—such as power supply issues, I2C configuration, and network errors—were systematically addressed, resulting in a functional prototype. This report outlines the project's scope, methodology, and outcomes, providing a comprehensive guide for replication or further improvement.

3. Project Description

The Smart Pet Feeder is an IoT-based system designed to automate pet feeding through remote control. The primary components include:

- **NodeMCU ESP8266**: A microcontroller with built-in Wi-Fi, acting as the brain of the system, processing commands and controlling peripherals.
- **Servo Motor**: A mechanical actuator that rotates to open and close a feeder mechanism, dispensing a portion of pet food.
- **16x2 LCD with I2C Module**: A display unit showing system status (e.g., "WiFi Connected," "Feeding…"), reducing the need for external monitoring.
- **Web Application**: A simple HTML/JavaScript interface with a "Feed" button, hosted locally or accessed via a browser, sending HTTP requests to the NodeMCU.

The system operates as follows: The NodeMCU connects to a Wi-Fi network and starts a web server. When the user clicks "Feed" on the web app, an HTTP GET request is sent to the NodeMCU's IP address (e.g., 192.168.108.147/feed). The NodeMCU responds by rotating the servo to 180° for 3 seconds, opening the feeder to release food, then returning it to 0° to close it. The LCD updates with messages like "Feeding…" and "Feed Complete," while the web app confirms the action with an alert.

The project's scope includes hardware assembly, software programming, and network integration, with a focus on reliability and user experience. It was developed using a breadboard for prototyping, making it modular and easy to modify. The feeder mechanism can be adapted to various container designs, such as a hinged flap or rotating dispenser, depending on the pet's food type (e.g., kibble).

Key features include remote accessibility, real-time feedback, and low-cost implementation, making it suitable for pet owners seeking automation without complex setups. The project also serves as an educational tool, demonstrating IoT concepts like Wi-Fi communication, servo control, and I2C interfacing.

4. Methodology

The development of the Smart Pet Feeder followed a structured approach, combining hardware setup, software coding, and iterative testing. Below is a detailed breakdown of the methodology, including the final code.

4.1 Hardware Assembly

- The NodeMCU ESP8266 was connected to a servo motor and an LCD with an I2C module using a breadboard and jumper cables.
- Initial wiring faced challenges with I2C communication and power distribution, resolved through debugging and external power integration.

4.2 Software Development

- The Arduino IDE was used to program the NodeMCU, with libraries for Wi-Fi, web server, servo, and I2C LCD functionality.
- The web app was developed using HTML and JavaScript, hosted locally for simplicity.

4.3 Code Implementation

Here’s the final working code for the NodeMCU:

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <Servo.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Wi-Fi credentials
const char* ssid = "Thor's Apple";
const char* password = "thormjolnir";

// Servo setup
Servo servo;
int servoPin = 2; // D4 on NodeMCU (GPIO 2)

// LCD setup
LiquidCrystal_I2C lcd(0x27, 16, 2); // Confirmed address 0x27

// Web server on port 80
ESP8266WebServer server(80);

void setup() {
  Serial.begin(115200);
  delay(1000); // Wait for Serial Monitor

  // Initialize LCD
  Wire.begin(4, 5); // SDA = GPIO 4 (D2), SCL = GPIO 5 (D1)
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Pet Feeder");
  lcd.setCursor(0, 1);
  lcd.print("Connecting...");

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConnected to Wi-Fi");
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("WiFi Connected");
  lcd.setCursor(0, 1);
  lcd.print(WiFi.localIP());

  // Servo setup
  servo.attach(servoPin);
  servo.write(0); // Initial position (closed)

  // Web server routes
  server.on("/", handleRoot);       // Home page
  server.on("/feed", handleFeed);   // Feed endpoint
  server.begin();
  Serial.println("Server started");
}

void loop() {
  server.handleClient(); // Handle incoming requests
}

void handleRoot() {
  server.send(200, "text/plain", "Pet Feeder Ready");
}
```

```

void handleFeed() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Feeding...");
  Serial.println("Feeding started");
  servo.write(180);  // Rotate to 180° (fully open feeder)
  delay(3000);       // Wait 3 seconds
  servo.write(0);    // Return to 0° (closed position)
  Serial.println("Feeding complete");
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Feed Complete");
  delay(2000);       // Show message for 2 seconds
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Pet Feeder");
  lcd.setCursor(0, 1);
  lcd.print("Idle");
  server.send(200, "text/plain", "Feed dispensed");
}

```

And the web app code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Pet Feeder</title>
</head>
<body>
  <button onclick="feedPet()">Feed</button>

  <script>
    function feedPet() {
      fetch('http://192.168.108.147/feed', { method: 'GET' })
        .then(response => {
          if (!response.ok) throw new Error('Network response was not ok');
          return response.text();
        })
        .then(data => {
          console.log(data);
          alert("Pet fed successfully!");
        })
        .catch(error => {
          console.error('Error:', error);
          alert("Failed to feed pet.");
        });
    }
  </script>
</body>
</html>

```

4.4 Debugging and Optimization

- **I2C Issues**: The LCD initially showed no text due to an incorrect address and contrast setting. An I2C scanner confirmed 0x27, and contrast was adjusted via the potentiometer.
- **Servo Testing**: A separate loop test ensured the servo moved to 180° reliably.
- **Power Management**: The LCD backlight dimmed under load; an external 5V eliminator was added to stabilize power.
- **Network Errors**: Web app failures were traced to IP mismatches and CORS, resolved by verifying the IP and adding a CORS header (optional).

4.5 Testing

- The system was tested by clicking "Feed," observing servo movement, LCD updates, and web app responses, iterating until all components worked seamlessly.

5. Hardware Requirements

The Smart Pet Feeder relies on the following hardware:

- **NodeMCU ESP8266**:
- A Wi-Fi-enabled microcontroller with 11 GPIO pins, 4MB flash, and 3.3V logic.
- Used for Wi-Fi connectivity, servo control, and I2C communication.
- Cost: ~$5-$10.
- **Servo Motor (e.g., SG90)**:
- A small, 180° rotation motor with 1.8kg·cm torque at 5V.
- Controls the feeder mechanism.
- Cost: ~$2-$5.
- **16x2 LCD with I2C Module**:
- A 16-character, 2-line display with an I2C backpack (PCF8574) reducing pin usage.
- Displays system status.
- Cost: ~$3-$7.
- **Breadboard**:
- A prototyping platform for easy connections.
- Cost: ~$2-$5.
- **Jumper Cables**:
- Male-to-male and male-to-female cables for wiring.
- Cost: ~$1-$3.
- **External Power Supply (Eliminator)**:
- A 5V DC adapter (e.g., 1A or 2A) with positive/negative terminals.
- Powers the LCD and servo to avoid NodeMCU overload.
- Cost: ~$5-$10 (if not already owned).

Specifications:

- NodeMCU: 5V input via USB, 3.3V or VIN output.
- Servo: 4.8-6V, ~100-500mA current draw.
- LCD: 5V recommended for I2C module, ~20mA without backlight, ~100mA with backlight.

Considerations:

- The NodeMCU's USB power was insufficient under full load, necessitating the eliminator.
- Components were chosen for affordability and compatibility with IoT projects.

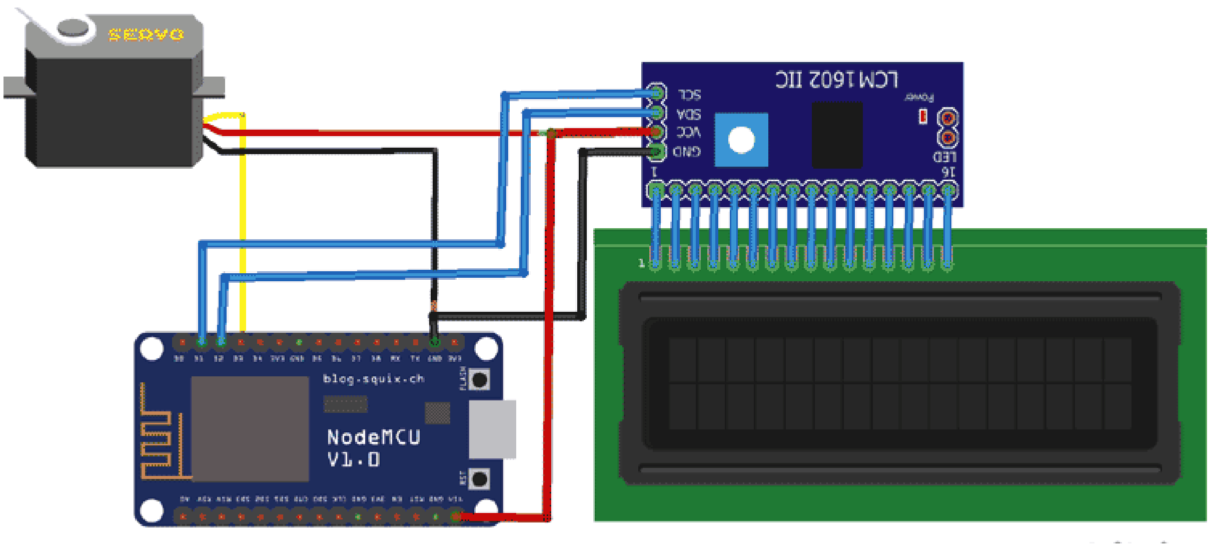
6. Wiring Connection

The wiring setup ensures all components communicate effectively:

- **NodeMCU ESP8266**:
- Powered via USB from a laptop initially, later supplemented by an eliminator.
- **Servo Motor**:
- **Signal (Orange)**: D4 (GPIO 2).
- **VCC (Red)**: Breadboard + rail (5V from eliminator).
- **GND (Black)**: Breadboard − rail (common GND with NodeMCU).
- **16x2 LCD with I2C Module**:
- **SDA**: D2 (GPIO 4).
- **SCL**: D1 (GPIO 5).
- **VCC**: Breadboard + rail (5V from eliminator).
- **GND**: Breadboard − rail (common GND).
- **Eliminator**:
- **Positive (+)**: Breadboard + rail.
- **Negative (−)**: Breadboard − rail, connected to NodeMCU GND.

Wiring Diagram (Text Description):

- Breadboard power rails distribute 5V and GND from the eliminator.
- NodeMCU's D4, D2, and D1 pins control the servo and LCD via signal lines.
- A common GND connects all components to avoid floating voltages.



Challenges:

- Initial dim LCD backlight was resolved by moving VCC to the eliminator.
- I2C pins were confirmed using an I2C scanner to avoid miswiring.

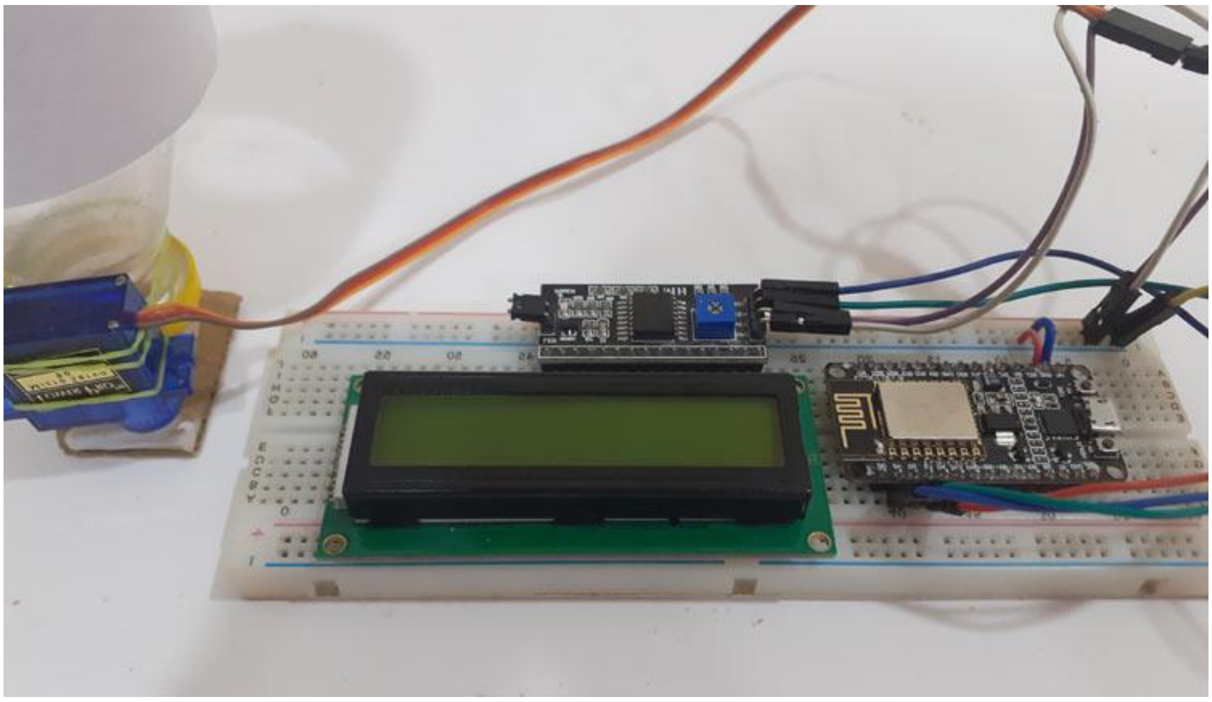
7. Software Requirements

The software stack includes:

- **Arduino IDE**:
- Version: Latest (e.g., 2.2.1).
- Used to program the NodeMCU with C/C++ code.
- ESP8266 board support added via Boards Manager (URL: http://arduino.esp8266.com/stable/package_esp8266com_index.json).
- **Libraries**:
- ESP8266WiFi: For Wi-Fi connectivity.
- ESP8266WebServer: To host the web server.
- Servo: For servo control.
- Wire: For I2C communication.
- LiquidCrystal_I2C (Frank de Brabander): For LCD interfacing.
- **Web App**:
- HTML and JavaScript, run locally or hosted via a simple server (e.g., Python’s http.server).
- No additional frameworks required.

Setup:

- Installed Arduino IDE and added ESP8266 support.
- Libraries were installed via Library Manager.
- Web app was tested in a browser on the same Wi-Fi network.



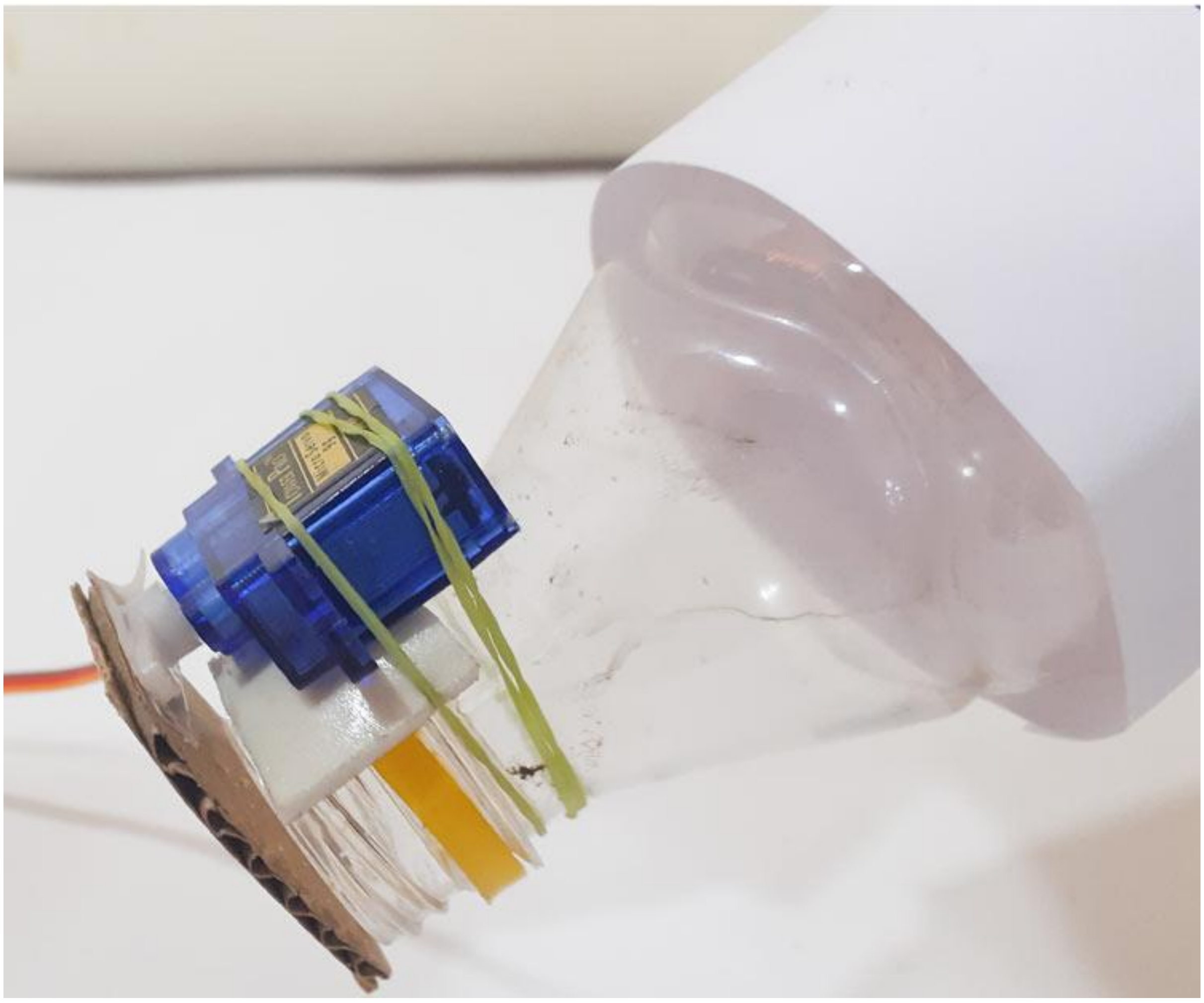
Code Functionality:

- NodeMCU code handles Wi-Fi, servo, LCD, and HTTP requests.
- Web app sends GET requests to trigger feeding.

8. Result

The Smart Pet Feeder was successfully implemented:

- **Wi-Fi Connection**: The NodeMCU reliably connects to "Thor's Apple," displaying the IP (e.g., 192.168.108.147) on the LCD.
- **Servo Operation**: Rotates to 180° for 3 seconds, then back to 0°, dispensing food as intended.
- **LCD Feedback**: Shows "Pet Feeder" / "Connecting…", "WiFi Connected" / IP, "Feeding…", "Feed Complete," and "Pet Feeder" / "Idle" in sequence.
- **Web App**: Triggers feeding when the "Feed" button is clicked, with a confirmation alert ("Pet fed successfully!") when the IP matches and network conditions are met.



Testing Outcomes:

- Initial LCD issues (no text, dim backlight) were fixed with I2C address confirmation, contrast adjustment, and an external 5V eliminator.
- Servo movement was verified with a loop test, ensuring 180° capability.
- Web app bugs (e.g., "Failed to feed") were traced to IP mismatches, resolved by updating the fetch URL.

Performance:

- Response time: ~3-4 seconds from button click to completion.
- Reliability: Consistent over multiple tests when powered adequately.

9. Conclusion

The Smart Pet Feeder project demonstrates a practical application of IoT for pet care automation. By integrating the NodeMCU ESP8266, servo motor, LCD, and a web interface, it achieves remote feeding with real-time feedback at a low cost. The methodology overcame challenges like power management, I2C setup, and network communication, resulting in a robust prototype.

The system's strengths include its simplicity, affordability, and modularity, making it adaptable for various feeder designs. Limitations include reliance on a stable Wi-Fi network and manual IP updates if the router reassigns addresses. Future enhancements could include a mobile app, scheduled feeding via an RTC module, or a food level sensor.

This project highlights the potential of IoT in everyday solutions, offering pet owners convenience and peace of mind. It also serves as a learning platform for embedded systems and network programming.