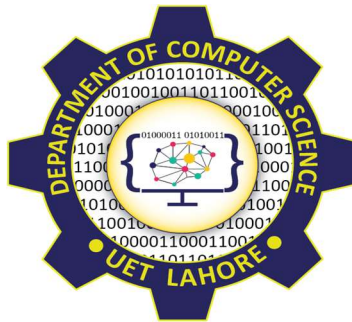


Smart Bin



Session 2023-2027

Submitted By:

Amir Hashmi	2023-CS-11
Hania Arshad	2023-CS-13
Sher Muhammad	2023-CS-15
Muhammad Tayyab	2023-CS-101
Muhammad Harris	2023-CS-160

Supervised By:

Sir Tehseen

Course:

Computer Organization and Assembly Language

Department of Computer Science

University of Engineering and Technology

Lahore

Smart Bin

Table of Contents

1. Introduction	1
1.1 Problem Statement	1
1.2 Objectives and Goals	1
2. System Architecture	1
2.1 High-Level System Overview	1
2.1.1 ESP32 Microcontroller	1
2.1.2 Arduino UNO	2
2.1.3 Sensors and Actuators	2
2.1.4 Cloud Integration	2
2.2 Component Interaction Diagram:	2
2.3 Data Flow Diagram:	3
2.4 Communication Protocols Used:	4
2.4.1 MQTT Protocol	4
2.4.2 UART Serial Communication	4
3. Hardware Components	4
3.1 Power Supply Requirements	4
3.2 Component List and Specifications:	5
3.3 Physical Bin Construction:	6
4. Hardware Implementations	6
4.1 Circuit Diagram and Schematics	6
4.2 Power Distribution	6
4.2.1 ESP32 Power:	6
4.2.2 Arduino Power:	6
4.3 Physical Assembly and Mounting	7
4.3.1 Sensor Positioning:	7
4.3.2 Servo Mounting:	7
4.3.3 Controller Placement:	7
4.3.4 Wiring Considerations:	7

Smart Bin

5. Software Implementation	7
5.1 ESP32 Code Overview	7
5.1.1 Wi-Fi and MQTT Configuration:	8
5.1.2 Ultrasonic Sensor Logic:	8
5.1.3 Fill Level Calculation Algorithm:	9
5.1.4 Proximity Detection Algorithm:	9
5.1.5 UART Communication with Arduino:	10
5.2 Arduino AVR Assembly Code Explanation	11
5.2.1 Timer Configuration for PWM:	11
5.2.2 UART Reception Handling:	11
5.2.3 Servo Control Logic:	12
5.3 Software Flowcharts	13
5.3.1 ESP32 Main Loop Flowchart:	13
5.3.2 Arduino AVR Assembly Flowchart:	14
6. Testing and Validation:	15
6.1 Test Methodology:	15
6.1.1 Component-level testing:	15
6.1.2 Subsystem testing:	15
6.1.3 System-level testing:	15
6.2 Fill Level Sensor Accuracy Testing:	15
6.3 Proximity Detection Reliability Testing:	15
6.4 Servo Motor Operation Testing:	15
7. Challenges Faced:	16
8. Future Improvement:	16
9. Wireframes:	17
10. Conclusion:	18
11. References:	18
12. Project Links:	19

1. Introduction

1.1 Problem Statement

Conventional waste bins present several issues:

- Physical contact requirement increases contamination risk
- No real-time fill-level information leads to inefficient collection schedules
- No integration with modern IoT monitoring systems
- Manual monitoring of bin status requires dedicated personnel

1.2 Objectives and Goals

The Smart Bin System aims to address these challenges through the following objectives:

- Implement touchless operation through proximity sensing to eliminate the need for physical contact
- Develop accurate fill-level monitoring to provide real-time data on bin capacity
- Integrate with IoT platforms via MQTT for remote monitoring and management

2. System Architecture

2.1 High-Level System Overview

The Smart Bin System employs distributed architecture with specialized controllers for different functions:

2.1.1 ESP32 Microcontroller

Serves as the central processing unit responsible for:

- Wi-Fi connectivity and MQTT communication
- Ultrasonic sensor management (both proximity and fill-level)
- System State Management
- Communication with the Arduino control unit

Smart Bin

2.1.2 Arduino UNO

Functions as a dedicated servo controller:

- Receives commands from ESP32 via UART
- Controls the bin lid servo motor using precise PWM signals
- Implemented in optimized AVR Assembly for responsive and reliable operation

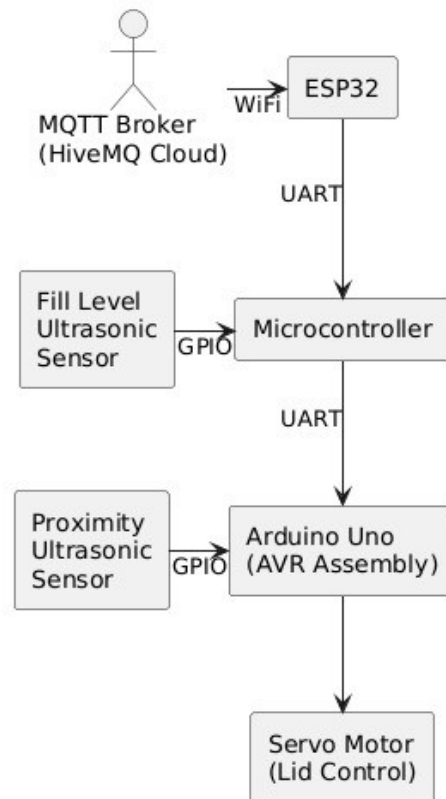
2.1.3 Sensors and Actuators

- Two HC-SR04 ultrasonic sensors for:
 - Proximity detection (hand presence)
 - Fill level measurement
- Servo motor for mechanical lid control

2.1.4 Cloud Integration

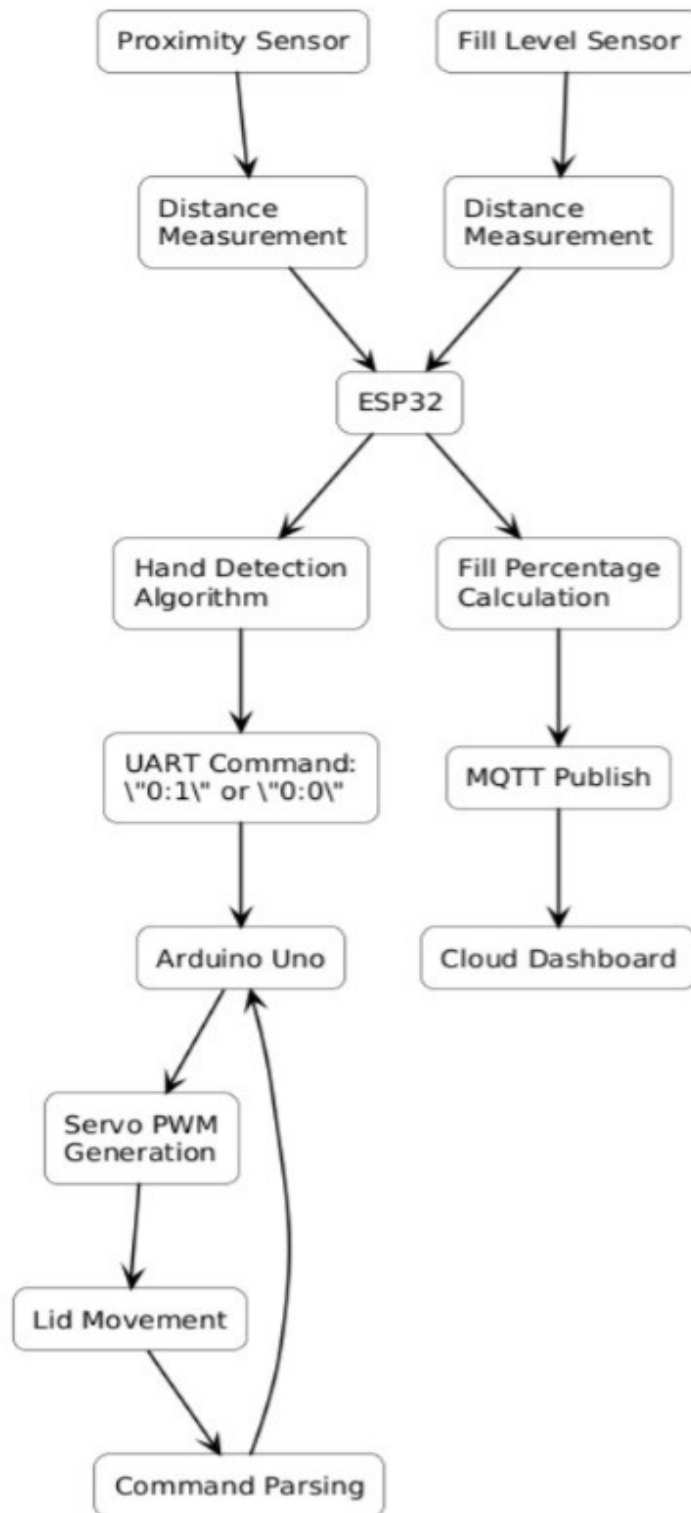
The system uses HiveMQ as the MQTT broker for data transmission and features a dashboard for visualizing the bin status.

2.2 Component Interaction Diagram:



Smart Bin

2.3 Data Flow Diagram:



Smart Bin

2.4 Communication Protocols Used:

The system utilizes two primary communication protocols:

2.4.1 MQTT Protocol

The communication between the ESP32 and cloud services is facilitated using a structured topic-based system. Two key MQTT topics are utilized: **smartdustbin/filllevel**, which reports the bin's fill percentage ranging from 0 to 100%. To ensure efficient real-time updates, QoS (Quality of Service) Level 0 is used, minimizing latency and overhead. Additionally, retained messages are enabled to maintain the last known state of the system, allowing new subscribers to immediately receive the status upon connection.

2.4.2 UART Serial Communication

is used for inter-controller communication between the ESP32 and Arduino. The communication is configured with a baud rate of 9600, 8 data bits, no parity, and 1 stop bit. Commands follow a simple structure to ensure clarity and reliability. For example, O:1 is used to send a command to open the lid, while O:0 instructs it to close. Additionally, the D:xx.x command format is used to transmit distance measurements, serving as informational data for the system.

3. Hardware Components

3.1 Power Supply Requirements

The system requires two different voltage levels for its components: the ESP32 operates at 3.3V, while the Arduino Uno runs at 5V. Supplying both devices with the same voltage, especially giving the ESP32 5V directly, can damage the ESP32 since it is not designed to handle higher voltages. Conversely, powering the Arduino with only 3.3V would likely cause it to malfunction or fail to operate properly, as it requires a stable 5V supply. Therefore, maintaining their specified voltage levels is crucial for reliable and safe operation.

Smart Bin

3.2 Component List and Specifications:

Component	Quantity	Specifications	Function
ESP32 Development Board	1	Dual-core 240MHz, 520KB SRAM, WiFi/BT, 36 GPIO	Main controller, WiFi connectivity, sensor management
Arduino Uno	1	ATmega328P, 16MHz, 32KB Flash, 2KB SRAM	Servo control via AVR Assembly
HC-SR04 Ultrasonic Sensor	2	Range: 2–400cm, Resolution: $\pm 0.3\text{cm}$	Proximity detection and fill level measurement
SG90 Servo Motor	1	Operating voltage: 4.8–6V, Torque: 1.8kg/cm, Rotation: 180°	Lid control mechanism
Breadboard	1	Standard 830 point solderless breadboard	Prototyping and connections
Jumper Wires	~20	Male-to-male, male-to-female	Component connections
Power Supply	2	5V/2A for Arduino, 3.3V for ESP32	System power
Status LED	1	Standard LED with current-limiting resistor	Visual status indication
Waste Bin	1	Height: 18.0cm (internal)	Modified with servo mount and sensors

Smart Bin

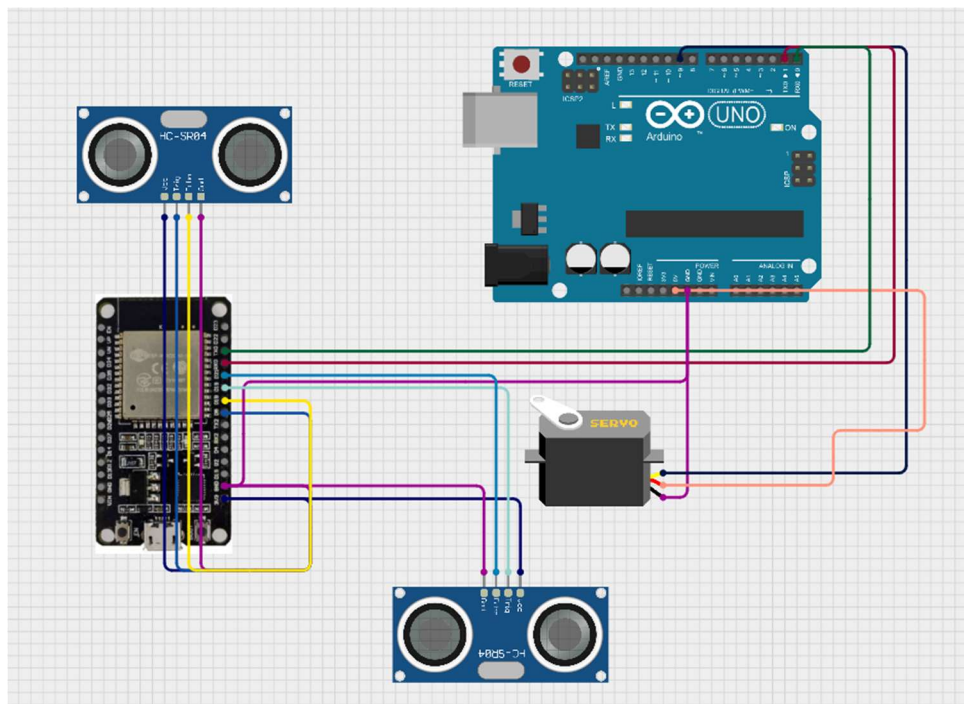
3.3 Physical Bin Construction:

The project utilizes a standard waste bin with the following modifications:

- **Lid Mechanism:** Modified to attach to servo motor arm
- **Sensor Mounting:** Internal bracket for fill-level sensor at top of bin
- **Internal Dimensions:** Height of 18.0cm used for fill-level calculation

4. Hardware Implementations

4.1 Circuit Diagram and Schematics



4.2 Power Distribution

The system uses independent power sources for the ESP32 and Arduino:

4.2.1 ESP32 Power:

- Powered via USB connection or external 3.3V regulated supply
- Provides 3.3V to both ultrasonic sensors

4.2.2 Arduino Power:

- Powered via separate USB connection or external 5V supply
- Provides 5V to the servo motor

Smart Bin

This separation prevents:

- Ground loop issues
- Voltage inconsistencies between controllers
- Power fluctuations when servo operates

4.3 Physical Assembly and Mounting

4.3.1 Sensor Positioning:

The fill level sensor is installed at the top inside part of the bin, facing downward to accurately detect the amount of waste. The proximity sensor is mounted on the front exterior of the bin, facing outward to detect when a user approaches.

4.3.2 Servo Mounting:

The servo motor is mounted at the hinge of the bin lid and is connected to the lid through a mechanical linkage. Its position is chosen to provide maximum torque efficiency for smooth and reliable lid operation.

4.3.3 Controller Placement:

Both the ESP32 and Arduino are housed inside a weatherproof enclosure, placed away from the waste material to prevent damage. Their placement also ensures easy access for future maintenance or reprogramming.

4.3.4 Wiring Considerations:

All wiring is routed carefully with proper strain relief to prevent disconnections or wear. Interference is minimized through organized wiring, and each connection is labeled clearly to simplify troubleshooting and maintenance.

5. Software Implementation

5.1 ESP32 Code Overview

The ESP32 firmware is written in C using the Arduino framework and serves as the primary intelligence of the system. Key components include:

Smart Bin

5.1.1 Wi-Fi and MQTT Configuration:

The system connects to a Wi-Fi network and uses the HiveMQ public MQTT broker to publish data. It assigns a unique client ID and uses separate topics to report the fill level and lid status.

```
1  const char* WIFI_SSID = "TECNOSPARK4";
2  const char* WIFI_PASSWORD = "Amir0017";
3
4  const char* MQTT_BROKER = "broker.hivemq.com";
5  const int MQTT_PORT = 1883;
6  const char* CLIENT_ID = "smartbin_017";
7  const char* TOPIC_FILL_LEVEL = "smartdustbin/filllevel";
8  const char* TOPIC_LID_STATE = "smartdustbin/lidstate";
```

5.1.2 Ultrasonic Sensor Logic:

The system uses the standard HC-SR04 working method: it sends a 10µs trigger pulse, measures how long the echo pulse lasts, and calculates the distance using the speed of sound.

```
1  float measureDistance(int trigPin, int echoPin) {
2      digitalWrite(trigPin, LOW);
3      delayMicroseconds(2);
4
5      digitalWrite(trigPin, HIGH);
6      delayMicroseconds(10);
7      digitalWrite(trigPin, LOW);
8
9      long duration = pulseIn(echoPin, HIGH, 15000);
10
11     float distance = duration * 0.034 / 2;
12
13     if (duration == 0 || distance < 0.5 || distance > 400) {
14         return -1;
15     }
16
17     return distance;
18 }
```

Smart Bin

5.1.3 Fill Level Calculation Algorithm:

The algorithm:

1. Measures the distance from the sensor (top of bin) to the waste surface
2. Calculates the filled height by subtracting from total bin height
3. Converts to percentage of total bin capacity
4. Applies bounds checking to ensure values remain within 0-100%

```
1 void measureAndPublishFillLevel() {  
2   float currentHeight = measureDistance(FILL_TRIG_PIN, FILL_ECHO_PIN);  
3  
4   if (currentHeight <= 0 || currentHeight > BIN_HEIGHT) {  
5     publishFillLevel(DEFAULT_FILL_LEVEL, false);  
6     return;  
7   }  
8  
9   float fillPercentage = ((BIN_HEIGHT - currentHeight) / BIN_HEIGHT) * 100;  
10  
11   if (fillPercentage < 0) fillPercentage = 0;  
12   if (fillPercentage > 100) fillPercentage = 100;  
13  
14   publishFillLevel(fillPercentage, false);  
15 }
```

5.1.4 Proximity Detection Algorithm:

The algorithm:

1. Takes multiple distance readings to avoid false triggers
2. Calculates an average distance
3. Compares against a defined threshold (10cm)
4. Sends open/close commands based on proximity detection.

Smart Bin

```
1 void checkProximityAndControlLid() {
2   float distances[3];
3   float avgDistance = 0;
4
5   for (int i = 0; i < 3; i++) {
6     distances[i] = measureDistance(PROX_TRIG_PIN, PROX_ECHO_PIN);
7     if (distances[i] > 0) {
8       avgDistance += distances[i];
9     } else {
10      i--;
11      delay(10);
12      continue;
13    }
14    delay(10);
15  }
16
17  avgDistance /= 3;
18
19  Serial.print("D:");
20  Serial.println(avgDistance);
21
22  if (avgDistance < LID_PROXIMITY_THRESHOLD && !isLidOpen) {
23    Serial.println("O:1");
24    isLidOpen = true;
25    publishLidState(isLidOpen, false);
26  }
27  else if (avgDistance >= LID_PROXIMITY_THRESHOLD && isLidOpen) {
28    Serial.println("O:0");
29    isLidOpen = false;
30    publishLidState(isLidOpen, false);
31  }
32 }
```

5.1.5 UART Communication with Arduino:

Each command starts with a single letter to indicate the type ('O' for lid operation, 'D' for distance). This is followed by a colon (':'), then the command parameter—either '1' to open, '0' to close, or a numeric distance value. Every command ends with a newline character to separate it from the next message.

```
1 Serial.print("D:");
2 Serial.println(avgDistance);
3
4 Serial.println("O:1");
5
6 Serial.println("O:0");
```

5.2 Arduino AVR Assembly Code Explanation

5.2.1 Timer Configuration for PWM:

This code sets up Timer1 in Fast PWM mode using the ICR1 register to define the TOP value. It applies a prescaler of 8 to get accurate timing and configures ICR1 to generate a 50Hz PWM signal, which is standard for controlling servo motors. The PWM output is sent through Output Compare pin OC1A (Pin 9).

```
1  Timer1_init:
2      ldi r16, (1<<COM1A1) | (1<<WGM11)
3      sts TCCR1A, r16
4
5      ldi r16, (1<<WGM13) | (1<<WGM12) | (1<<CS11)
6      sts TCCR1B, r16
7
8      ldi r16, LOW(40000)
9      ldi r17, HIGH(40000)
10     STSW ICR1H, r17, r16
11
12     ret
```

5.2.2 UART Reception Handling:


This interrupt handler reads characters received via UART and stores them in a buffer until it detects a newline character. Once a full command is received, it sets a flag to signal that the command is ready to be processed.

```
1  USART_RX_complete:
2      lds r16, UDR0
3      cpi r16, 0x0A
4      breq end_of_command
5      lds r17, buffer_index
6
7      ldi r30, LOW(UART_BUFFER)
8      ldi r31, HIGH(UART_BUFFER)
9      add r30, r17
10     st Z, r16
11
12     inc r17
13     sts buffer_index, r17
14     rjmp rx_exit
15
16  end_of_command:
17     ldi r16, 1
18     sts cmd_ready, r16
```

Smart Bin

5.2.3 Servo Control Logic:

The servo control logic sets the OCR1A register to either the open or closed position based on predefined constants—4000 for open (2ms pulse) and 2000 for closed (1ms pulse). It adds a delay to ensure the servo finishes moving, toggles a status LED for feedback, and sends an acknowledgment back to the ESP32.

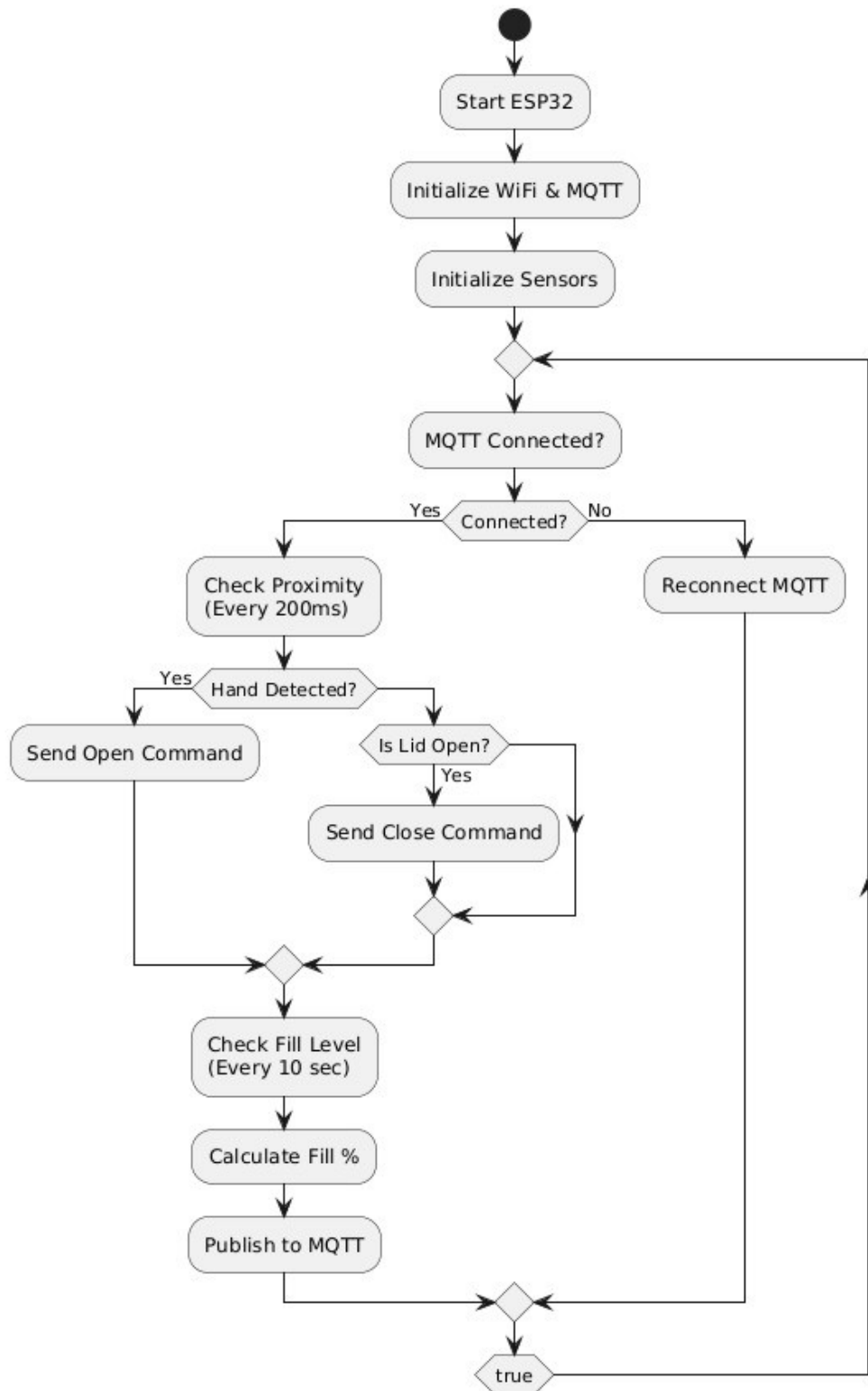


```
1  do_open_servo:
2      ; Send acknowledgment
3      ldi ZL, LOW(2*open_msg)
4      ldi ZH, HIGH(2*open_msg)
5      rcall UART_send_string
6      ; Turn on status LED
7      sbi PORTB, 5
8      ; Set servo to open position
9      ldi r16, LOW(SERVO_OPEN)
10     ldi r17, HIGH(SERVO_OPEN)
11     STSw OCR1AH, r17, r16
12     delay 1000          ; Wait for servo to move
13     rjmp finish_command
14
15  do_close_servo:
16      ; Send acknowledgment
17      ldi ZL, LOW(2*close_msg)
18      ldi ZH, HIGH(2*close_msg)
19      rcall UART_send_string
20      ; Set servo to closed position
21      ldi r16, LOW(SERVO_CLOSED)
22      ldi r17, HIGH(SERVO_CLOSED)
23      STSw OCR1AH, r17, r16
24      delay 1000          ; Wait for servo to move
25      ; Turn off status LED
26      cbi PORTB, 5
27      rjmp finish_command
```


Smart Bin

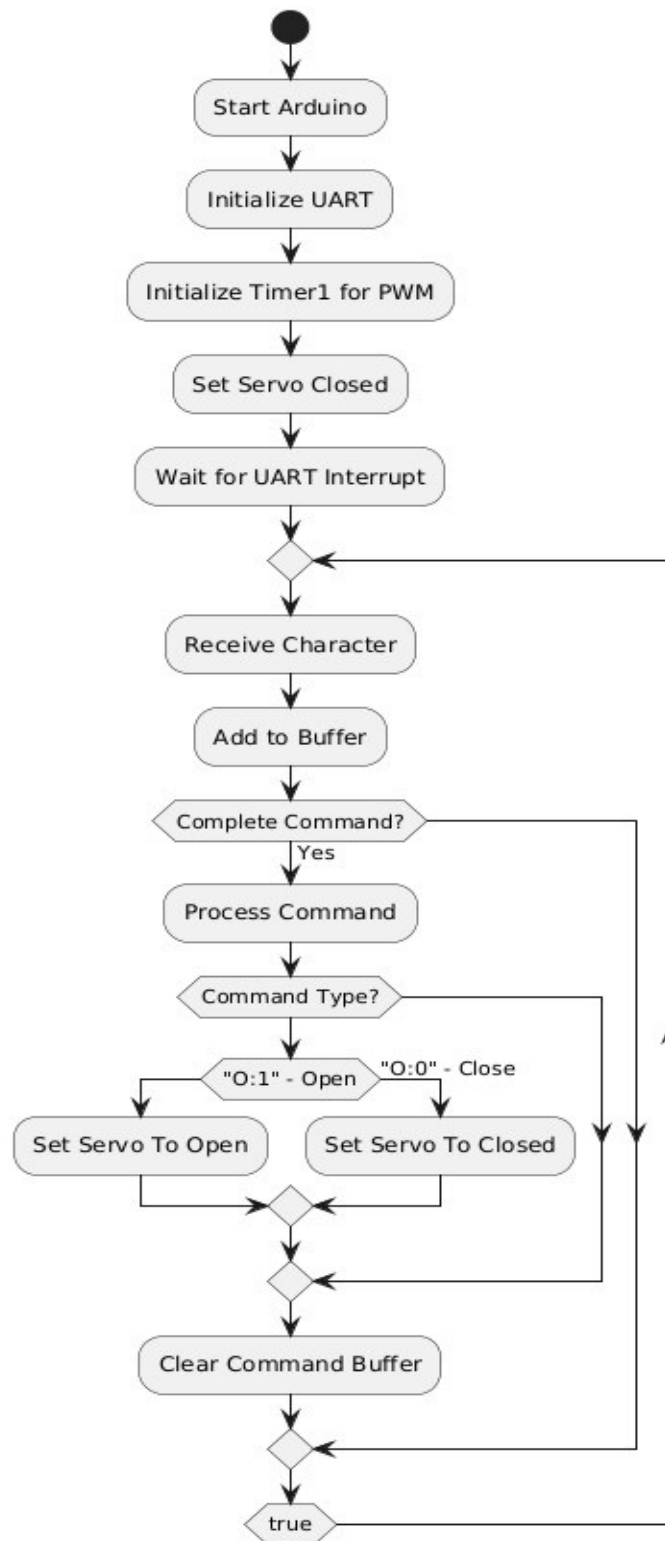
5.3 Software Flowcharts

5.3.1 ESP32 Main Loop Flowchart:



Smart Bin

5.3.2 Arduino AVR Assembly Flowchart:



6. Testing and Validation:

6.1 Test Methodology:

The testing methodology for this system followed a systematic approach to ensure all components functioned correctly both individually and as an integrated system. Testing was conducted in phases:

6.1.1 Component-level testing:

Each hardware component (sensors, servo motor, microcontrollers) was tested individually to verify basic functionality.

6.1.2 Subsystem testing:

Integrated testing of related components (e.g., ESP32 with ultrasonic sensors, Arduino with servo motor).

6.1.3 System-level testing:

Complete system validation in both controlled and real-world environments.

6.2 Fill Level Sensor Accuracy Testing:

The ultrasonic sensor was tested at various bin fill levels (0% to 100%) and with different waste types. Results showed $\pm 3\%$ accuracy, with slight deviations on irregular surfaces. Averaging helped improve reliability.

6.3 Proximity Detection Reliability Testing:

The hand detection system was evaluated for range, false triggers, and response time. It achieved over 95% reliability with averaging and delay mechanisms reducing false positives effectively.

6.4 Servo Motor Operation Testing:

Servo operation was tested for angle precision, timing, power usage, and durability (500+ cycles). The system performed consistently, with delays ensuring stable motion and responsiveness.

6.5 MQTT Communication Testing:

MQTT functionality was tested for connection stability, reconnection, message accuracy, and latency. Messages were reliably delivered in under 200ms, with reconnection logic handling brief outages.

7. Challenges Faced:

- We accidentally burned the ESP32 by powering it directly with the Arduino's 5V line. This was fixed by adding proper voltage regulation and level shifting.
- The small servo motor couldn't open a large bin lid due to limited torque, so we switched to a smaller bin to reduce the load.
- Ultrasonic sensors gave inconsistent readings on some waste materials, which we improved by averaging multiple measurements.
- Network interruptions caused MQTT disconnects that needed manual resets sometimes.

8. Future Improvement:

- Use proper voltage level shifters for safer hardware connections.
- Try better or alternative sensors for more accurate fill level detection.
- Improve MQTT reliability with offline data caching and better reconnection logic.
- Add remote monitoring to quickly detect and fix issues.

Smart Bin

9. Wireframes:

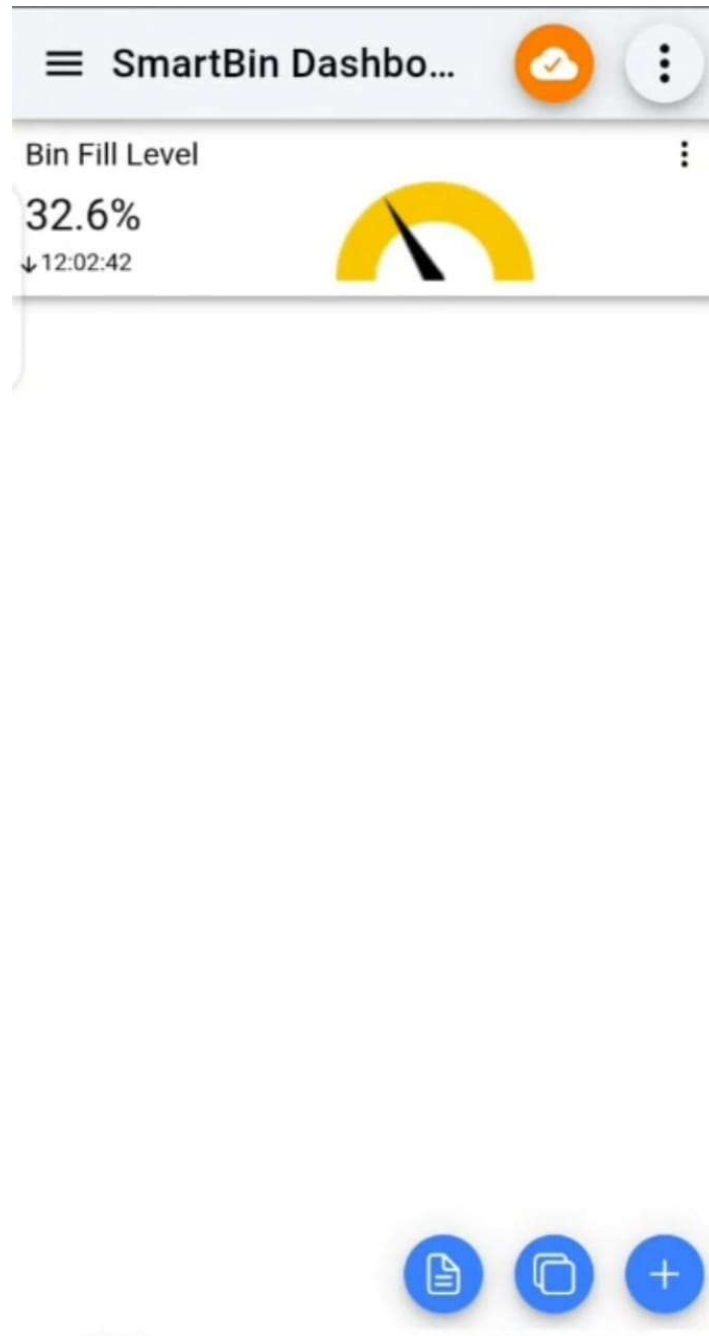


Fig. MQTT Dashboard

Smart Bin



Fig. Smart Bin

10. Conclusion:

This report presents the design and implementation of a Smart Bin System. The system offers touchless lid operation and real-time fill-level monitoring, enhancing both user convenience and operational efficiency. By employing an ESP32 microcontroller as the primary processor and an Arduino Uno running AVR assembly to handle servo control, the project effectively demonstrates a multi-controller setup with dedicated task management. Ultrasonic sensors enable reliable proximity detection and fill-level measurement, while MQTT communication allows remote monitoring via IoT dashboards. Testing confirmed accurate proximity detection within $\pm 1\text{cm}$ and fill-level measurement within $\pm 5\%$. Overall, the system meets its design goals, offering a cost-effective and practical solution using readily available components.

11. References:

1. <https://circuitdigest.com/microcontroller-projects/iot-based-smart-bin>

12. Project Links:

- <https://github.com/AmirHashmi017/COAL-Project>
- https://www.linkedin.com/posts/amirhashmi017_iot-smartcities-arduino-activity-7329163392621924352-m_Zt?utm_source=share&utm_medium=member_desktop&rcm=ACoAAEmOux8BaFL03tcQVob8HYdr7PBz61GvTGc
- <https://youtu.be/m3SjOxRbfSE>