



الجامعة الإسلامية العالمية ماليزيا
INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA
يُونِيسْتِي إِسْلَامُ أَنْتَارَايَحْسَا مِلْدِسِيَا

Garden of Knowledge and Virtue

MINI PROJECT REPORT: WASHING MACHINE

GROUP 4

MCTA 3203

SEMESTER 1 2024/2025

MECHATRONICS SYSTEM INTEGRATION

DATE OF SUBMISSION: 16 JANUARY 2025

NO	NAME	MATRIC NUMBER
1.	IRDINA NABIHAH BINTI MOHD NAZRI	2214772
2.	KHALISAH AMANI BINTI YAHYA AZMI	2218184
3.	LUQMAN AZFAR BIN AZMI	2219857
4.	MOHAMAD NASRI BIN MOHAMAD NAZRI	2219879
5.	MUHAMAD NURHAKIMIE THAQIF BIN ABDULLAH	2213217

TABLE OF CONTENT

NO.	CONTENT
1.	INTRODUCTION
2.	ABSTRACT
3.	MATERIALS AND EQUIPMENT
4.	EXPERIMENTAL SETUP
5.	METHODOLOGY
6.	RESULT
7.	DISCUSSION
8.	CONCLUSION
9.	RECOMMENDATIONS
10.	ACKNOWLEDGEMENTS
11.	STUDENT'S DECLARATION

INTRODUCTION

Laundry management has undergone significant advancements with the integration of modern technologies, offering enhanced convenience, efficiency, and a better user experience. This project aims to design and develop an intelligent washing machine system that incorporates RFID (Radio-Frequency Identification) technology, sensors, displays, buzzers, LEDs, and other electronic components. The primary objective is to create a prototype that simplifies laundry operations while maintaining a user-friendly interface and ensuring high functionality.

The concept of a smart washing machine is grounded in leveraging IoT (Internet of Things) and automation technologies to improve performance and usability. The goals of this project include automating laundry tasks, optimizing energy and water consumption, providing intuitive user interactions, and enhancing security through user and laundry identification using RFID technology.

By automating tasks, reducing manual intervention, and optimizing energy and water usage, this project aims to demonstrate the potential of smart technologies in improving household appliances. The integration of RFID and IoT technologies not only addresses common pain points in laundry management but also showcases the feasibility of developing cost-effective, scalable, and efficient solutions. This project envisions a future where intelligent washing machines are an integral part of smart homes, transforming the way laundry is managed and setting a new standard for convenience and efficiency in daily life.

ABSTRACT

This experiment focuses on developing a smart washing machine system that integrates RFID technology with a master-slave Arduino microcontroller setup to enable automated laundry management and enhance user convenience. The primary objective is to create an intelligent framework capable of real-time laundry identification, monitoring, and personalized control. RFID tags are utilized to identify clothes or users, automating the selection of appropriate washing cycles and settings as well as using it as a token. Supporting this setup, various sensors such as water level and temperature sensors are employed for their precision and reliability in optimizing resource usage and regulating washing conditions.

The system incorporates a master-slave communication architecture using Arduino microcontrollers, where the master Arduino oversees system operations and the slave units handle specific tasks, such as RFID tag processing, sensor data acquisition, and output control. A Pixy camera is integrated to detect the presence of clothes in the washing machine and identify fabric color, enabling the system to optimize washing cycles and prevent issues like color bleeding. The master Arduino processes data from the Pixy camera, LCD display, sensors, and RFID units to adjust operations and provide real-time feedback. Visual and audible notifications, delivered through LED indicators and a buzzer, enhance user interaction by signaling events such as cycle completion, errors, or imbalances. This integration of advanced technologies ensures a seamless, efficient, and user-friendly laundry management system.

METHODOLOGY

This project employed a systematic approach to design a smart washing machine prototype that integrates advanced technologies for efficient and user-friendly laundry management. The experimental setup began with the integration of hardware components, which were connected and configured using breadboards and jumper wires to facilitate a modular design. The system employed a master-slave microcontroller configuration, with the Arduino Mega functioning as the master and the Arduino Uno as the slave. The master Arduino Mega was responsible for managing overall system operations, while the slave Arduino Uno handled specific tasks, including processing data from the LM35 temperature sensor. The temperature sensor was connected to the Arduino Uno to monitor and regulate washing conditions, ensuring precise control during operation.

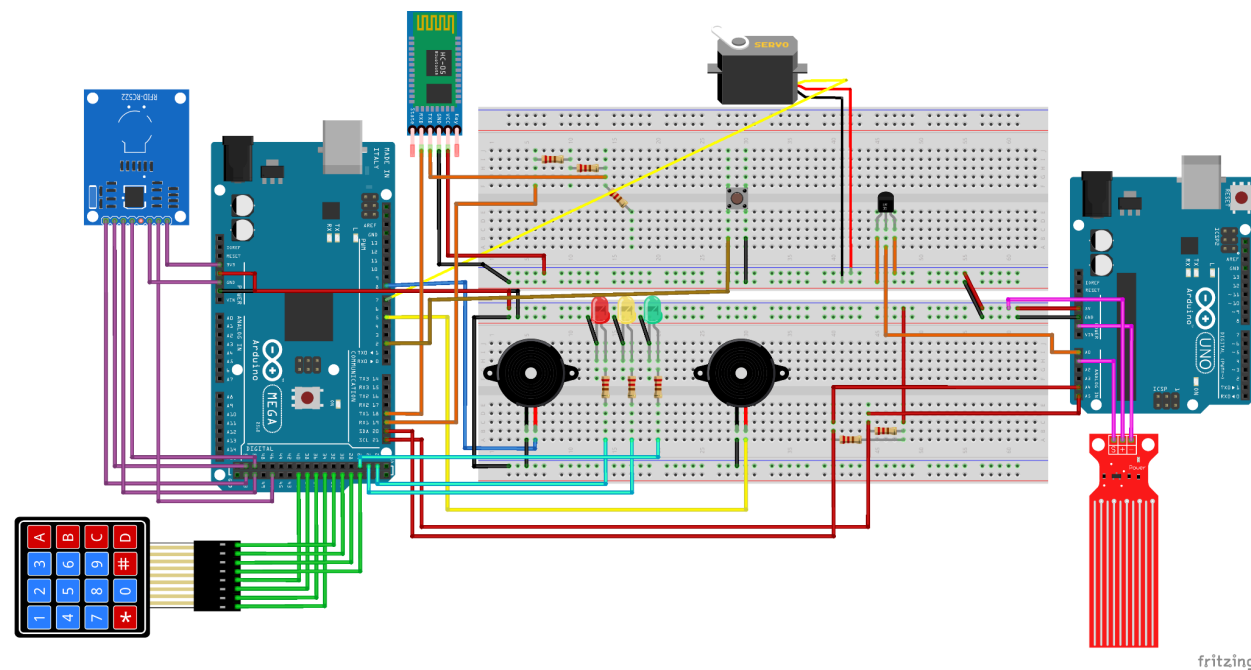
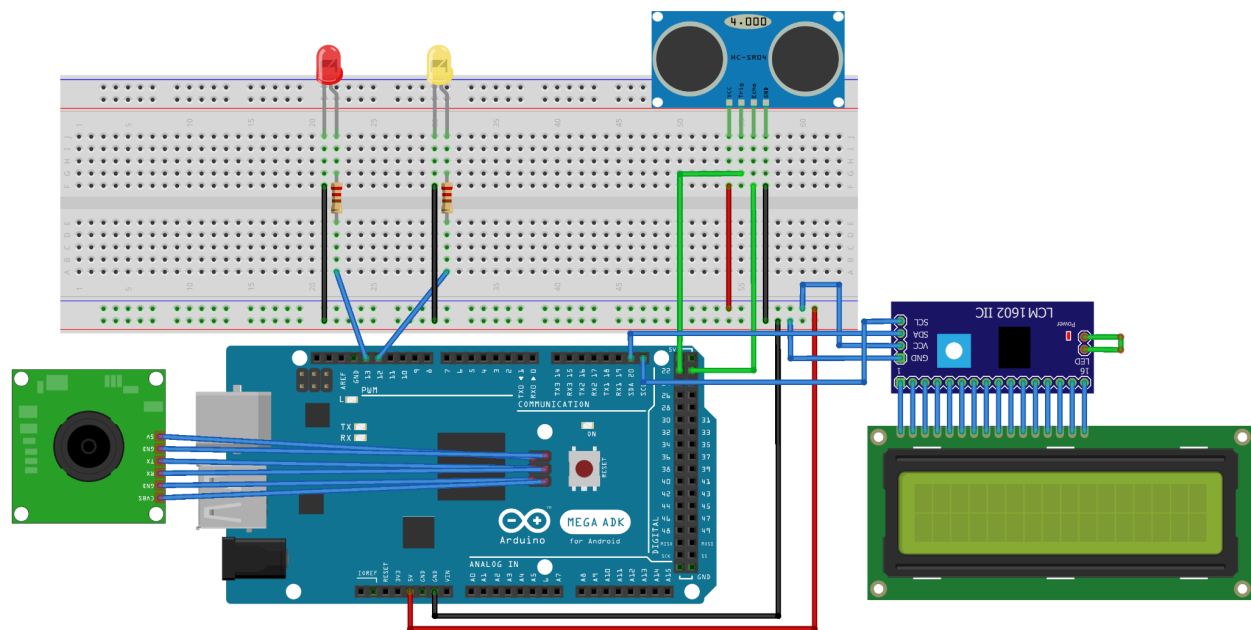
the ultrasonic sensor and water level sensor, played a critical role in ensuring the system's operational reliability. The ultrasonic sensor detected the presence of clothes and identified fabric color, enabling the machine to select appropriate washing cycles and prevent color bleeding. The water level sensor monitored and controlled the water levels, ensuring the washing process only began when adequate water was available. The LM35 temperature sensor, connected to the slave Arduino Uno, provided real-time temperature readings to optimize washing conditions.

The PixyCam was integrated to enhance garment detection capabilities by identifying the presence and color of clothes. This information was processed by the another Arduino Uno, which dynamically adjusted washing settings to optimize performance. The user interface was designed to be intuitive and interactive, featuring an LCD display for real-time feedback and a membrane switch keypad for manual input. Additional input methods included Serial Monitor and Bluetooth Terminal via the HC-06 module. Visual and auditory notifications were provided through LEDs and buzzers, signaling key events such as cycle completion, errors, or system malfunctions. The servo motor was employed to provide visual confirmation of system activation and operational status. The washing machine's operations were orchestrated by combining these components into a cohesive system, with all hardware and software elements integrated to achieve seamless functionality.

MATERIALS AND EQUIPMENT

- 3x Breadboard
- Jumper wires
- USB cables
- Arduino Mega 2560 microcontroller
- 2x Arduino Uno R3 microcontroller
- LCD display (16x2 I2C)
- LED
- RFID Module: (MFRC522)
- BT Module: HC-06
- 2x Buzzer
- Temp Sensor: LM 35
- Pixycam
- Ultrasonic Sensor
- Push button
- Resistors
- 4x4 Membrane Switch Keypad
- Servo motor
- Water Level Sensor

EXPERIMENTAL SETUP



RESULTS

Master Code (System1) :

Library & Setup:

```
// Master Code using Arduino ATmega 2560

// Arduino IDE version 2.3.4

#include <SPI.h>
#include <MFRC522.h>
#include <Servo.h>
#include <Wire.h>
#include <Keypad.h>
```

Easy MFRC522 by Pablo Sompano

MFRC522 by Github Community

Keypad by Mark Stanley

We are using ATmega for this Master Code, SPI.h is used for establishing SPI protocol with RFID RC522 which uses SPI Protocol to communicate with arduino and allow bluetooth to be used and MFRC522.h is used to make the command for RFID easier to establish. Servo.h is used to control the servo to indicate washing machine state. Wire.h is used to establish I2C connection via wire through SDA SCL connection on arduino to establish Master Slave protocol. Lastly, keypad.h is used to use the keypad membrane to allow user to input

```
// Initialize Servo
Servo washingMachineServo;

// Pin definitions
#define SLAVE_ADDRESS 0x08 // I2C address for slave
#define BT_BAUD_RATE 9600 // Baud rate for Bluetooth communication

#define GREEN_LED_PIN 26 // Green LED for "Ready"
#define YELLOW_LED_PIN 24 // Yellow LED for "Washing"
#define RED_LED_PIN 22 // Red LED for "Emergency Stop"
#define RST_PIN 47 // Reset pin for RFID module
#define SS_PIN 53 // Slave select pin for RFID module
#define WASH_MACHINE_PIN 7 // Pin to control washing machine
#define STOP_BUTTON_PIN 2 // Digital pin connected to the stop button
```



```

// Global variables
volatile bool stopFlag = false; // Flag to detect emergency stop
bool isCardAuthenticated = false; // Tracks card authentication state
MFRC522 rfid(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;

// Buzzer and melody settings
#define BUZZER_PIN 8
#define BUZZER_PIN_2 5

const int startMelody[] = {440, 494, 523, 587, 659, 0, 659, 587, 523, 494, 440};
const int startNoteDurations[] = {500, 500, 500, 500, 500, 275, 500, 500, 500, 500, 500};

const int stopMelody[] = {880, 988, 1047, 1175, 1319};
const int stopNoteDurations[] = {300, 300, 300, 300, 300};

// Keypad configuration
const byte ROWS = 4; // Number of rows
const byte COLS = 4; // Number of columns

char keys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

byte rowPins[ROWS] = {33, 31, 29, 27}; // Row pins
byte colPins[COLS] = {41, 39, 37, 35}; // Column pins

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

void setup() {
  // Pin configurations
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(BUZZER_PIN_2, OUTPUT);
  pinMode(STOP_BUTTON_PIN, INPUT_PULLUP); // Enable internal pull-up resistor
  attachInterrupt(digitalPinToInterrupt(STOP_BUTTON_PIN), stopButtonISR, FALLING); // Interrupt on button press
}

```

```

pinMode(GREEN_LED_PIN, OUTPUT);
pinMode(YELLOW_LED_PIN, OUTPUT);
pinMode(RED_LED_PIN, OUTPUT);

// Set initial LED states: Indicate system is ready at startup
digitalWrite(GREEN_LED_PIN, HIGH);
digitalWrite(YELLOW_LED_PIN, LOW);
digitalWrite(RED_LED_PIN, LOW);

// Initialize communication protocols
Wire.begin();           // Join I2C bus as a master
Serial.begin(9600);      // Initialize Serial Connection
Serial1.begin(BT_BAUD_RATE); // Initialize Bluetooth communication
SPI.begin();             // Initialize SPI protocol
rfid.PCD_Init();

// Initialize Servo
washingMachineServo.attach(WASH_MACHINE_PIN);
washingMachineServo.write(0); // Initial position, simulating OFF state

// Initialize RFID key
for (byte i = 0; i < 6; i++) {
    key.keyByte[i] = 0xFF;
}

// Log startup messages
logToBT("I2C Master Initialized.");
logToBT("Place your RFID card to start...");
}

void loop() {
    if (!isCardAuthenticated) {
        indicateSystemReady();
        waitForCard();
    } else {
        char key = keypad.getKey(); // Check for input from the keypad

        if (key) {
            processKeypadCommand(key); // Process the key pressed
        }

        // Check for input from Serial or Bluetooth
        if (Serial.available() > 0 || Serial1.available() > 0) {
            String command;
            if (Serial.available() > 0) {

```

```

        command = Serial.readStringUntil('\n');
    } else if (Serial1.available() > 0) {
        command = Serial1.readStringUntil('\n');
    }
    command.trim();
    processCommand(command);
}
}

// Indicator: System Ready
void indicateSystemReady() {
    digitalWrite(GREEN_LED_PIN, HIGH); // Green LED only On
    digitalWrite(YELLOW_LED_PIN, LOW);
    digitalWrite(RED_LED_PIN, LOW);
}

// Indicator: Washing State
void indicateWashingState() {
    digitalWrite(GREEN_LED_PIN, LOW);
    digitalWrite(YELLOW_LED_PIN, HIGH); // Yellow LED only On
    digitalWrite(RED_LED_PIN, LOW);
}

// Indicator: Emergency
void indicateEmergencyStop() {
    digitalWrite(GREEN_LED_PIN, LOW);
    digitalWrite(YELLOW_LED_PIN, LOW);
    digitalWrite(RED_LED_PIN, HIGH); // Red LED only ON

    delay(10000); // delay a bit

    indicateSystemReady(); // Turn on Green LED
}

// Verify Stop Button
void stopButtonISR() {
    static unsigned long lastInterruptTime = 0;
    unsigned long interruptTime = millis();

    // Debounce check (ignore if within 200ms of the last interrupt)
    if (interruptTime - lastInterruptTime > 200) {
        stopFlag = true;
        indicateEmergencyStop(); // Turn on red LED
        logToBT("Emergency stop activated."); // Set flag to stop the machine
    }
}

```

```

    }
    lastInterruptTime = interruptTime;
}

// Melody for Start
void playMelody(int melody[], int noteDurations[], int length) {
    for (int i = 0; i < length; i++) {
        int noteDuration = noteDurations[i];
        if (melody[i] == 0) {
            delay(noteDuration); // Pause for the duration if note is 0
        } else {
            playTone(melody[i], noteDuration);
        }
    }
}

// Tone for Start
void playTone(int frequency, int duration) {
    int period = 1000000 / frequency; // Calculate the period in microseconds
    int pulse = period / 2;           // Calculate the pulse width
    unsigned long endTime = millis() + duration;

    while (millis() < endTime) {
        digitalWrite(BUZZER_PIN, HIGH); // Turn on Buzzer
        delayMicroseconds(pulse);
        digitalWrite(BUZZER_PIN, LOW);  // Turn off Buzzer
        delayMicroseconds(pulse);
    }
}

// Melody for Stop
void playMelody2(int melody[], int noteDurations[], int length) {
    for (int i = 0; i < length; i++) {
        int noteDuration = noteDurations[i];
        if (melody[i] == 0) {
            delay(noteDuration); // Pause for the duration if note is 0
        } else {
            playTone2(melody[i], noteDuration);
        }
    }
}

// Tone for Stop
void playTone2(int frequency, int duration) {
    int period = 1000000 / frequency; // Calculate the period in microseconds

```

```

    int pulse = period / 2;           // Calculate the pulse width
    unsigned long endTime = millis() + duration;
    while (millis() < endTime) {
        digitalWrite(BUZZER_PIN_2, HIGH);    // Turn on Buzzer
        delayMicroseconds(pulse);
        digitalWrite(BUZZER_PIN_2, LOW);     // Turn off Buzzer
        delayMicroseconds(pulse);
    }
}

// log Messages to Serial and Bluetooth
void logToBT(String message) {
    Serial.println(message);    // Send to Serial Monitor
    Serial1.println(message);   // Send to Bluetooth
}

// Sending Message to Slave (Unused)
void sendToSlave(String message) {
    Wire.beginTransaction(SLAVE_ADDRESS);    // Start transmission to slave
    Wire.write(message.c_str());             // Send the message
    Wire.endTransmission();                  // End transmission
    Serial.println("Sent to slave: " + message);
}

// Check Token changes from slave (Unused)
int getTokensFromSlave() {
    // Get token amount from the slave via I2C
    String tokenAmount = "";
    Wire.requestFrom(8, 32);
    while (Wire.available()) {
        char received = Wire.read();
        tokenAmount += received;
    }
    return tokenAmount.toInt();
}

// Print Available Commands
void printCommands() {
    logToBT("Commands:");
    logToBT("1. Add tokens (ADD): ");
    logToBT("2. Check tokens (CHECK): ");
    logToBT("3. Start System (START): ");
    logToBT("4. Maintenance Mode (FIX): ");
}

```

```

// Process User Command from BT or Serial
void processCommand(String command) {
    // Checking for user input if it correct then assigned to different
    function
    if (command.startsWith("ADD")) {
        addTokens();
    } else if (command.equalsIgnoreCase("CHECK")) {
        checkTokens();
    } else if (command.equalsIgnoreCase("START")) {
        useMachine();
    } else if (command.equalsIgnoreCase("FIX")) {
        fix();
    } else {
        logToBT("Invalid command. Please try one of the following:");
        printCommands();
    }
}

// Process User Command from Keypad
void processKeypadCommand(char key) {
    if (key == 'A') { // A: Add tokens
        logToBT("Keypad input: A (Add Tokens)");
        logToBT("Enter the number of tokens to add:");
        int tokensToAdd = getNumericInputFromKeypad(); // Get numeric
        input from the keypad
        if (tokensToAdd > 0) {
            logToBT("Adding tokens: " + String(tokensToAdd)); // Adding token
            addTokensFromKeypad(tokensToAdd); // Update token
        } else {
            logToBT("Invalid input for tokens.");
        }

    } else if (key == 'B') { // B: Check
        tokens
        logToBT("Keypad input: B (Check Tokens)");
        checkTokens();

    } else if (key == 'C') { // C: Start the
        washing machine
        logToBT("Keypad input: C (Start Machine)");
        logToBT("Select washing temperature:");
        logToBT("1. Hot (3 tokens)");
        logToBT("2. Warm (2 tokens)");
        logToBT("3. Cold (1 token)");
    }
}

```

```

        int tempChoice = getNumericInputFromKeypad();           // Get numeric
input from the keypad
        if (tempChoice < 1 || tempChoice > 3) {
            logToBT("Invalid temperature choice.");
            return;
        }

        logToBT("Enter washing time in seconds (min: 10, increments of 10):");
// Adding more time with increment of 10 = 1 Token
        int washingTime = getNumericInputFromKeypad();
        if (washingTime < 10 || washingTime % 10 != 0) {
            logToBT("Invalid washing time.");
            return;
        }

        logToBT("Starting machine with temperature choice: " +
String(tempChoice) + " and time: " + String(washingTime));
        startMachineWithKeypad(tempChoice, washingTime);

    } else if (key == 'D') {                                     // D: Maintenance
mode
        logToBT("Keypad input: D (Fix)");
        fix();

    } else { // Invalid key
        logToBT("Invalid key pressed on keypad. Use A, B, C, or D."); //
Invalid
    }
}

// Process Numeric Input
int getNumericInputFromKeypad() {
    String input = "";           // Buffer to store numeric input
    char key;

    logToBT("Enter a number:");

    // Wait for numeric input from keypad
    while (true) {
        key = keypad.getKey();
        if (key) {
            if (isdigit(key)) {
                input += key;           // Append digit to input buffer
                logToBT("Entered: " + input); // Log Input from user to Serial
& BT

```

```

        } else if (key == '#') {           // '#' = For Entering Input
            break;
        } else {
            logToBT("Invalid key. Enter digits only, end with '#'.");
        }
    }
}

return input.toInt();                    // Convert the input string to an
integer: System Using Integer not String
}

// Read Card Serial
void waitForCard() {
    if (rfid.PICC_IsNewCardPresent() && rfid.PICC_ReadCardSerial()) { //Reading
Card UID
        String uid = "Card detected! UID: ";
        for (byte i = 0; i < rfid.uid.size; i++) {
            uid += String(rfid.uid.uidByte[i], HEX) + " ";
        }
        logToBT(uid);    // Log the UID

        isCardAuthenticated = true;
        logToBT("Card authenticated. You can now use the system.");
        printCommands();

        rfid.PICC_HaltA();
        rfid.PCD_StopCrypto1();
    }
}

// Authenticate card and store the token value
bool authenticateCard() {
    if (!rfid.PICC_IsNewCardPresent() || !rfid.PICC_ReadCardSerial()) {
        logToBT("No card detected.");
        return false;
    }

    byte blockAddr = 1;    // Block address where tokens are stored
    if (rfid.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, blockAddr, &key,
&(rfid.uid)) != MFRC522::STATUS_OK) {
        logToBT("Authentication failed.");
        rfid.PICC_HaltA();
        return false;
    }
}

```



```

        return true;
    }

    // Check user Input from BT Terminal & Serial Monitor
    int getInputFromUser() {
        String input = "";
        while (true) {
            // Check Serial input (Serial Monitor)
            if (Serial.available()) {
                input = Serial.readStringUntil('\n');
                input.trim();
                break;
            }

            // Check Serial1 input (Bluetooth terminal)
            if (Serial1.available()) {
                input = Serial1.readStringUntil('\n');
                input.trim();
                break;
            }
        }

        int value = input.toInt(); // Convert input to integer
        if (value > 0) {
            return value;          // Return valid input
        } else {
            logToBT("Invalid input. Please enter a number.");
            return getInputFromUser(); // Retry if input is invalid
        }
    }

    // Checking Token inside the card
    void checkTokens() {
        if (!authenticateCard()) return; // Check card authentication first

        byte blockAddr = 1;
        byte buffer[18];
        byte bufferSize = sizeof(buffer);
        if (rfid.MIFARE_Read(blockAddr, buffer, &bufferSize) != MFRC522::STATUS_OK)
        {
            logToBT("Reading failed.");
            rfid.PICC_HaltA();
            return;
        }
    }

```

```

    int tokens = buffer[0];    // Token inside buffer[0]
    logToBT("Tokens available: " + String(tokens));

    rfid.PICC_HaltA();
    rfid.PCD_StopCrypto1();
}

// Adding Token inside the card: From BT or Serial
void addTokens() {
    if (!authenticateCard()) return; // Authenticate the card first

    byte blockAddr = 1;
    byte buffer[18];
    byte bufferSize = sizeof(buffer);

    // Read the current token balance
    if (rfid.MIFARE_Read(blockAddr, buffer, &bufferSize) != MFRC522::STATUS_OK)
    {
        logToBT("Reading failed.");
        rfid.PICC_HaltA();
        return;
    }

    // Print Current token by checking inside Buffer[0]
    int currentTokens = buffer[0];
    logToBT("Current tokens: " + String(currentTokens));

    // Ask the user to input the token amount
    logToBT("Enter the number of tokens to add:");
    int tokensToAdd = getInputFromUser();

    // Token must exist in order to used
    if (tokensToAdd <= 0) {
        logToBT("Invalid token amount. Please enter a positive number.");
        return;
    }

    // Update the token balance
    int newTokenBalance = currentTokens + tokensToAdd;
    buffer[0] = newTokenBalance;

    // Adding updated token value to the card
    if (rfid.MIFARE_Write(blockAddr, buffer, 16) == MFRC522::STATUS_OK) {
        logToBT("Tokens successfully added. New balance: " +

```

```

String(newTokenBalance));
    } else {
        logToBT("Failed to update token count.");
    }

    rfid.PICC_HaltA();
    rfid.PCD_StopCrypto1();

    printCommands();
}

// Adding Token inside the card: From Keypad
void addTokensFromKeypad(int tokensToAdd) {
    if (!authenticateCard()) return;    // Authenticate the card first

    byte blockAddr = 1;
    byte buffer[18];
    byte bufferSize = sizeof(buffer);

    // Read the current token balance
    if (rfid.MIFARE_Read(blockAddr, buffer, &bufferSize) != MFRC522::STATUS_OK)
    {
        logToBT("Reading failed.");
        rfid.PICC_HaltA();
        return;
    }

    int currentTokens = buffer[0];
    int newTokenBalance = currentTokens + tokensToAdd;

    buffer[0] = newTokenBalance;

    if (rfid.MIFARE_Write(blockAddr, buffer, 16) == MFRC522::STATUS_OK) {
        logToBT("Tokens successfully added. New balance: " +
String(newTokenBalance));
    } else {
        logToBT("Failed to update token count.");
    }

    rfid.PICC_HaltA();
    rfid.PCD_StopCrypto1();

    printCommands();
}

```

```

// Maintenance: Check Temp & Water Level from Slave
void fix() {
    Wire.beginTransmission(8); // Start communication with slave at address 8
    Wire.write('F');           // Send the 'Fix' command to the slave
    Wire.endTransmission();

    delay(100);                // Wait for slave to prepare the data

    Wire.requestFrom(8, 4);    // Request 4 bytes (2 for temperature, 2 for
water level)
    if (Wire.available() == 4) {
        // Read temperature bytes
        int highByteTemp = Wire.read();
        int lowByteTemp = Wire.read();
        float temperature = (highByteTemp << 8 | lowByteTemp) / 100.0;

        // Read water level bytes
        int highByteWater = Wire.read();
        int lowByteWater = Wire.read();
        int waterLevel = (highByteWater << 8 | lowByteWater);

        // Display temperature and water level
        Serial.print("Temperature received from slave: ");
        Serial.print(temperature);
        Serial.println(" °C");

        Serial.print("Water level received from slave: ");
        Serial.println(waterLevel);

        logToBT("Current temperature: " + String(temperature) + " °C");
        logToBT("Current water level: " + String(waterLevel));

    } else {
        Serial.println("Error: Could not retrieve data from slave.");
    }

    printCommands();
}

// Main Function : From BT or Serial
void useMachine() {
    stopFlag = false;           // Reset the stop flag to ensure
smooth operation

    if (!authenticateCard()) return; // Authenticate the card. If

```

failed, exit.

```
    byte blockAddr = 1;                // Memory block address in RFID to
read/write data
    byte buffer[18];                    // Buffer to hold data read from
the card
    byte bufferSize = sizeof(buffer);   // Size of the buffer

    // Attempt to read data from the RFID card
    if (rfid.MIFARE_Read(blockAddr, buffer, &bufferSize) != MFRC522::STATUS_OK)
    {
        logToBT("Reading failed.");      // Log failure if unable to read
        rfid.PICC_HaltA();                // Halt RFID communication
        return;
    }

    indicateWashingState();              // Turn on Washing State LED

    int tokens = buffer[0];              // Retrieve the number of tokens
from the RFID card
    logToBT("Tokens available: " + String(tokens)); // Log available tokens

    // 🍷 Step 1: Ask user to choose the washing temperature
    logToBT("Select washing temperature:");
    logToBT("1. Hot (3 tokens)");
    logToBT("2. Warm (2 tokens)");
    logToBT("3. Cold (1 token)");

    int tempChoice = getInputFromUser(); // Get user input for temperature
choice
    int tempTokens = 0;                  // Tokens required for the selected
temperature

    // Process user's temperature choice
    switch (tempChoice) {
        case 1:
            logToBT("You selected: Hot");
            tempTokens = 3;                // Deduct 3 tokens for hot wash
            break;
        case 2:
            logToBT("You selected: Warm");
            tempTokens = 2;                // Deduct 2 tokens for warm wash
            break;
        case 3:
            logToBT("You selected: Cold");
```

```

        tempTokens = 1; // Deduct 1 token for cold wash
        break;
    default:
        logToBT("Invalid choice. Please try again.");
        return; // Exit if the choice is invalid
    }

    // Check if the user has enough tokens for the selected temperature
    if (tokens < tempTokens) {
        logToBT("Not enough tokens for the selected temperature. Required: " +
String(tempTokens) + ", Available: " + String(tokens));
        return;
    }

    // Deduct tokens for temperature selection
    tokens -= tempTokens;
    logToBT("Tokens deducted for temperature: " + String(tempTokens) + ".
Remaining tokens: " + String(tokens));

    // 🕒 Step 2: Ask user to enter washing time
    logToBT("Enter washing time in seconds (min: 10, increments of 10): ");
    int washingTime = getInputFromUser(); // Get user input for
washing time

    // Validate the washing time entered by the user
    if (washingTime < 10 || washingTime % 10 != 0) {
        logToBT("Invalid washing time. Please enter a multiple of 10
seconds.");
        return;
    }

    int requiredTokens = 1 + ((washingTime - 10) / 10); // Calculate tokens
needed for the time
    logToBT("Tokens required for time: " + String(requiredTokens));

    // Check if the user has enough tokens for the selected time
    if (tokens >= requiredTokens) {
        tokens -= requiredTokens; // Deduct tokens for
the time
        buffer[0] = tokens; // Update the buffer
with remaining tokens

        // Write updated tokens back to the RFID card
        if (rfid.MIFARE_Write(blockAddr, buffer, 16) == MFRC522::STATUS_OK) {
            logToBT("Tokens deducted. Remaining tokens: " + String(tokens));

```

```

        logToBT("Starting washing machine for " + String(washingTime) + "
seconds...");

        playMelody(startMelody, startNoteDurations, 11); // Play start
melody

        unsigned long startTime = millis();                // Record the
start time

        unsigned long remainingTime = washingTime * 1000; // Convert
washing time to milliseconds

        // Operate the washing machine for the specified time
        while (millis() - startTime < remainingTime) {
            if (stopFlag) {                                // Check if the
stop flag is triggered
                logToBT("Operation stopped by user.");
                break;
            }
            washingMachineServo.write(90);                  // Simulate
machine operation
            delay(500);
            washingMachineServo.write(0);
            delay(500);
        }

        if (!stopFlag) {                                    // If not stopped
prematurely
            indicateSystemReady();                          // Indicate the
system is ready

            logToBT("Washing complete. Thank you!");
            playMelody2(stopMelody, startNoteDurations, 5); // Play stop
melody
        }

        printCommands();                                    // Print next
available commands
    } else {
        logToBT("Failed to update token count."); // Log failure if writing
to card fails
    }
    } else {
        logToBT("Not enough tokens. Required: " + String(requiredTokens) + ",
Available: " + String(tokens));
    }
}

```

```

    rfid.PICC_HaltA(); // Halt RFID
communication
    rfid.PCD_StopCrypto1(); // Stop
cryptographic communication with the card
}

// Main Function : From Keypad
void startMachineWithKeypad(int tempChoice, int washingTime) {
    stopFlag = false; // Reset the stop flag

    if (!authenticateCard()) return; // Authenticate the card. If
failed, exit.

    byte blockAddr = 1; // Memory block address in RFID to
read/write data
    byte buffer[18]; // Buffer to hold data read from
the card
    byte bufferSize = sizeof(buffer); // Size of the buffer

    // Attempt to read data from the RFID card
    if (rfid.MIFARE_Read(blockAddr, buffer, &bufferSize) != MFRC522::STATUS_OK)
    {
        logToBT("Reading failed."); // Log failure if unable to read
rfid.PICC_HaltA(); // Halt RFID communication
        return;
    }

    int tokens = buffer[0]; // Retrieve the number of tokens
from the RFID card
    int tempTokens = (tempChoice == 1) ? 3 : (tempChoice == 2) ? 2 : 1; //
Tokens required for temperature choice

    // Check if the user has enough tokens for the selected temperature
    if (tokens < tempTokens) {
        logToBT("Not enough tokens for the selected temperature.");
        return;
    }

    int requiredTokens = 1 + ((washingTime - 10) / 10); //
Calculate tokens needed for the time
    if (tokens < tempTokens + requiredTokens) {
        logToBT("Not enough tokens for the selected time.");
        return;
    }
}

```



```

    tokens -= (tempTokens + requiredTokens); //
Deduct tokens for temperature and time
    buffer[0] = tokens; //
Update the buffer with remaining tokens

    // Write updated tokens back to the RFID card
    if (rfid.MIFARE_Write(blockAddr, buffer, 16) == MFRC522::STATUS_OK) {
        logToBT("Tokens deducted. Remaining tokens: " + String(tokens));
        logToBT("Starting washing machine...");
        indicateWashingState(); //
Turn On Washing State LED

        playMelody(startMelody, startNoteDurations, 11); //
Play start melody

        unsigned long startTime = millis(); //
Record the start time
        unsigned long remainingTime = washingTime * 1000; //
Convert washing time to milliseconds

        // Operate the washing machine for the specified time
        while (millis() - startTime < remainingTime) {
            if (stopFlag) { //
Check if the stop flag is triggered
                logToBT("Operation stopped by user.");
                break;
            }
            washingMachineServo.write(90); //
Simulate machine operation
            delay(500);
            washingMachineServo.write(0);
            delay(500);
        }

        if (!stopFlag) { // If
not stopped prematurely
            indicateSystemReady(); //
Indicate the system is ready
            logToBT("Washing complete. Thank you!");
            playMelody2(stopMelody, startNoteDurations, 5); // Play
stop melody
        }
    } else {
        logToBT("Failed to update token count."); // Log
failure if writing to card fails

```

```

    }

    rfid.PICC_HaltA();                                // Halt
RFID communication
    rfid.PCD_StopCrypto1();                            // Stop
cryptographic communication with the card
}

```

Our implements of a washing machine is a control system using an Arduino Mega 2560 microcontroller as Master. The system uses RFID cards for user authentication and operates on a token-based payment system. Users can interact with the machine through either a 4x4 keypad, Bluetooth communication, or Serial Monitor. The system includes features for adding tokens to RFID cards, checking token balances, and starting wash cycles with different temperature settings (hot, warm, or cold) that consume varying amounts of tokens.

The washing machine's operation is simulated using a servo motor, and the system provides visual feedback through three LEDs: green for ready state, yellow for washing state, and red for emergency stop. Audio feedback is implemented through two buzzers that play different melodies for start and stop operations. The system includes an emergency stop button with debounce protection implemented through an interrupt service routine.

The code communicates with a slave Arduino device (UNO R3) via I²C protocol to monitor temperature and water levels for maintenance purposes. All operations are logged both to the Serial Monitor and via Bluetooth for remote monitoring. The washing cycle duration is configurable in 10-second increments, with each additional increment consuming one token. Temperature selections require different token amounts: hot wash uses 3 tokens, warm wash uses 2 tokens, and cold wash uses 1 token. The system stores token balances directly on the RFID cards, updating them after each operation.

The structure of the code is typical setup-loop pattern, with the main loop constantly checking for user input through multiple interfaces. The code includes comprehensive error handling for card authentication, token validation, and user input verification, ensuring robust operation. Helper functions handle specific tasks like playing melodies, processing commands, and managing the washing cycle, making the code modular and maintainable.

Slave (System 1):

```
#include <Wire.h>

const int tempPin = A0;           // LM35 sensor pin
const int waterLevelPin = A1;     // Water level sensor pin
const int slaveAddress = 8;       // I2C address of this slave

void setup() {
    Wire.begin(slaveAddress);      // Join I2C bus with slave address
    Wire.onRequest(requestEvent);  // Register function to send data
    pinMode(tempPin, INPUT);
    pinMode(waterLevelPin, INPUT);
    Serial.begin(9600);            // For debugging
}

// Function to read temperature from LM35
float readTemperature() {
    int sensorValue = analogRead(tempPin);
    float voltage = sensorValue * (5.0 / 1023.0); // Convert to voltage
    float temperature = voltage * 100;           // LM35 outputs 10 mV/°C
    return temperature;
}

// Function to read water level sensor value
int readWaterLevel() {
    int waterLevelValue = analogRead(waterLevelPin);
    return waterLevelValue; // Return raw analog value
                             // (0-1023)
}

// Function to send data to the master
void requestEvent() {
    float temp = readTemperature();
    int tempInt = static_cast<int>(temp * 100); // Temperature scaled by 100
    int waterLevel = readWaterLevel();          // Read water level sensor

    // Send temperature as two bytes
    Wire.write((tempInt >> 8) & 0xFF);          // High byte of temperature
    Wire.write(tempInt & 0xFF);                 // Low byte of temperature

    // Send water level as two bytes
    Wire.write((waterLevel >> 8) & 0xFF);        // High byte of water level
    Wire.write(waterLevel & 0xFF);              // Low byte of water level
}
```

```

    Serial.print("Sent temperature to master: ");
    Serial.println(temp);
    Serial.print("Sent water level to master: ");
    Serial.println(waterLevel);
}

void loop() {
}

```

Slave code is an I2C slave device (UNO R3) that monitors environmental conditions through temperature and water level sensors. The system reads temperature data from an LM35 sensor connected to analog pin A0, converting raw values to Celsius through a voltage transformation process. Simultaneously, it captures water level readings from a sensor on pin A1.

When the master device requests data through I²C (slave address 8), the program packages both temperature and water level readings into four bytes - two bytes for each measurement. Temperature data is scaled by 100 to preserve decimal precision before transmission. The system employs an event-driven architecture, responding only to I²C requests rather than continuous polling.

System 2:

Library and setup (System2) :

```

#include <Pixy.h>           // For the Pixy camera module to detect objects
                             and colors
#include <LiquidCrystal_I2C.h> // For the I2C LCD display
#include <NewPing.h>         // For the ultrasonic distance sensor
#include <SPI.h>             // Required for Pixy camera communication

```

Pixy by Charmslabs

LiquidCrystal i2C by Frank De Brabander

Newping by Tim Eckel

For slave, we are using UNO R3. `pixy.h` is used to communicate with the camera and read the camera signature, `liquidcrystal_I2C.h` is used to show the information on the LCD 16 by 2, `newping.h` is used to control the ultrasonic sensor and `spi.h` is used to establish connection of SPI protocol.

```
// Define pins and constants for the system
#define TRIGGER_PIN  22      // Ultrasonic sensor trigger pin
#define ECHO_PIN     23      // Ultrasonic sensor echo pin
#define LED_PIN      13      // LED indicator for object presence
                             // detection
#define WHITE_LED_PIN 12      // LED indicator specifically for
                             // white shirt detection
#define MAX_DISTANCE 200     // Maximum detection distance in
                             // centimeters
#define MIN_BLOCK_SIZE 20    // Minimum size of detected object to
                             // be considered valid
#define SCAN_ATTEMPTS 3      // Number of attempts to scan for
                             // white shirt

// Initialize hardware objects
Pixy pixy;                  // Create Pixy camera object
LiquidCrystal_I2C lcd(0x27, 16, 2); // Initialize LCD with address
0x27, 16 columns, 2 rows
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // Initialize
ultrasonic sensor

// System state variables
bool clothDetected = false; // Tracks if any object is currently
                             // detected
int whiteShirtSignature = 1; // Pixy camera signature ID for white
                             // shirt
unsigned long lastDetectionTime = 0; // Timestamp of last detection
const unsigned long DETECTION_TIMEOUT = 1000; // Time before
allowing new detection (debouncing)
bool isWhiteShirtDetected = false; // Tracks if a white shirt is
specifically detected

void setup() {
```

```

Serial.begin(9600);           // Initialize serial communication for
debugging

// Configure LED pins
pinMode(LED_PIN, OUTPUT);
pinMode(WHITE_LED_PIN, OUTPUT);
digitalWrite(LED_PIN, LOW);
digitalWrite(WHITE_LED_PIN, LOW);

// Initialize LCD display
lcd.init();
lcd.backlight();
lcd.clear();
lcd.print("System Ready");

pixy.init();                  // Initialize Pixy camera
Serial.println("Initialization complete");
}

void loop() {
    // Get distance reading from ultrasonic sensor
    int distance = sonar.ping_cm();

    // Check if object is within detection range (0-20cm)
    if (distance > 0 && distance < 20) {
        // Only process if no recent detection or timeout has passed
        (debouncing)
        if (!clothDetected || (millis() - lastDetectionTime >
DETECTION_TIMEOUT)) {
            clothDetected = true;
            lastDetectionTime = millis();
            digitalWrite(LED_PIN, HIGH); // Turn on presence detection LED

            // Scan for white shirt and update detection status
            isWhiteShirtDetected = scanForWhiteShirt();

            // Control white shirt indicator LED based on detection
            digitalWrite(WHITE_LED_PIN, isWhiteShirtDetected ? HIGH : LOW);

```

```

    // Update LCD display with detection results
    lcd.clear();
    if (isWhiteShirtDetected) {
        lcd.print("White shirt");
        lcd.setCursor(0, 1);
        lcd.print("detected!");
    } else {
        lcd.print("No white shirt");
        lcd.setCursor(0, 1);
        lcd.print("found");
    }
}
} else {
    // Reset system state when no object is detected
    clothDetected = false;
    isWhiteShirtDetected = false;
    digitalWrite(LED_PIN, LOW);
    digitalWrite(WHITE_LED_PIN, LOW);
    lcd.clear();
    lcd.print("System Ready");
}

delay(50); // Small delay to prevent excessive processing
}

// Function to detect white shirt using Pixy camera
bool scanForWhiteShirt() {
    int successfulScans = 0; // Counter for successful white shirt
    detections

    // Perform multiple scan attempts for reliable detection
    for (int attempt = 0; attempt < SCAN_ATTEMPTS; attempt++) {
        uint16_t blocks = pixy.getBlocks(); // Get detected objects from
        Pixy

        if (blocks) {
            // Check each detected block
            for (uint16_t i = 0; i < blocks; i++) {
                // Verify if block matches white shirt criteria (signature

```

```

and size)
    if (pixy.blocks[i].signature == whiteShirtSignature &&
        pixy.blocks[i].width > MIN_BLOCK_SIZE &&
        pixy.blocks[i].height > MIN_BLOCK_SIZE) {
        successfulScans++;

        // Log detection details for debugging
        Serial.print("Block detected - Width: ");
        Serial.print(pixy.blocks[i].width);
        Serial.print(" Height: ");
        Serial.println(pixy.blocks[i].height);

        break; // Exit current scan if white shirt is detected
    }
}
delay(50); // Brief delay between scan attempts
}

// Determine final detection result (majority of scans must be
successful)
bool detected = successfulScans > SCAN_ATTEMPTS/2;

// Log final detection results
Serial.print("Detection result: ");
Serial.println(detected ? "White shirt detected" : "No white shirt
found");
Serial.print("Successful scans: ");
Serial.println(successfulScans);

return detected;
}

```

System2 is a smart detection system that identifies white shirts using multiple sensors. The system uses an ultrasonic sensor to detect when objects are within 20 cm range. When something is detected, it activates a Pixy camera that specifically looks for white shirts through a series of three quick scans. To ensure accuracy, the system only confirms a white shirt detection if the majority of these scans are successful and the detected object meets minimum size requirements.

The system provides user feedback through two LEDs (one for object detection, another for white shirt confirmation) and an LCD screen that displays the current status and detection results. To prevent rapid, flickering readings, the system waits one second between detection cycles. The code includes debug logging through serial communication to help monitor the system's performance and troubleshoot any issues.

DISCUSSION

The automated washing machine system detailed in this project showcases an impressive integration of advanced technologies, transforming a common household appliance into a smart, user-focused solution. By employing RFID technology as a token-based system, users can securely and effortlessly authenticate their interactions with the machine. This innovative approach enables personalized washing settings and enhances the overall user experience. The system incorporates components such as Pixy Camera, servo motors, buzzers, and a master-slave Arduino microcontroller configuration, demonstrating an effective and cohesive design. The master Arduino oversees system operations, while the slave units handle tasks like RFID data processing, sensor monitoring, and actuator control, ensuring efficient communication and coordination. This emphasis on leveraging embedded systems and smart technologies in household appliances aligns with the growing trend of home automation, highlighting the project's technical sophistication and its potential to contribute to the evolving landscape of user-centric smart home solutions.

User Interface:

The user interface component plays a vital role in enhancing the interaction mechanisms of the automated washing machine system, combining visual, auditory, and tactile feedback for a seamless user experience. Libraries such as Wire, LiquidCrystal_I2C, and MFRC522 enable the integration of LCD displays and RFID functionality, with RFID used for secure user authentication via personalized RFID card interactions. The LCD displays provide a user-friendly interface, dynamically updating to guide users through settings and configurations, while keypad buttons labeled A, B, and C handle tactile inputs, supported by additional input methods like Serial Monitor and BT Terminal. A buzzer provides auditory feedback, signaling key events such as the start and completion of the washing process. RFID authentication occurs

at the beginning, allowing users to proceed with the process upon successful authentication or prompting them to play a game for a free wash if unauthorized access is detected. The system's ability to manage settings, respond to button presses, and execute washing cycles effectively showcases a thoughtful and interactive user interface design, aligning with the goals of creating intuitive smart home appliances.

Process:

The system leverages a servo motor, ultrasonic sensor, and water level sensor to efficiently orchestrate the washing machine's operations. The ultrasonic sensor is used to detect the presence of clothes inside the washing machine and to identify the color of the fabric. If clothes are detected, and the color is identified, the system automatically selects and utilizes washing products suitable for preserving the color integrity of the fabric, preventing fading. The servo motor plays a crucial role in indicating that the washing machine is operational once the system is activated, serving as a visual confirmation for the user. The inclusion of a water level sensor ensures that the washing machine starts only when the water reaches the required level. If the water level is insufficient, the system prevents the process from starting, demonstrating a realistic and practical approach to addressing situations where the washing machine cannot operate due to low water levels. This feature not only safeguards the functionality of the washing machine but also ensures the user is updated and notified of any issues. By combining these components, the system effectively integrates multiple layers of automation and safety. The ultrasonic sensor and water level sensor work in tandem to guarantee proper operation, while the servo motor provides physical feedback, enhancing user interaction. These elements collectively demonstrate an innovative and user-centric approach to automating washing machine processes, aligning with the objectives of creating a smart and efficient home appliance.

In conclusion, this automated washing machine system successfully integrates advanced technologies such as RFID, Pixy Camera, ultrasonic sensors, servo motors, and a master-slave Arduino configuration to create a smart, user-centric solution. This project demonstrates the potential of embedding smart technologies in household appliances, paving the way for future innovations in home automation and user-friendly solutions.

CONCLUSION

The project successfully achieved its primary objectives by developing an intelligent, secure, and user-friendly washing machine prototype. This innovative system integrates computer vision technology through the Pixy camera, enabling smart garment detection and classification—an advancement that highlights the potential for automation in laundry systems. The inclusion of multiple sensors, communication protocols, and a user-centric interface reflects a comprehensive understanding of embedded systems design and their practical application.

The system's modular architecture and advanced features, such as RFID-based security, token-based payment mechanisms, real-time monitoring, and intelligent garment detection, combine to create a robust and versatile solution. These features demonstrate its adaptability for real-world applications in shared laundry facilities, dormitories, and commercial laundromats, addressing modern challenges with efficiency and reliability. Moreover, the design emphasizes user experience and intelligent automation, showcasing the potential of embedded systems in enhancing everyday tasks.

This prototype serves as a proof of concept and a stepping stone for further development in smart appliance technology. It paves the way for the evolution of automated laundry systems, offering intelligent garment detection and handling capabilities that can be refined and expanded upon in future iterations. The project not only demonstrates the successful implementation of advanced technologies but also sets a benchmark for innovation in the realm of intelligent home and commercial appliances.

RECOMMENDATION

The smart washing machine prototype can be further enhanced through advanced hardware and software upgrades that will enhance functionality and user experience. The use of machine learning algorithms in garment detection could allow the identification of fabric types, identification of stains, and automatic selection of the best washing programs. This can further be extended to include a new and improved imaging system that includes several Pixy cameras or higher-resolution sensors for the detailed analysis of garments from multiple angles. Hardware improvement would also include several peripherals, such as fabric tension sensors and weight detection mechanisms for overload prevention and optimization of washing performance.

Another modern expansion to methods of paying, including mobile, cryptocurrency, and the integration of campus card systems, would make it possible to provide the system for more users. The value addition in developing a mobile application includes offering real-time monitoring, notifications, predictive maintenance alerts, tracking of machine availability, and historical usage analytics to improve user convenience. From the infrastructure point of view, a full-fledged cloud-based backend would provide the possibility to manage multiple washing machines from one place, predict maintenance, balance the load, and optimize the system based on real-time data.

IoT integration would allow connecting the system with smart home ecosystems for automated operation, considering user preferences and daily routines. Additional sensors would enhance the system's sustainability for water quality, humidity, and energy consumption environmental monitoring while providing value data to the users and facility managers. Advanced diagnostic features and error detection enable remote troubleshooting, reducing maintenance costs. Introduction of blockchain for security enhancement in payment processing and user authentication. These enhancements would transform the prototype into a scalable,

efficient, and intelligent solution for shared laundry facilities, dormitories, and commercial settings.

ACKNOWLEDGEMENTS

A special thanks goes out to Dr. Wahyu Sediono and Dr. Zulkifli Bin Zainal Abidin, my teaching assistant, and my peers for their invaluable help and support in finishing this report. Their advice, feedback, and experience have greatly influenced the level of quality and understanding of this work. Their time, patience, and commitment to supporting my academic success are greatly appreciated.



STUDENT'S DECLARATION


Certificate of Originality and Authenticity

This is to certify that we are **responsible** for the work submitted in this report, that the original work is our own except as specified in the references and acknowledgement, and that the original work contained herein have not been untaken or done by unspecified sources or persons.


We hereby certify that this report has **not been done by only one individual** and **all of us have contributed to the report**. The length of contribution to the reports by each individual is noted within this certificate.

We also hereby certify that we have **read** and **understand** the content of the total report and no further improvement on the reports is needed from any of the individual's contributors to the report.

We therefore, agreed unanimously that this report shall be submitted for **marking** and this **final printed report** has been **verified by us**.

Signature: 
Name: IRDINA NABIHAH BINTI MOHD NAZRI
Matric Number: 2214772


Read ☒
Understand ☒
Agree ☒

Signature: 
Name: KHALISAH AMANI BINTI YAHYA AZMI
Matric Number: 2218184

Read ☒
Understand ☒
Agree ☒

Signature: 
Name: LUQMAN AZFAR BIN AZMI
Matric Number: 2219857

Read ☒
Understand ☒
Agree ☒

Signature: 
Name: MOHAMAD NASRI BIN MOHAMAD NAZRI
Matric Number: 2219879

Read ☒
Understand ☒
Agree ☒

Signature: 
Name: MUHAMAS NURHAKIMIE THAQIF BIN ABDULLAH
Matric Number: 2213217

Read ☒
Understand ☒
Agree ☒

