Powder Dispenser
Final Project Report

Group 26
Members: Sara Crogh, Sareena Sandhu, Navjot Singh, and Daniela Guerra

Department of Mechanical Engineering,
University of California at Berkeley
12/19/2024

Opportunity
Preparing powdered supplements efficiently is a growing challenge for users, including fitness enthusiasts, chefs and the elderly. Manual measurement is time-consuming and error-prone, leading to inconsistencies in portion sizes. This project addresses this issue by automating the powder dispensing process, ensuring consistent, accurate portions for users.

High-Level Strategy
Our dispenser consists of three subsystems: the weight selection panel, the linear dispensing auger, and the load cell based weighing mechanism. Each subsystem is connected to an ESP32 microcontroller. The process begins with the user selecting their desired serving size (in grams) by rotating the potentiometer. That value is displayed on an electronic screen. This value is then confirmed by the user via the press of a button. The user then presses the button a second time to confirm that the scale has been zeroed. Upon this confirmation, the ESP32 sends a signal to start the motor (Geared DC Motor, 12 V).

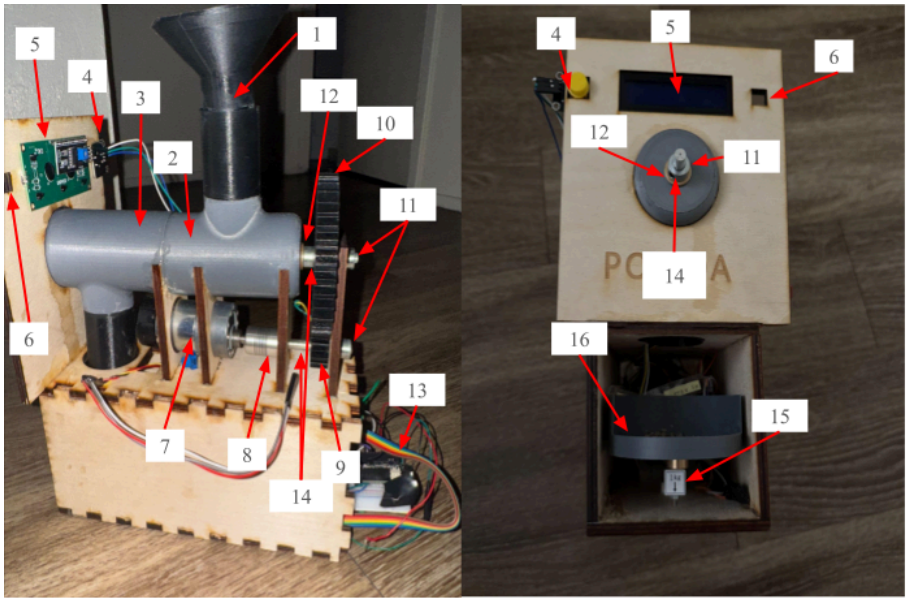
The motor is connected to two gears with a 4:1 ratio, which drive an auger drill. The auger drill precisely pushes the powder forward at a controlled feed rate. To maintain accurate dispensing, we utilized proportional-integral (PI) control to regulate the motor's angular speed. This ensures a consistent dispensing rate. The powder is dispensed into a cup positioned on a load cell. As the load cell reading approaches the target weight, the ESP32 responds by slowing the motor and auger speed to enhance accuracy.

In addition to the weight selection panel, linear dispensing auger system, and load cell system, we initially planned to create a fully automated setup capable of dispensing multiple powders, pouring water, and mixing with a magnetic whisk. However, we chose to focus on a single-powder system to ensure it worked reliably before scaling to a larger system.

One of the key challenges we faced was calibrating the system for different powder densities, which significantly impacted the consistency and accuracy of the dispensing process. Powders with varying flow properties required different settings, and switching between them risked cross-contamination. To address these challenges, we decided to refine the system for a single powder type, allowing us to focus on precision and reliability.

By limiting the scope, we ensured the system could consistently deliver accurate amounts without contamination, a critical factor for the user experience. Future work will explore solutions to calibrate for varying powder densities and incorporate multiple compartments for different powders while preventing cross-contamination. Expanding the system to handle these complexities without sacrificing precision and ease of use remains a key goal.

Integrated Physical Device



1) Hopper
2) Transport Tube
3) Auger Drill (Hidden)
4) Button
5) LED Display
6) Potentiometer
7) Motor
8) Shaft Coupler
9) Driving Gear
10) Driven Gear
11) Shaft Collar
12) Bushing
13) Electronics
14) Shafts
15) Strain Gauge
16) Cup Platform

Function-critical Decisions

Key design choices included:

• Using a stepper motor for precise control.

• Designing a modular hopper for easy powder replacement.

Our powder dispenser required a combination of precise mechanical control and reliable electronic feedback to ensure accuracy and ease of use. Key design decisions included:

1. Using a Stepper Motor for Precise Control
    - The stepper motor was selected to achieve the precise angular movement necessary for accurate powder dispensing. It ensures consistent increments of motion, making it ideal for the auger mechanism.
    - Calculation:
        - Calculated to handle 1kg of powder per compartment.
2. Designing a Modular Hopper for Easy Powder Replacement
    - A modular design simplifies maintenance and switching between powders. The hopper connects securely to the dispensing mechanism, minimizing cross-contamination.
    - Calculation:
        - Ensured with 1mm hopper opening precision.
3. Load Cell Integration for Dynamic Weight Feedback
    - The load cell provides real-time weight measurements to ensure precise dispensing. A 500g load cell was used, calibrated for accuracy within ±1 g.
    - Specs:
        - Sensitivity = 0.7 +/- 0.05 mV/V
        - The HX711 amplifier supports a resolution better than 0.1 mV, which aligns with this sensitivity requirement.

4. Gear Ratio Design for Motor-Auger Interface
    - A 4:1 gear ratio, though not necessary, was chosen to increase the encoder resolution from 22.5° to 5.625°. This allowed for better control and precision. The other byproduct of this gear ratio was an increase in torque which makes dispensing smoother for denser or frictive powders.
    - Calculation:
        - Motor Speed = 200 RPM
        - Shaft Speed = 50 RPM
            - Shaft Speed = ¼ Motor Speed
        - Auger Feed Rate = 5 g/rev.
        - Dispensing Rate = 250 g/min.
            - Dispensing Rate = Shaft Speed ✕ Auger Feed Rate
        - This matches the output speed of the geared system.
5. Auger Design for Consistent Flow
    - The auger's helical structure was designed to ensure consistent powder flow.
    - Calculation:
        - Helix angle = 30°, ensuring smooth powder flow without clogging.
        - Diameter and pitch were chosen to provide the required feed rate.

Circuit and State Transition Diagram

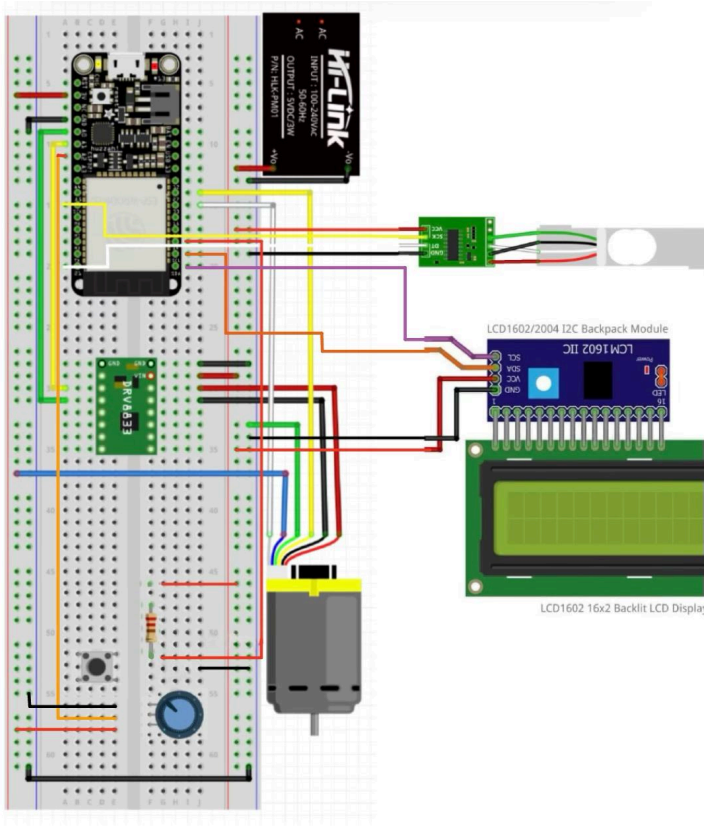


Figure 1 illustrates the electrical connections.

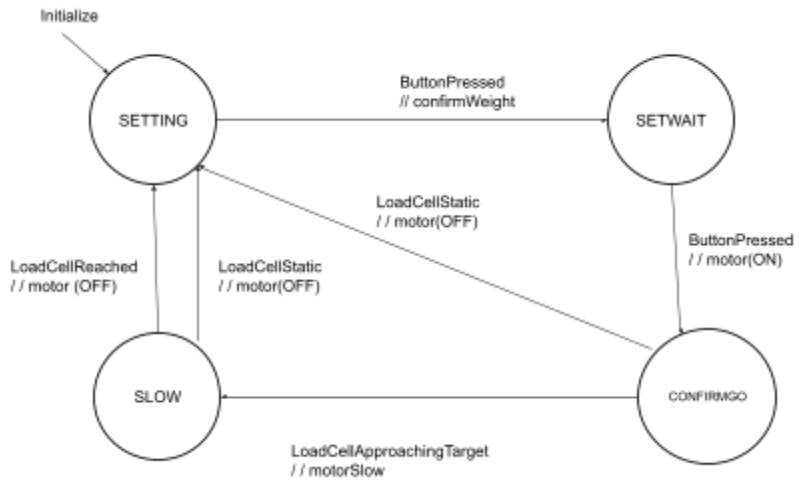


Figure 2 depicts the state transit logic.

1. SETTING: This is the initial state where the user selects the desired weight using a potentiometer. The chosen weight is displayed, and the transition to SETWAIT occurs when the button is pressed (ButtonPressed // confirmWeight).
2. SETWAIT: The system waits for the user to confirm that the scale is zeroed. Upon confirmation (ButtonPressed // motor(ON)), the system transitions to CONFIRMGO.
3. CONFIRMGO: The motor starts dispensing powder. If the load cell reading remains static for 5 second periods (LoadCellStatic // motor(OFF)), the motor stops to indicate a potential issue (e.g., no powder). As the target weight is approached (LoadCellApproachingTarget // motorSlow), the motor slows for precision. Once the desired weight is reached (LoadCellReached // motor(OFF)), the system transitions to SETTING.
4. SLOW: This intermediate state ensures fine-tuned dispensing as the load cell approaches the target weight, before ultimately transitioning back to SETTING.

Reflection

This project underscored the importance of iterative design, time management, and early subsystem integration. While our modular approach was effective, dedicating more time to testing and integration would have improved overall performance. Collaborating with peers and leveraging resources like the Machine Shop staff were critical to our success. Future teams should focus on early integration and realistic design goals to optimize outcomes. Expanding the project with additional subsystems and refining the existing design could lead to a fully automated, elegant solution.

Appendix

1. Bill of Materials

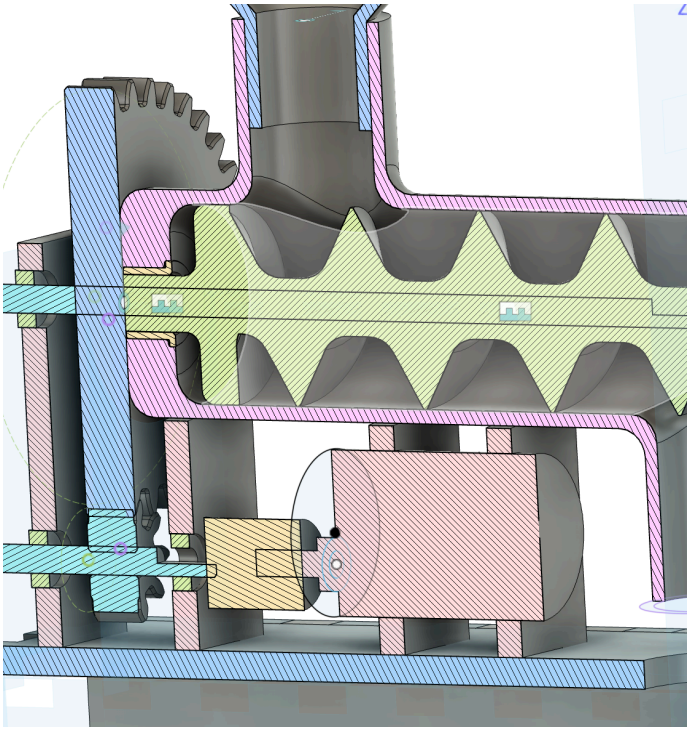
Part Name	Description	Vendor	Quantity	Link	Cost ($USD)	Status
Motor	12V Gear Motor w/Encoder, model No.GB37Y3530-12V-251R	dfrobot	1	https://www.dfrobot.com/product-634.html	0	Pre-owned
ESP32	Adafruit HUZZAH32 ESP32 Feather	Adafruit	1		0	Pre-owned
Digital I/O Button	2234-PUSH-04-ND	Digikey	1	https://www.digikey.com/en/products/detail/osepp-electronics-ltd/PUSH-04/11198597	0	In Progress
Potentiometer	Potentiometer	--	1	0	0	Pre-owned
Shaft	Straight	Amazon	3	https://www	9	Purchased

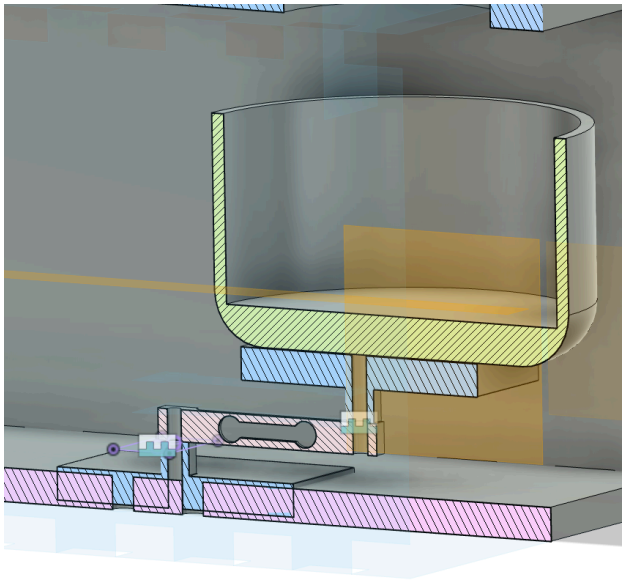
	Rotary Shafts			.amazon.com/dp/B0BNQ5BLP1?ref=ppx_yo2ov_dt_b_fed_asin_title		
Load Cells + Amplifier	4pcs Load Cell 50kg Half Bridge Strain Gauge Human Body Digital Scale Weight Sensor + 1pc HX711 Amplifier AD Module for Arduino + a larger loadcell for the larger readings (1kg) so 2 loadcells	Amazon	2	https://www.amazon.com/dp/B0CRDBJ8L7?ref=ppx_yo2ov_dt_b_fed_asin_title + https://www.amazon.com/dp/B07NRTJCFD?ref=ppx_yo2ov_dt_b_fed_asin_title	27	Purchased
Motor Driver	Maker Drive: Simplifying H-Bridge Motor Driver for Beginner	Cytron	1	NA	0	Pre-owned
Couplers	Flexible Shaft Coupling	Misumi	1	N/A	0	Pre-owned
Washers	Non-metal washers	Misumi	20	N/A	0	Pre-owned

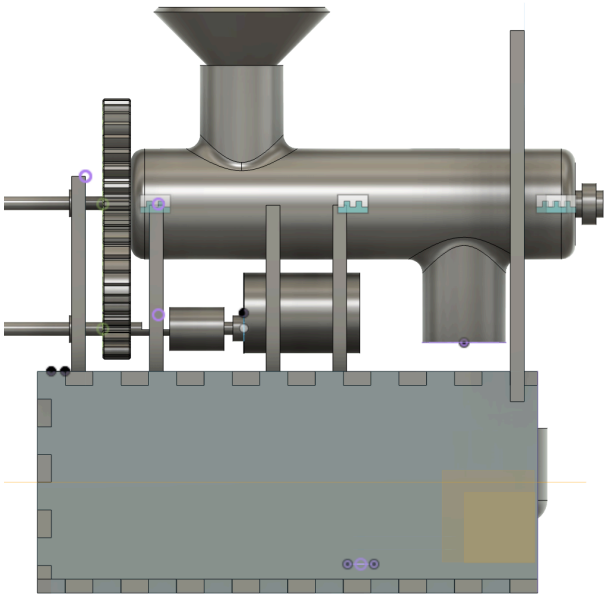
Shaft Collars	Clamping Shaft Collars	Misumi	4	N/A	0	Pre-owned
Bearings	Small Deep Groove Ball Bearing	Amazon	4	https://www.amazon.com/dp/B0CCNDW9JR?ref=ppx_yo2ov_dt_b_fed_asin_title	8	Purchased
Bolts	Fully Threaded Bolts & Studs M3	Amazon	10	https://www.amazon.com/dp/B0CSW91TXQ?ref=ppx_yo2ov_dt_b_fed_asin_title	15	Purchased
Wood	Hardwood Plank	Amazon	1	https://www.amazon.com/dp/B07F1WGHQW?ref=ppx_yo2ov_dt_b_fed_asin_title&th=1	44	Purchased
Bushing	Bushings	Amazon	4	https://www.amazon.com/dp/B0D6R8G11N?ref=ppx_yo2ov_dt_b_fed_asin_title	18	Purchased
Adhesive	Epoxy & Super Glue	Jacobs	10	https://store.jacobshall.org/collections/types?q=Adhesive	12	Purchased
Display	3pcs I2C	Amazon	3	https://www	14	Purchased

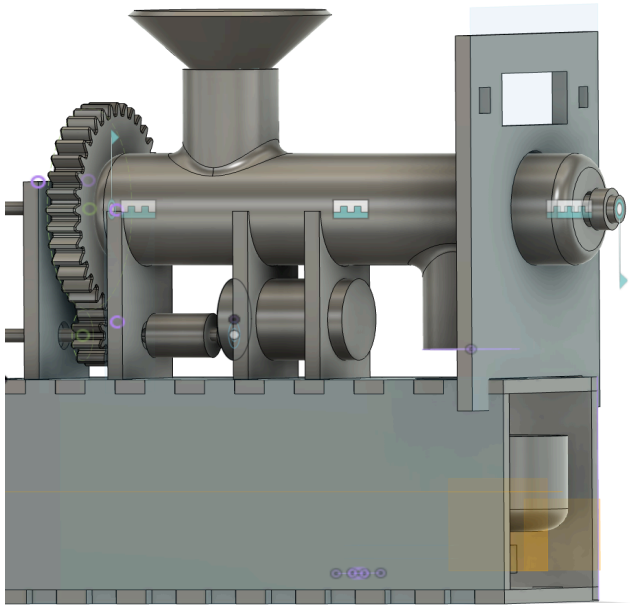
	IIC 1602 LCD Display Module 16x02 LCD Screen Module for Arduino Raspberry Pi			.amazon.com/gp/product/B0BWTFN9WF/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&th=1		

2. CAD Images









.

3. Code Include annotated code screenshots for key functionalities.

```
#include <Arduino.h>
#include "soc/rtc.h"
#include "HX711.h"
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Load cell and LCD setup
HX711 scale;
LiquidCrystal_I2C lcd(0x27, 16, 2); // Initialize a 16x2 LCD using I2C

// Pin assignments
#define BTN 14 // Button pin
#define LCOUT 21 // Load cell data pin
#define LCCLOCK 4 // Load cell clock pin
#define POT 34 // Potentiometer pin
#define LCD 36 // LCD data pin
#define LCDCLOCK 32 // LCD clock pin

// Motor pin assignments
const int motorPin1 = 26;
const int motorPin2 = 25;
```

```
const int pwmPin = 27;

// Constants
#define DEBOUNCE_DELAY 50 // Debounce time for button
#define PWM_CHANNEL 0
#define PWM_FREQ 5000
#define PWM_RESOLUTION 8
#define MAX_WEIGHT 100 // Maximum allowable weight

// State definitions
#define SETTING 0
#define SETWAIT 1
#define CONFIRMGO 2
#define REACHED 3

// Variables
volatile bool buttonIsPressed = false;
unsigned long lastPressTime = 0;
int state = SETTING;
int potReading = 0;
int desiredWeight = 0;
long currentWeight = 0;
int threshold = 0;
bool messageDisplayed = false;
unsigned long messageDisplayTime = 0;
const unsigned long messageDisplayDuration = 2000;
const unsigned long resetDelay = 7000;

// Function declarations
void stopMotor();
void checkPot();
void runMotorUntilWeight(int speed, float currentWeight, int threshold);
bool CheckForButtonPress();

// Interrupt Service Routine for button press
void IRAM_ATTR isr() {
    unsigned long currentTime = millis();
    if (currentTime - lastPressTime > DEBOUNCE_DELAY) {
        buttonIsPressed = true;
        lastPressTime = currentTime;
```

```
    }
}

void setup() {
    Serial.begin(115200); // Initialize serial communication

    // Initialize LCD
    Wire.begin(LCD, LCDCLOCK);
    lcd.begin(16, 2);
    lcd.backlight();

    // Configure pins
    pinMode(motorPin1, OUTPUT);
    pinMode(motorPin2, OUTPUT);
    pinMode(pwmPin, OUTPUT);
    pinMode(BTN, INPUT_PULLDOWN);
    pinMode(POT, INPUT);

    // Attach interrupt for button
    attachInterrupt(digitalPinToInterrupt(BTN), isr, RISING);

    // Initialize PWM
    ledcSetup(PWM_CHANNEL, PWM_FREQ, PWM_RESOLUTION);
    ledcAttachPin(pwmPin, PWM_CHANNEL);

    // Initialize load cell
    scale.begin(LCOUT, LCCLOCK);
    scale.set_scale(2054.75); // Calibration factor
    scale.tare(); // Reset scale
}

void loop() {
    unsigned long currentTime = millis();
    switch (state) {
        case SETTING:
            if (!messageDisplayed) {
                Serial.println("Place the cup on the scale.");
                lcd.clear();
                lcd.print("Place cup.");
                delay(1000);
```

```
            lcd.clear();
            lcd.print("Select weight.");
            messageDisplayed = true;
            messageDisplayTime = millis();
        }

        if (CheckForButtonPress()) {
            messageDisplayed = false;
            state = SETWAIT;
            break;
        }

        if (millis() - messageDisplayTime >= messageDisplayDuration) {
            checkPot();
        }
        break;

    case SETWAIT:
        stopMotor();
        if (!messageDisplayed) {
            Serial.println("Press button again to confirm weight.");
            lcd.clear();
            lcd.print("Confirm weight.");
            messageDisplayed = true;
            messageDisplayTime = millis();
        }

        if (millis() - messageDisplayTime >= messageDisplayDuration) {
            if (CheckForButtonPress()) {
                Serial.print("Weight confirmed. Target weight: ");
                lcd.clear();
                lcd.print("Weight: ");
                lcd.print(desiredWeight);
                Serial.println(desiredWeight);
                messageDisplayed = false;
                state = CONFIRMGO;
            }
        }
        break;
```

```
        case CONFIRMGO:
            if (!messageDisplayed) {
                Serial.println("Taring scale and starting motor...");
                lcd.clear();
                lcd.print("Starting motor...");
                scale.tare();
                messageDisplayed = true;
                messageDisplayTime = millis();
            } else if (millis() - messageDisplayTime >=
messageDisplayDuration) {
                currentWeight = scale.get_units();
                runMotorUntilWeight(153, currentWeight, threshold);
                state = REACHED;
            }
            break;

        case REACHED:
            if (!messageDisplayed) {
                Serial.println("Target weight reached. Stopping motor.");
                lcd.clear();
                lcd.print("Weight reached.");
                stopMotor();
                messageDisplayed = true;
                messageDisplayTime = millis();
            }
            if (currentTime - messageDisplayTime >= resetDelay) {
                state = SETTING;
                messageDisplayed = false;
            }
            break;
    }
}

void stopMotor() {
    ledcWrite(PWM_CHANNEL, 0);
    digitalWrite(motorPin1, LOW);
    digitalWrite(motorPin2, LOW);
}

int smoothPotReading() {
```

```
    const int numSamples = 10;
    static int samples[numSamples];
    static int index = 0;
    static long total = 0;

    total -= samples[index];
    samples[index] = analogRead(POT);
    total += samples[index];
    index = (index + 1) % numSamples;

    return total / numSamples;
}

void checkPot() {
    int smoothedReading = smoothPotReading();
    desiredWeight = map(smoothedReading, 0, 4095, 0, MAX_WEIGHT);
    threshold = desiredWeight;
    Serial.print("Desired Weight: ");
    Serial.println(desiredWeight);
}

void runMotorUntilWeight(int speed, float currentWeight, int threshold) {
    while (currentWeight < threshold) {
        currentWeight = scale.get_units();

        if (currentWeight < threshold - 20) {
            ledcWrite(PWM_CHANNEL, speed);
            digitalWrite(motorPin1, HIGH);
            digitalWrite(motorPin2, LOW);
        } else {
            ledcWrite(PWM_CHANNEL, speed - 60);
            digitalWrite(motorPin1, HIGH);
            digitalWrite(motorPin2, LOW);
        }

        Serial.print("Current weight: ");
        Serial.println(currentWeight);

        delay(100);
    }
```

```
    stopMotor();
    Serial.println("Motor stopped after reaching target weight.");
}

bool CheckForButtonPress() {
    if (buttonIsPressed) {
        buttonIsPressed = false;
        return true;
    }
    return false;
}
```