Automatic Ramen Maker: Dispensing Systems

ME 102B Group 23: Andrew Liu, Jacqueline Montoya, Phoebe Liu

Opportunity

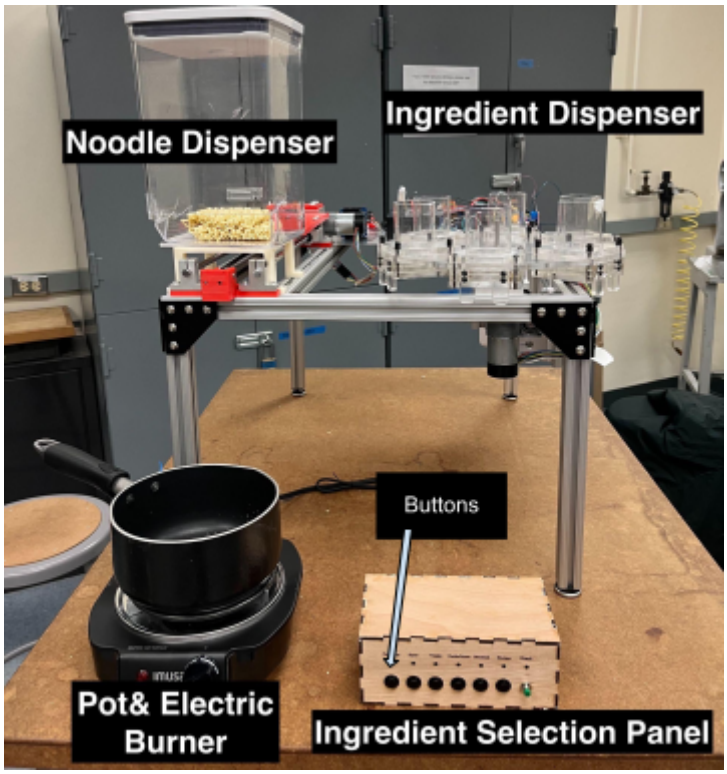
According to the World Instant Noodles Association, there are more than 100 billion servings of instant noodles consumed annually. To target this market, we created a foundation for an automated ramen maker. Our system focuses on the dispensing of ingredients into a pot. This allows the user to customize their instant ramen to their liking. It can be made available for individual households, community settings, and in convenience stores (e.g. similar to South Korea).

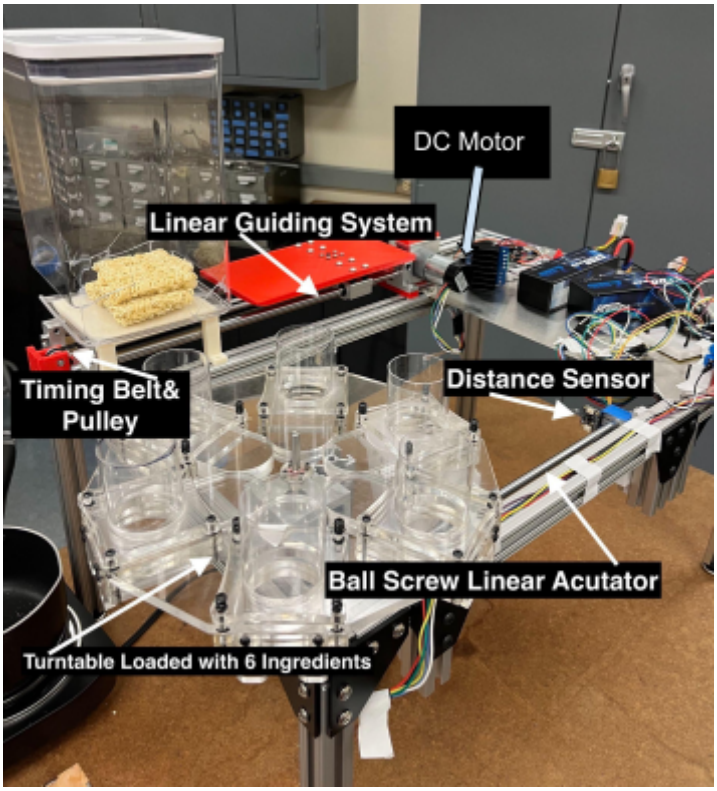
High Level Strategy

Our dispensing system includes three separate subsystems - ingredient selection panel, linear dispensing system, and rotary dispensing system. Each subsystem is controlled by one ESP32 and we used ESP-NOW to wirelessly communicate between microcontrollers. First, the user interacts with the display by selecting the ingredients they desire. Each ingredient has three options - ingredient not selected, one portion, or two portions. LEDs are used to visually communicate to the user which ingredients have been selected. When the start button is pressed, the ingredient values are sent over to the dispensing subsystems. Then, the linear dispensing system pushes out one block of instant ramen and returns to its original position. This is done by tracking down the current position using encoder and pivot to different states when certain event checkers are triggered. For the linear system connected to the rotating turntable, the analog LiDAR sensor works as a limit switch. Simultaneously, the rotary system uses the ingredient values as inputs to dispense the corresponding ingredient and number of portions. We utilized PI control on the angular speed to orient the turntable to a relatively accurate angular position. The integration of the systems successfully dispenses correct portions into a pot.

In addition to the selection panel, linear subsystem, and rotary subsystem, we initially wanted to create a fully automated system that poured in water and turned on/off the stove. Due to time constraints and having a small team, we decided to focus on the dispensing systems. We were able to manufacture parts to build our product and code the electronics needed to actuate our system. We reached our goal of integrating the systems together to automatically complete the processes we outlined once the start button was pushed. In our initial concept, we had included slides to guide the ingredients into the pot. Ultimately we decided to manually hold the pot to the dispensing locations to allow us time to focus on coding. Future work includes designing slides and additional subsystems to fully automate the process.

Integrated Physical Device





Critical Design Dimensions/Calculations

We used three 12V DC motors with a reduction ratio of 1:43 to actuate two linear motions and one rotary motion. The stall torque provided by this motor= 20 $kg \cdot cm$ so the maximum torque available would be $20 \times 60\% = 12\, kg \cdot cm$ which is equivalent to 1.176 $N \cdot m$ in a standard unit of torque.

Linear Actuation for dispensing the noodle

Assume: friction between the noodle and the bottom of the container is negligible; desired linear acceleration=$3m/s^2$

$Mass_{total} = m_{pushing\ plate} + m_{one\ ramen} = 0.385kg$ $\qquad$ $F_{pushing} = Mass_{total} \times a_{des} = 1.155N$

Calculate torque required from motor in a belt-pulley system and assume the efficiency of our G2 timing belt is 98%

$r_{pulley} = 8mm$ $\qquad$ $\tau_{req} = \frac{F_{pushing} \times r_{pulley}}{1000 \cdot \eta} = 9.4 \times 10^{-3} N \cdot m$

Rotary Actuation on the ingredient dispenser

6 tubes with radius $r_{tube} = 2.03 \times 10^{-2} m$ and a distance to the central shaft of D= 0.1016m

The acrylic base plate has thickness of $6.35 \times 10^{-3} m$ and radius of r_{plate}0.127m.

Given that acrylic density is 1180 kg/m^3

Then we assume $w_{desired} = \frac{1\, rev}{5\, sec} = 1.2566\, rad/s$ in a time interval of 0.1s

$m_{tube} = 50\, g$ $\quad and \quad$ $\alpha_{des} = w_{des}/\Delta t = 12.566\, rad/s$

Sum of moment of inertia,

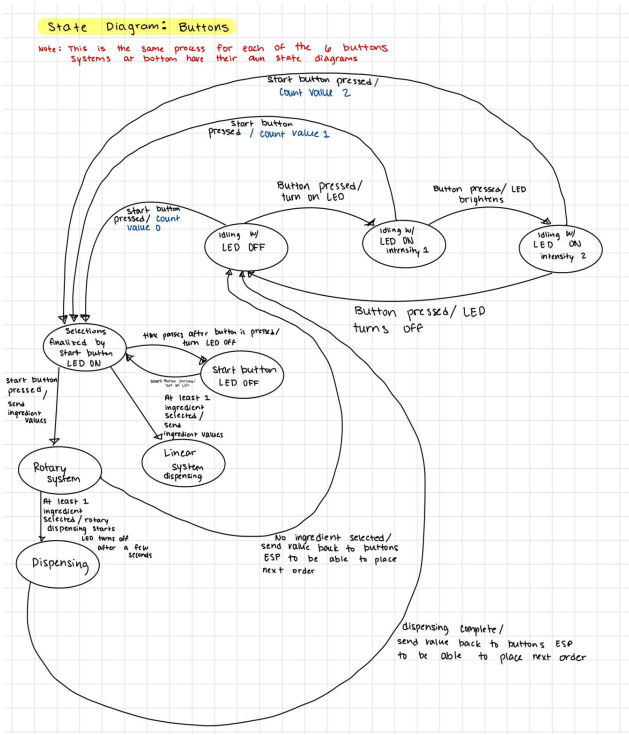
$I_{tot} = I_{plate} + 6 \times I_{tube} = \frac{1}{2} \times m_{plate} \times r_{plate}^2 + 6 \times (\frac{1}{2} \times m_{tube} \times r_{tube}^2 + m_{tube} \times D^2) = 6.22 \times 10^{-3}\, kg \cdot m^2$

$\tau_{des} = I_{tot} \cdot \alpha_{des} = 0.078\, N \cdot m < \tau_{motorMax}$

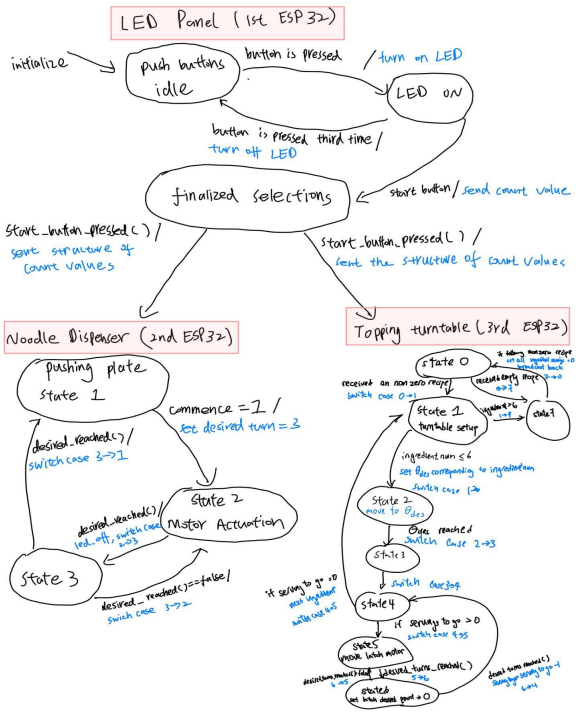
Both required torque are much smaller than the max torque provided by the 12V motor ($1.176N \cdot m$) so it is sufficient to actuate both the linear motion and rotary motion.

State Transition Diagrams

State Diagram for Selection Panel



State Diagram for Entire Ramen Making System



Circuit Diagrams

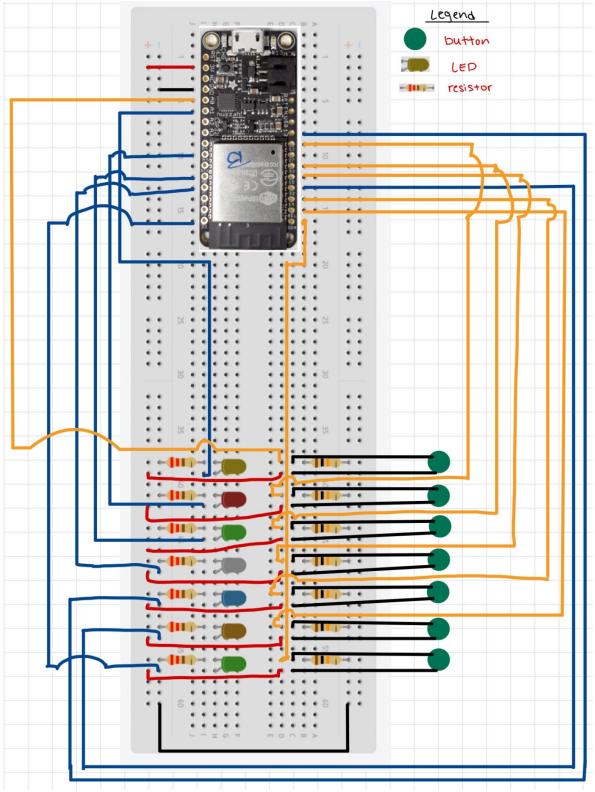


Fig.1: Circuit Diagram for Selection Panel

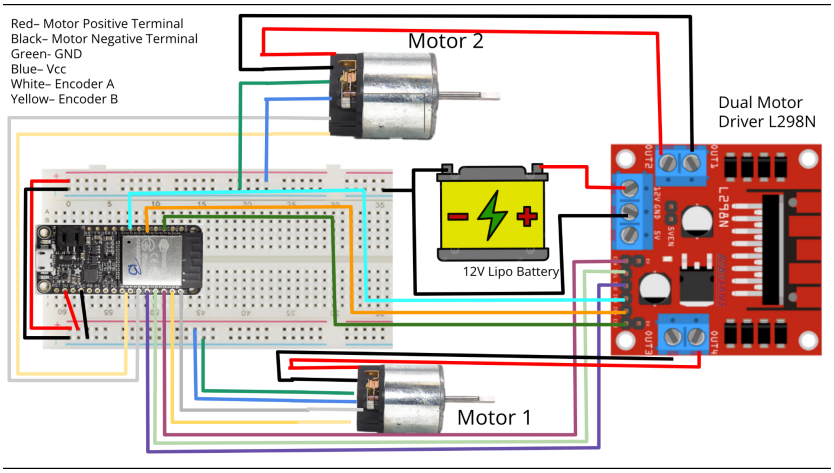


Fig 2. Circuit Diagram for the Ingredient Turntable

Final Thoughts/Reflection

After many collaborative meetings throughout the semester, we successfully created an integrated ramen ingredient dispensing system. This project requires hard work, dedication, and many hours went into the code and manufacturing. We went through an iterative design process, collaborating with one another to fix and refine where needed. Our advice for project selection is to decide what is manageable given the team size and time constraints. Our project was intensive, but with proper planning it can be accomplished. To further this project, we would have implemented more subsystems to create a fully functional automated system. As well as work on refining a system to deliver the dispensed ingredients into the pot.

Appendices

Appendix A: Bill of Materials

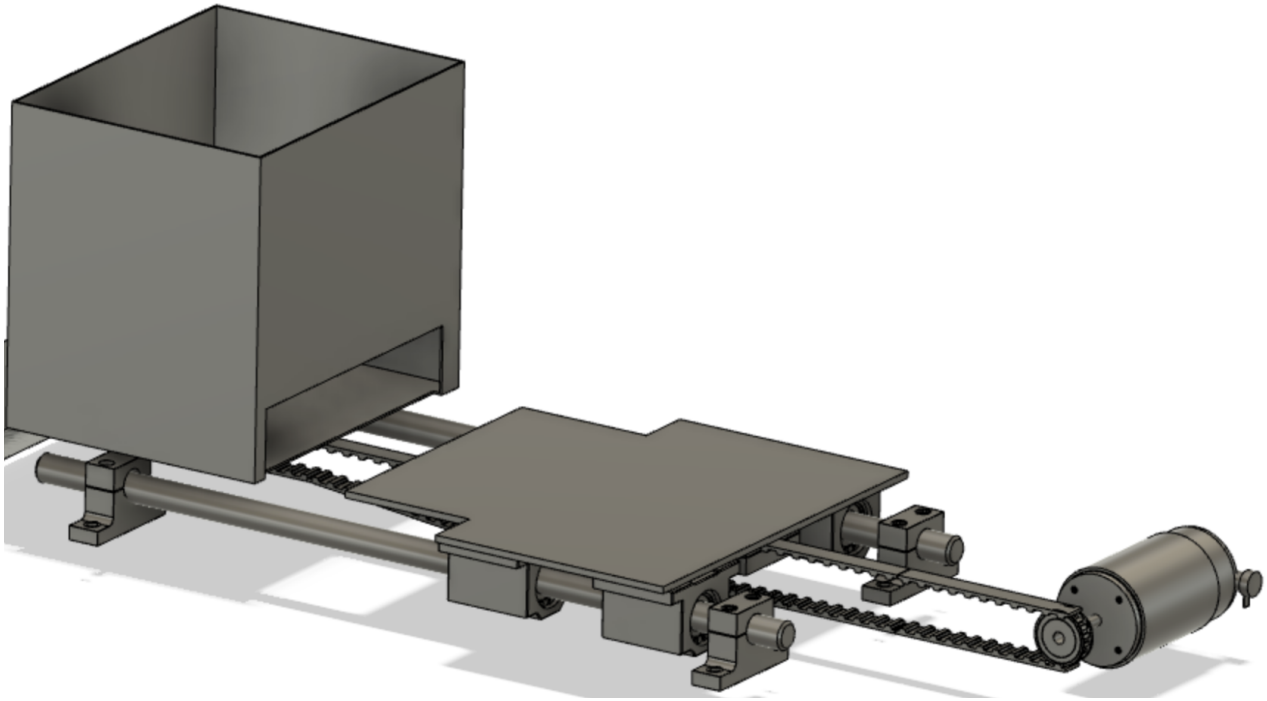
Ramen Maker's Purchase Portfolio					$ 506.78
Item Name	**Description**	**Price (ea.)**	**Quantity**	**Vendor**	**Subtotal**
T-Slot Rail 3' Long	Single 4-Slot Rail, Silver, 1" High x 1" Wide, Solid, 3' Long	$ 15.90	4	McMaster Carr	$ 63.60
Clear Acrylic Sheet	Clear Acrylic - 1/4" x 16" x 32"	$ 49.95	1	Jacobs Hall Material Store	$ 49.95
Aluminum Plate	24" x 24" x 0.032" 6061 Aluminum Plate (FabLight)	$ 16.06	1	Jacobs Hall Material Store	$ 16.06
Plywood	Plywood - 1/8" x 18" x 30"	$ 8.32	1	Jacobs Hall Material Store	$ 8.32
T-Slot Inside Corner Bracket	20Pcs Black Inside Corner Bracket Gusset with T-Slot Nuts & Screws, L-Shaped Aluminum Extrusion	$ 12.99	1	Amazon	$ 12.99
Electric Single Burner	Electric Single Burner 1100 Watts	$ 13.78	1	Amazon	$ 13.78
Timing Belt Pulley wheel	GT2 Pulley 20 Teeth 6mm bore 6mm Width 20T Timing Belt Pulley Wheel Aluminum	$ 6.60	1	Amazon	$ 6.60
Timing Belt Clamping Plate	*2GT Type Timing Belt Connector Clamp Plate Connecting Plate for 6mm Width Timing Belt*	$ 17.41	1	Amazon	$ 17.41
Idler Pulley	*20 Teeth 3mm Bore Idler Timing Pulley with Bearing*	$ 7.25	1	Amazon	$ 7.25
Breadboard+ jumper wires	Breadboard Jumper Wires Kit	$ 10.95	1	Amazon	$ 10.95
Linear Rail Slider Set	Linear Motion Bearing Block for SBR16 Linear Guide Rail	$ 42.89	1	Amazon	$ 42.89
Airtight Food Storage	Noodle Container	$ 20.94	1	Amazon	$ 20.94
acrylic cement	SCIGRIP 10315 16 Acrylic Plastic Cement	$ 23.53	1	Amazon	$ 23.53
Corner Bracket Plate	L Shape Joint Plates:6063 aluminum;T-Nuts and Hex Screw	$ 16.71	1	Amazon	$ 16.71

T Shape Joint Plates	outside corner bracket, with M5 slide in t-slot nuts, M5x10mm hex socket cap screw bolt	$ 9.34	1	Amazon	$ 9.34
M5 T-nuts	M5 T-Nuts Assortment Kit	$ 16.05	1	Amazon	$ 16.05
Motor Driver	Motor Driver Controller Board Module	$ 12.88	1	Amazon	$ 12.88
Acrylic Tube	Acrylic Tube (1.63"ID 1.7"OD 1' long)	$ 13.49	1	Amazon	$ 13.49
Dried Shrimp	Dried Shrimp	$ 4.99	1	Amazon	$ 4.99
Seaweed Flakes	Seaweed Flakes	$ 9.99	1	Amazon	$ 9.99
Dehydrated Vegetable Flakes	Dehydrated Vegetable Flakes	$ 9.99	1	Amazon	$ 9.99
Onion and Garlic Flakes	Onion and Garlic Flakes	$ 9.49	1	Amazon	$ 9.49
Pork Noodle Soup Powder	Pork Noodle Soup Powder	$ 9.41	1	Amazon	$ 9.41
Round Spacer Washer	Round Spacer Washer, 200 Pack Nylon 4.2mm ID x 7mm OD x 8mm L	$ 6.99	1	Amazon	$ 6.99
Threaded Stem Furniture Glides	Threaded Stem Furniture Glides, 1/4 Inch Stem Diameter	$ 4.17	1	Amazon	$ 4.17
Flange Coupling Connector	Flange Coupling Connector	$ 8.49	1	Amazon	$ 8.49
Nuts and Bolts Assortment Kit	Nuts and Bolts assortment kit	$ 9.99	1	Amazon	$ 9.99
Shaft Couplings	Shaft Couplings	$ 7.99	1	Amazon	$ 7.99
Shaft Optical Axis Set	Shaft Optical Axis Set	$ 22.99	1	Amazon	$ 22.99
Food Grade Tubing	Food Grade Tubing	$ 14.59	1	Amazon	$ 14.59
Push Button Switch	Push Buttons 12 V Black Waterproof	$ 11.99	1	Amazon	$ 11.99
Momentary Push Button Switch	Momentary Mini Push Button Switch	$ 6.98	1	Amazon	$ 6.98
LED Assortment Kit	LED Assortment Kit	$ 5.99	1	Amazon	$ 5.99

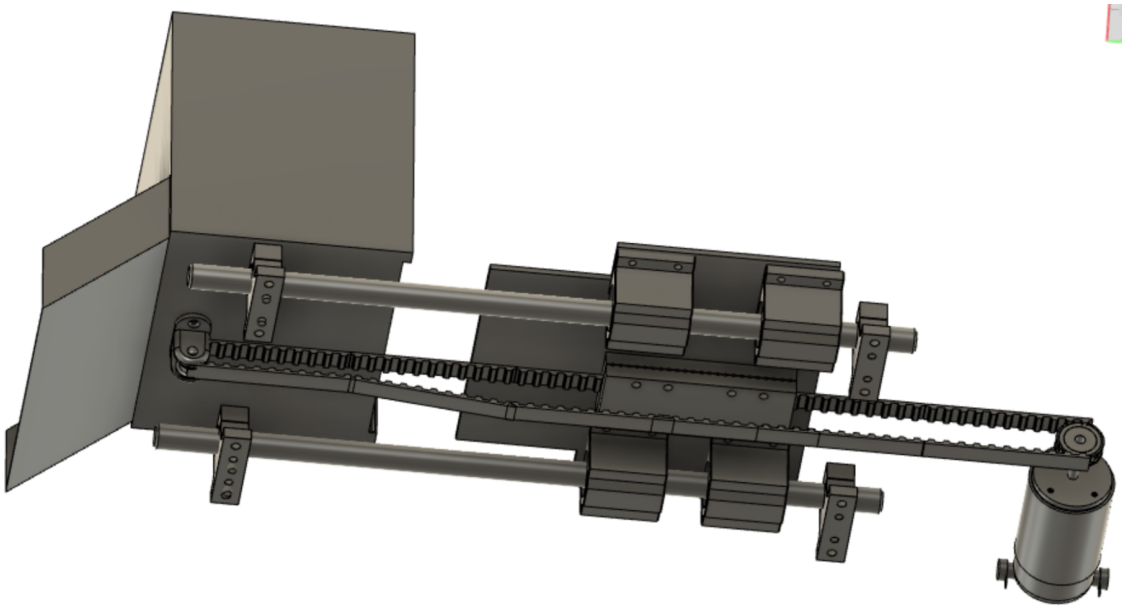
Appendix B: CAD of Mechanical Transmission Elements

Noodle Dispenser

Isometric View

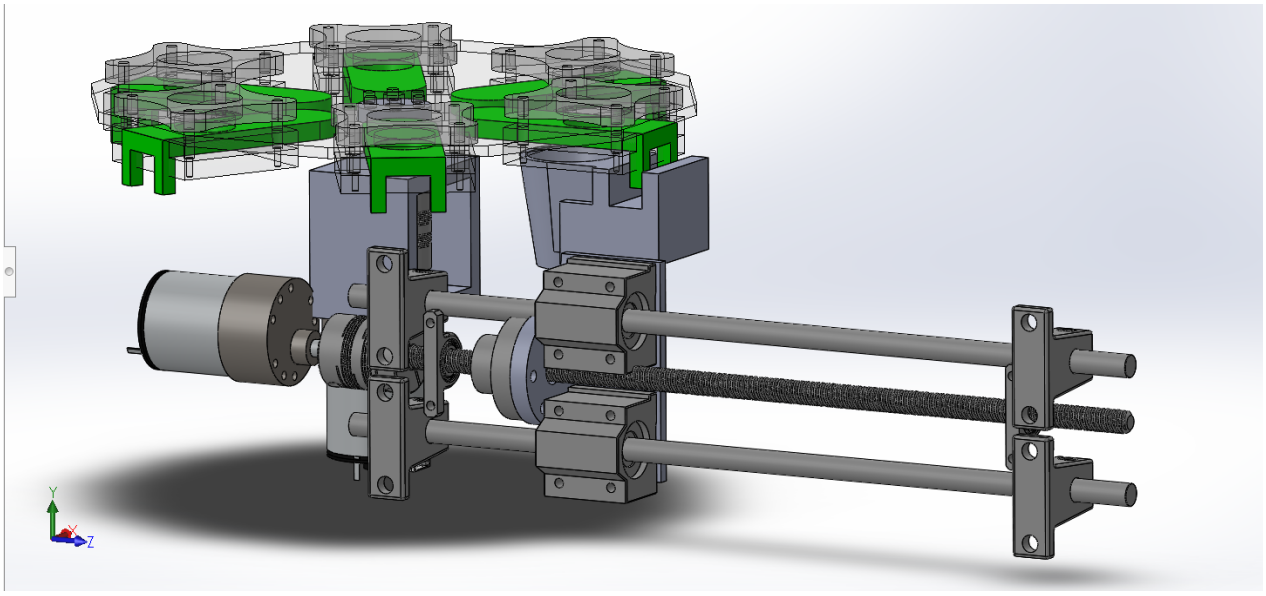


View from the Bottom

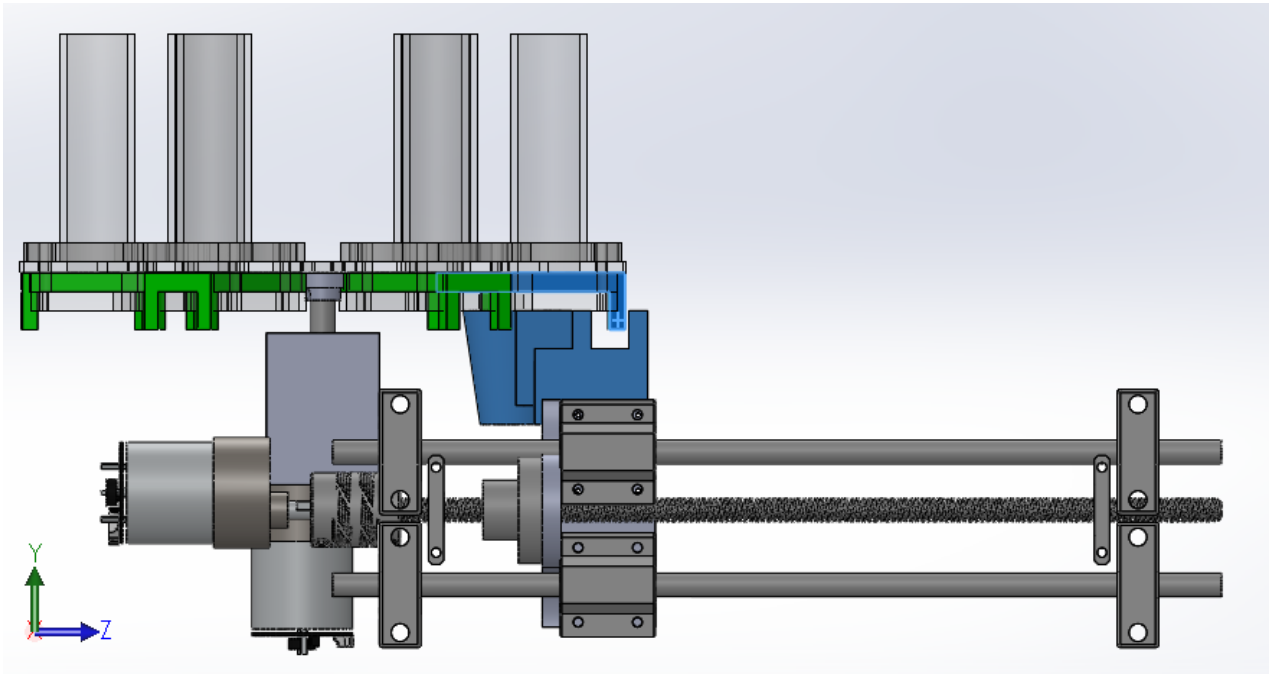


Condiment Dispenser

Angled Side View



Cross Sectional View



Appendix C: Full Code

Ingredient Selection Panel Code

```
//Script for Buttons and LEDs
#include <Arduino.h>
#include <esp_now.h>
#include <WiFi.h>

//Define Constants and Variables
#define LED1 25
#define LED2 4
#define LED3 18
#define LED4 19
#define LED5 13
#define LED6 32
#define LED7 21
#define ledChannel1 1
#define ledChannel2 2
#define ledChannel3 3
#define ledChannel4 4
#define ledChannel5 5
#define ledChannel6 6
#define ledChannel7 7
#define freq 5000
#define res 8
#define dutyCycle0 0
#define dutyCycle1 15
#define dutyCycle2 255

#define BTN1 26
#define BTN2 12
#define BTN3 27
#define BTN4 15
#define BTN5 14
#define BTN6 22
#define BTN7 23

int commence = 1;

int state1 = 0;
volatile bool button1IsPressed = false;
int state2 = 0;
volatile bool button2IsPressed = false;
int state3 = 0;
volatile bool button3IsPressed = false;
```

```
int state4 = 0;
volatile bool button4IsPressed = false;
int state5 = 0;
volatile bool button5IsPressed = false;
int state6 = 0;
volatile bool button6IsPressed = false;
int state7 = 0;
volatile bool button7IsPressed = false;
int ready_on = 0;
int ready_off = 0;

volatile bool debounceT = false;
hw_timer_t * timer0 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
void IRAM_ATTR onTime0(){
  portENTER_CRITICAL_ISR(&timerMux0);
    debounceT = true;
  portEXIT_CRITICAL_ISR(&timerMux0);
    timerStop(timer0);
}

//Initialization
void IRAM_ATTR isr1() {
  button1IsPressed = true;
  timerStart(timer0);
}
void IRAM_ATTR isr2() {
  button2IsPressed = true;
  timerStart(timer0);
}
void IRAM_ATTR isr3() {
  button3IsPressed = true;
  timerStart(timer0);
}
void IRAM_ATTR isr4() {
  button4IsPressed = true;
  timerStart(timer0);
}
void IRAM_ATTR isr5() {
  button5IsPressed = true;
  timerStart(timer0);
}
void IRAM_ATTR isr6() {
```

```
  button6IsPressed = true;
  timerStart(timer0);
}
void IRAM_ATTR isr7(){
  button7IsPressed = true;
  timerStart(timer0);
}


// ESP NOW code
uint8_t broadcastAddress[] = {0x94, 0xB9, 0x7E, 0x6B, 0xAA, 0x90}; //turntabl+latch
esp32
uint8_t broadcastAddress_noodle[] = {0x0C, 0xDC, 0x7E, 0xCB, 0x0C, 0xF8}; //noodle
dispenser esp32
typedef struct struct_message {
  int ing1;
  int ing2;
  int ing3;
  int ing4;
  int ing5;
  int ing6;
  int commence;
} struct_message;

struct_message myData;

esp_now_peer_info_t peerInfo;

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery
Fail");
}

typedef struct struct_message_2 {
  int commence;
  //int new;
} struct_message_2;

struct_message_2 incomingReadings;

void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
  memcpy(&incomingReadings, incomingData, sizeof(incomingReadings));
```

```
  commence = incomingReadings.commence;
  Serial.println(commence);
}

void setup() {
  //Configure LED PWM functionalities
  ledcSetup(ledChannel1,freq,res);
  ledcSetup(ledChannel2,freq,res);
  ledcSetup(ledChannel3,freq,res);
  ledcSetup(ledChannel4,freq,res);
  ledcSetup(ledChannel5,freq,res);
  ledcSetup(ledChannel6,freq,res);
  ledcSetup(ledChannel7,freq,res);

  //Attach the channel to the GPIO to be controlled
  ledcAttachPin(LED1,ledChannel1);
  ledcAttachPin(LED2,ledChannel2);
  ledcAttachPin(LED3,ledChannel3);
  ledcAttachPin(LED4,ledChannel4);
  ledcAttachPin(LED5,ledChannel5);
  ledcAttachPin(LED6,ledChannel6);
  ledcAttachPin(LED7,ledChannel7);

  //Buttons
  pinMode(BTN1,INPUT);
  attachInterrupt(BTN1,isr1,RISING);
  pinMode(BTN2,INPUT);
  attachInterrupt(BTN2,isr2,RISING);
  pinMode(BTN3,INPUT);
  attachInterrupt(BTN3,isr3,RISING);
  pinMode(BTN4,INPUT);
  attachInterrupt(BTN4,isr4,RISING);
  pinMode(BTN5,INPUT);
  attachInterrupt(BTN5,isr5,RISING);
  pinMode(BTN6,INPUT);
  attachInterrupt(BTN6,isr6,RISING);
  pinMode(BTN7,INPUT);
  attachInterrupt(BTN7,isr7,RISING);
  Serial.begin(115200);

  TimerInterruptInit();

  WiFi.mode(WIFI_STA);
```

```
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }
  esp_now_register_send_cb(OnDataSent);
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
  esp_now_register_recv_cb(OnDataRecv);

  // ESP noodle
  // Register peer
  memcpy(peerInfo.peer_addr, broadcastAddress_noodle, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }

}

//Main Loop
void loop() {
  delay(100);

  switch (state1) {
    //same event and service pattern for each button
    case 0:
      if(CheckForButtonPress1() && commence == 1) { //event
        //Serial.println("Button 1 pressed. State Button 1: 1");
        led1_on1(); //service
        state1 = 1; //service
      }
      break;
    case 1:
      if(CheckForButtonPress1() && commence == 1){ //event
```

```
      led1_on2(); //service
      state1 = 2; //service
      //Serial.println("Button 1 pressed. State Button 1: 2");
    }
    break;
  case 2:
    if(CheckForButtonPress1() && commence == 1){ //event
      led1_off(); //service
      state1 = 0; //service
      //Serial.println("Button 1 pressed. State Button 1: 0");
    }
    break;
}

switch (state2) {
  case 0:
    if(CheckForButtonPress2() && commence == 1) {
      led2_on1();
      state2 = 1;
    }
    break;
  case 1:
    if(CheckForButtonPress2() && commence == 1){
      led2_on2();
      state2 = 2;
    }
    break;
  case 2:
    if(CheckForButtonPress2() && commence == 1){
      led2_off();
      state2 = 0;
    }
    break;
}

switch (state3) {
  case 0:
    if(CheckForButtonPress3() && commence == 1) {
      led3_on1();
      state3 = 1;
    }
    break;
  case 1:
```

```
    if(CheckForButtonPress3() && commence == 1){
      led3_on2();
      state3 = 2;
    }
    break;
  case 2:
    if(CheckForButtonPress3() && commence == 1){
      led3_off();
      state3 = 0;
    }
    break;
}

switch (state4) {
  case 0:
    if(CheckForButtonPress4() && commence == 1) {
      led4_on1();
      state4 = 1;
    }
    break;
  case 1:
    if(CheckForButtonPress4() && commence == 1){
      led4_on2();
      state4 = 2;
    }
    break;
  case 2:
    if(CheckForButtonPress4() && commence == 1){
      led4_off();
      state4 = 0;
    }
    break;
}

switch (state5) {
  case 0:
    if(CheckForButtonPress5() && commence == 1) {
      led5_on1();
      state5 = 1;
    }
    break;
  case 1:
    if(CheckForButtonPress5() && commence == 1){
```

```
        led5_on2();
        state5 = 2;
      }
      break;
    case 2:
      if(CheckForButtonPress5() && commence == 1){
        led5_off();
        state5 = 0;
      }
      break;
  }

  switch (state6) {
    case 0:
      if(CheckForButtonPress6() && commence == 1) {
        led6_on1();
        state6 = 1;
      }
      break;
    case 1:
      if(CheckForButtonPress6() && commence == 1){
        led6_on2();
        state6 = 2;
      }
      break;
    case 2:
      if(CheckForButtonPress6() && commence == 1){
        led6_off();
        state6 = 0;
      }
      break;
  }

  if (commence == 1) {
    if (state1 == 0 && state2 == 0 && state3 == 0 && state4 == 0 && state5 == 0 &&
state6 == 0) {
      ready_on = 1;
    }

    else {
      ready_on = 2;
    }
  }
```

```
  switch (ready_on) {
    case 1:
      if(CheckForButtonPress7() && commence == 1) {
        led7_on2();
        //state7 = 1;
        Serial.println("Start button pressed!");
        Serial.println("Button states below in order of Buttons 1 to 6 from top to
bottom:");
        Serial.println(state1);
        Serial.println(state2);
        Serial.println(state3);
        Serial.println(state4);
        Serial.println(state5);
        Serial.println(state6);
        //commence = 2;
        myData.ing1 = state1;
        myData.ing2 = state2;
        myData.ing3 = state3;
        myData.ing4 = state4;
        myData.ing5 = state5;
        myData.ing6 = state6;
        myData.commence = 2;
        esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &myData,
sizeof(myData));
        Serial.println("Restart Order!");
        delay(5000);
        led7_off();
      }

    case 2:
      if(CheckForButtonPress7() && commence == 1) { //event
          //everything below is the service done when Button 7 (the "start" button is
pressed)
        led7_on2();
        //state7 = 1;
        Serial.println("Start button pressed!");
        Serial.println("Button states below in order of Buttons 1 to 6 from top to
bottom:");
        Serial.println(state1);
        Serial.println(state2);
        Serial.println(state3);
        Serial.println(state4);
```

```
        Serial.println(state5);
        Serial.println(state6);
        myData.ing1 = state1;
        myData.ing2 = state2;
        myData.ing3 = state3;
        myData.ing4 = state4;
        myData.ing5 = state5;
        myData.ing6 = state6;
        myData.commence = 1;
        esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &myData,
sizeof(myData));
        esp_err_t result2 = esp_now_send(broadcastAddress_noodle, (uint8_t *)
&myData, sizeof(myData));
        Serial.println("Order sent!");
        delay(5000);
        state7 = 0;
        state1 = 0;
        state2 = 0;
        state3 = 0;
        state4 = 0;
        state5 = 0;
        state6 = 0;
        led1_off();
        led2_off();
        led3_off();
        led4_off();
        led5_off();
        led6_off();
        led7_off();
        Serial.println("Ready to place order!");
      commence = 0;
      }
  }
    //break;
    //case 1:
    // note this case was not used so far but might be implemented at a later stage
    // need to tune the details of how the subsystems will communicate
      //if(CheckForButtonPress7()){
        //led7_off();
        //state7 = 0;
      //}
      //break;
```

```
  //ledcWrite(ledChannel1,dutyCycle1);
  //ledcWrite(ledChannel2,dutyCycle1);
  //ledcWrite(ledChannel3,dutyCycle1);
  //ledcWrite(ledChannel4,dutyCycle1);
  //ledcWrite(ledChannel5,dutyCycle1);
  //ledcWrite(ledChannel6,dutyCycle1);

}

void TimerInterruptInit(){
  timer0 = timerBegin(0,80,true);
  timerAttachInterrupt(timer0,&onTime0,true);
  timerAlarmWrite(timer0, 250000,true);
  timerAlarmEnable(timer0);
  timerStop(timer0);
}

bool CheckForButtonPress1(){
  if(debounceT && button1IsPressed){
    portENTER_CRITICAL(&timerMux0);
      debounceT = false;
    portEXIT_CRITICAL(&timerMux0);
    timerStop(timer0);
    button1IsPressed = false;
    return true;
  }
  else{
      return false;
  }
}
void led1_on1(){
  ledcWrite(ledChannel1,dutyCycle1);
}
void led1_on2(){
  ledcWrite(ledChannel1,dutyCycle2);
}
void led1_off(){
  ledcWrite(ledChannel1,dutyCycle0);
}

bool CheckForButtonPress2(){
```

```
  if(debounceT && button2IsPressed){
    portENTER_CRITICAL(&timerMux0);
      debounceT = false;
    portEXIT_CRITICAL(&timerMux0);
    timerStop(timer0);
    button2IsPressed = false;
    return true;
  }
  else{
      return false;
  }
}
void led2_on1(){
  ledcWrite(ledChannel2,dutyCycle1);
}
void led2_on2(){
  ledcWrite(ledChannel2,dutyCycle2);
}
void led2_off(){
  ledcWrite(ledChannel2,dutyCycle0);
}

bool CheckForButtonPress3(){
  if(debounceT && button3IsPressed){
    portENTER_CRITICAL(&timerMux0);
      debounceT = false;
    portEXIT_CRITICAL(&timerMux0);
    timerStop(timer0);
    button3IsPressed = false;
    return true;
  }
  else{
      return false;
  }
}
void led3_on1(){
  ledcWrite(ledChannel3,dutyCycle1);
}
void led3_on2(){
  ledcWrite(ledChannel3,dutyCycle2);
}
void led3_off(){
  ledcWrite(ledChannel3,dutyCycle0);
```

```
}

bool CheckForButtonPress4(){
  if(debounceT && button4IsPressed){
    portENTER_CRITICAL(&timerMux0);
      debounceT = false;
    portEXIT_CRITICAL(&timerMux0);
    timerStop(timer0);
    button4IsPressed = false;
    return true;
  }
  else{
      return false;
  }
}
void led4_on1(){
  ledcWrite(ledChannel4,dutyCycle1);
}
void led4_on2(){
  ledcWrite(ledChannel4,dutyCycle2);
}
void led4_off(){
  ledcWrite(ledChannel4,dutyCycle0);
}

bool CheckForButtonPress5(){
  if(debounceT && button5IsPressed){
    portENTER_CRITICAL(&timerMux0);
      debounceT = false;
    portEXIT_CRITICAL(&timerMux0);
    timerStop(timer0);
    button5IsPressed = false;
    return true;
  }
  else{
      return false;
  }
}
void led5_on1(){
  ledcWrite(ledChannel5,dutyCycle1);
}
void led5_on2(){
  ledcWrite(ledChannel5,dutyCycle2);
```

```
}
void led5_off(){
  ledcWrite(ledChannel5,dutyCycle0);
}

bool CheckForButtonPress6(){
  if(debounceT && button6IsPressed){
    portENTER_CRITICAL(&timerMux0);
      debounceT = false;
    portEXIT_CRITICAL(&timerMux0);
    timerStop(timer0);
    button6IsPressed = false;
    return true;
  }
  else{
      return false;
  }
}
void led6_on1(){
  ledcWrite(ledChannel6,dutyCycle1);
}
void led6_on2(){
  ledcWrite(ledChannel6,dutyCycle2);
}
void led6_off(){
  ledcWrite(ledChannel6,dutyCycle0);
}

bool CheckForButtonPress7(){
  if(debounceT && button7IsPressed){
    portENTER_CRITICAL(&timerMux0);
      debounceT = false;
    portEXIT_CRITICAL(&timerMux0);
    timerStop(timer0);
    button7IsPressed = false;
    return true;
  }
  else{
      return false;
  }
}
void led7_on2(){
  ledcWrite(ledChannel7,dutyCycle2);
```

```
}
void led7_off(){
  ledcWrite(ledChannel7,dutyCycle0);
}
```

Noodle Dispenser Code

```
#include <esp_now.h>
#include <WiFi.h>
#include <ESP32Encoder.h>
#include <Arduino.h>
ESP32Encoder encoder;

#define LED_PIN 13 // declare the builtin LED pin number
#define ENC_1A 17 //CLK
#define ENC_1B 21 //DT (90 deg phase change with CLK)

int enable1Pin = 16; //for both l and r enable pin
int motor1Pin1 = 19;
int motor1Pin2 = 18;
//encoder1 setup
int counter1 = 0;
String currentDir ="";

int desired_turns1=0;
const int countsperturn=1440;
int desired_counts1=0;
int count1_tolerance=30;

// Setting PWM properties
const int freq = 30000;
const int channel_0=0;
const int channel_1=1;
const int res = 8;
int dutyCycle = 200;

int state = 1;
volatile bool buttonIsPressed = false;

int commence = 0;
int ingredient1 = 0;
int ingredient2 = 0;
int ingredient3 = 0;
int ingredient4 = 0;
int ingredient5 = 0;
```

```
int ingredient6 = 0;


// Structure example to receive data
// Must match the sender structure
typedef struct struct_message {
    int ing1;
    int ing2;
    int ing3;
    int ing4;
    int ing5;
    int ing6;
    int commence;
} struct_message;

// Create a struct_message called myData
struct_message myData;

// callback function that will be executed when data is received
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
  memcpy(&myData, incomingData, sizeof(myData));
  ingredient1 = myData.ing1;
  ingredient2 = myData.ing2;
  ingredient3 = myData.ing3;
  ingredient4 = myData.ing4;
  ingredient5 = myData.ing5;
  ingredient6 = myData.ing6;
  commence = myData.commence;
  Serial.println(ingredient1);
  Serial.println(ingredient2);
  Serial.println(ingredient3);
  Serial.println(ingredient4);
  Serial.println(ingredient5);
  Serial.println(ingredient6);
  Serial.print("Commence:");
  Serial.println(commence);
}

uint8_t broadcastAddress[] = {0x94, 0xB9, 0x7E, 0x6B, 0xA6, 0xF8};
typedef struct struct_message_2 {
  int commence;
  //int new;
} struct_message_2;
```

```
struct_message_2 incomingReadings;
esp_now_peer_info_t peerInfo;
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery
Fail");
}

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

  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);

  // Init ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }
  esp_now_register_send_cb(OnDataSent);
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
  esp_now_register_recv_cb(OnDataRecv);


  pinMode(LED_PIN, OUTPUT);
  // sets the pins as outputs:
  pinMode(motor1Pin1, OUTPUT);
  pinMode(motor1Pin2, OUTPUT);
  pinMode(enable1Pin, OUTPUT);
  // configure LED PWM functionalitites
  ledcSetup(channel_0, freq,res) ; // setup PWM channel for BST L_PWM
  ledcSetup(channel_1, freq,res) ; // setup PWM channel for BST R_PWM
 ledcAttachPin( motor1Pin1, channel_0) ; // Attach BST L_PWM
```

```
  ledcAttachPin( motor1Pin2, channel_1) ;
  digitalWrite(enable1Pin,HIGH) ;

  Serial.print("Forward with duty cycle: ");
  Serial.println(dutyCycle);
  // testing
  Serial.print("Testing DC Motor...");

    ESP32Encoder::useInternalWeakPullResistors = UP; // Enable the weak pull up
resistors
  encoder.attachHalfQuad(ENC_1A, ENC_1B); // Attache pins for use as encoder pins
  encoder.clearCount ( );
  encoder.setCount(0);  // set starting count value after attaching

  led_off();
}

void loop() {
  if(ingredient1==1 || ingredient2==1 || ingredient3==1 || ingredient4==1 ||
ingredient5==1 || ingredient6==1 || ingredient1==2 || ingredient2==2 ||
ingredient3==2 || ingredient4==2 || ingredient5==2 || ingredient6==2){
    delay(2000);
    commence = 1;
    ingredient1 = 0;
    ingredient2 = 0;
    ingredient3 = 0;
    ingredient4 = 0;
    ingredient5 = 0;
    ingredient6 = 0;
    //new = 1;
    incomingReadings.commence = commence;
    //incomingReadings.new = new;
    esp_err_t result = esp_now_send(broadcastAddress, (uint8_t*) &incomingReadings,
sizeof(incomingReadings));
    Serial.println("Commence sent!");
  }
  if(commence == 2){
    delay(10000);
    commence = 1;
    ingredient1 = 0;
    ingredient2 = 0;
    ingredient3 = 0;
    ingredient4 = 0;
```

```
    ingredient5 = 0;
    ingredient6 = 0;
    //new = 1;
    incomingReadings.commence = commence;
    //incomingReadings.new = new;
    esp_err_t result = esp_now_send(broadcastAddress, (uint8_t*) &incomingReadings,
sizeof(incomingReadings));
    Serial.println("Commence sent! extra");
  }
  //start button is pressed, dispensing start to work
  if(commence==1){
    counter1 = encoder.getCount( );
  buttonIsPressed = true;
  ////////////End of Encoder 1
  switch (state) {
    case 1:
      Serial.println("This is state 1");
      led_off();
      motor1_off();
      if (CheckForButtonPress() == true) {

        led_on();
        desired_turns1=3;
        desired_counts1=desired_turns1*countsperturn;
        state = 2;
      }
      break;

    case 2:
      Serial.print("This is state 2, ");
      Serial.print("Direction: ");
      Serial.print(currentDir);
      Serial.print(" | counter1: ");
      Serial.print(counter1);
      Serial.print(" | desired counts: ");
      Serial.println(desired_counts1);
      if(desired_turns_reached()==false){
        if (counter1<=desired_counts1 + count1_tolerance){

dutyCycle=float(abs(counter1-desired_counts1))/float(3*countsperturn)*100+100;//dutyc
ycle range from 100-200

          led_off();
```

```
          motor1_forward();
          Serial.println("forward with dutycycle");
          Serial.println(dutyCycle);
          delay(50);
        }
        else if (counter1>=desired_counts1 - count1_tolerance){

dutyCycle=float(abs(counter1-desired_counts1))/float(3*countsperturn)*100+100;

          led_off();
          motor1_backward();
          Serial.println("backward with dutycycle");
          Serial.println(dutyCycle);
          delay(50);
        }
      }
      else if(desired_turns_reached()){
        led_off();
        state = 3;
      }
      break;
    case 3:
      Serial.println("This is state 3");
      motor1_off();
      desired_turns1=0;
      desired_counts1=desired_turns1*countsperturn;
      if(desired_turns_reached()==false){
        state=2;
      }
      else if(desired_turns_reached()){
        led_off();
        state = 1;
        commence=0;
      }
      break;
  }

    }
}
bool desired_turns_reached() {
  if (counter1<=desired_counts1 + count1_tolerance && counter1>=desired_counts1 -
count1_tolerance){
    return true;
```

```
  }
  else {
    return false;
  }
}

bool CheckForButtonPress() {
  if (buttonIsPressed == true){
    buttonIsPressed = false;
    led_on();
    return true;
  }
  else {
    return false;
  }
}
void motor1_forward() {

ledcWrite(channel_0, 0);
ledcWrite(channel_1, dutyCycle);
  currentDir="forward";
}
void motor1_backward() {

  ledcWrite(channel_0, dutyCycle);
ledcWrite(channel_1, 0);
  currentDir="backward";
}
void motor1_off() {
  ledcWrite(channel_0, 0);
ledcWrite(channel_1, 0);
  currentDir="stopped";
}

void led_on() {
  digitalWrite(LED_PIN, HIGH);
}

void led_off() {
  digitalWrite(LED_PIN, LOW);
}
```

Dry Ingredients Dispenser Code

```
//#include "Wire.h"
//#include "VL53L0X.h"

//VL53L0X tofSensor;

#include <Arduino.h>
#define LED_PIN 13 // declare the builtin LED pin number
#define BTN 25  // declare the button ED pin number

// Motor 1
int enable1Pin = 16;
int motor1Pin1 = 19;
int motor1Pin2 = 18;
#define ENC_1A 17 //CLK
#define ENC_1B 21 //DT (90 deg phase change with CLK)

// Motor 2 turntables
int enable2Pin = 27;
int motor2Pin1 = 15;
int motor2Pin2 = 14;
#define ENC_2A 4 //CLK
#define ENC_2B 5 //DT (90 deg phase change with CLK)



//encoder1 setup
int counter1 = 0; //current encoder count
String currentDir1 ="";//current encoder count direction

int desired_turns1=0;
const int countsperturn1=1440; //counts per revolution (by trial)
int desired_counts1=0;
int count1_tolerance=25; //encoder count tolerance

//encoder2 setup
const int countsperturn=1410; //encoder counts per 1 revolution
//int count1_tolerance=350;
int count2_slow_tolerance=200; //postion (encoder count) error tolerance
int count2_stop_tolerance=25;

//PI controller constants
float Kp=2;
```

```
float Ki=2;
int KiMax=125;
float e=0; //error
float sumE=0; //sum error
int D = 0;// adjusted pwm

int omegaSpeed = 0;
int omegaDes = 0;
int enc_count1 = 0; // encoder count new
int enc_count0 = 0; // encoder count old
//Setup interrupt variables ----------------------------
volatile bool deltaT0 = false;
volatile bool deltaT = false;     // check timer interrupt 2
hw_timer_t * timer0 = NULL;
hw_timer_t * timer1 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;


int theta = 0;// current count
int thetaDes = 0;//desired encoder count
int thetaDes_interval=countsperturn/6;//interval to increase by eac time button is
pressed (360/6 = 60 degrees)
int condiment=0;// current condiment
int turns=0;// current turns

int save_current_state=0;
volatile bool interrupted_from_case= false;
int save_current_state_latch=0;
volatile bool interrupted_from_case_latch= false;
int last_state=0;

int Currenttime=0;
int transition_time=0;
String status ="";//current status to print
int ingredient_num= 1;
int serving_to_go=0;
int serving=0;
int revs=0;
#include <esp_now.h>
#include <WiFi.h>

////
```

```
int commence = 0;
int ingredient1 = 0;
int ingredient2 = 0;
int ingredient3 = 0;
int ingredient4 = 0;
int ingredient5 = 0;
int ingredient6 = 0;

/////

// Structure example to receive data
// Must match the sender structure
typedef struct struct_message {
    int ing1;
    int ing2;
    int ing3;
    int ing4;
    int ing5;
    int ing6;
    int commence;
    //int b;
    //int c;
    //int d;
} struct_message;

// Create a struct_message called myData
struct_message myData;

// callback function that will be executed when data is received
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
  memcpy(&myData, incomingData, sizeof(myData));
  ingredient1 = myData.ing1;
  ingredient2 = myData.ing2;
  ingredient3 = myData.ing3;
  ingredient4 = myData.ing4;
  ingredient5 = myData.ing5;
  ingredient6 = myData.ing6;
  commence = myData.commence;
  Serial.println(ingredient1);
  Serial.println(ingredient2);
  Serial.println(ingredient3);
  Serial.println(ingredient4);
```

```
  Serial.println(ingredient5);
  Serial.println(ingredient6);
  Serial.print("Commence:");
  Serial.println(commence);
}

uint8_t broadcastAddress[] = {0x94, 0xB9, 0x7E, 0x6B, 0xA6, 0xF8};
typedef struct struct_message_2 {
  int commence;
  //int new;
} struct_message_2;

struct_message_2 incomingReadings;
esp_now_peer_info_t peerInfo;
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nLast Packet Send Status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery
Fail");
}

// Setting PWM properties
const int freq1 = 30000;
const int pwmChannel1 = 1;
const int resolution1 = 8;
int dutyCycle1 = 250;

const int freq2 = 30000;
const int pwmChannel2 = 2;
const int resolution2 = 8;
int dutyCycle2 = 0;//dutycycle range is 200-255 with great friction
const int MAX_PWM_VOLTAGE = 250;
const int NOM_PWM_VOLTAGE = 150;

//initial setting
int state = 0;


volatile bool buttonIsPressed = false;
//void IRAM_ATTR isr() {  // the function to be called when interrupt is triggered
//  buttonIsPressed = true; //set buttonIsPressed = true
//}
```

```
#include <ESP32Encoder.h>
ESP32Encoder encoder;
ESP32Encoder encoder2;


void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux1);
  enc_count1 = encoder2.getCount();
  omegaSpeed=enc_count1-enc_count0;
  enc_count0=enc_count1;
  deltaT = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux1);
}

void IRAM_ATTR onTime0() {
  portENTER_CRITICAL_ISR(&timerMux0);
  deltaT0 = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux0);
}



void setup() {
  Serial.begin(115200);
  delay(3000);
  //lidar sensor
  //Wire.begin(23,22); //SDA,SCL
  // wait until serial port opens for native USB devices
  //if(tofSensor.init() !=true){
  //  Serial.println("could not initialize Tof Sensor");
  //}
    // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);

  // Init ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }
  esp_now_register_send_cb(OnDataSent);
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
```

```
  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
  esp_now_register_recv_cb(OnDataRecv);
  delay(5000);

  //pinMode(BTN,INPUT);
  //attachInterrupt(BTN, isr, RISING); // attach isr interrupt to button when rising
  pinMode(LED_PIN, OUTPUT);
  // sets the pins as outputs:
  //motor1
  pinMode(motor1Pin1, OUTPUT);
  pinMode(motor1Pin2, OUTPUT);
  pinMode(enable1Pin, OUTPUT);
  //motor2
  pinMode(motor2Pin1, OUTPUT);
  pinMode(motor2Pin2, OUTPUT);
  pinMode(enable2Pin, OUTPUT);
  // configure LED PWM functionalitites
  ledcSetup(pwmChannel1, freq1, resolution1);
  ledcSetup(pwmChannel2, freq2, resolution2);
  // attach the channel to the GPIO to be controlled
  ledcAttachPin(enable1Pin, pwmChannel1);
  ledcWrite(pwmChannel1, dutyCycle1);
  //Serial.print("Forward with duty cycle: ");
  Serial.println(dutyCycle1);
  ledcAttachPin(enable2Pin, pwmChannel2);
  ledcWrite(pwmChannel2, dutyCycle2);
  //Serial.print("Forward with duty cycle: ");
  Serial.println(dutyCycle2);
  // testing
  Serial.print("ready to accept espNOW message");
  // motor 1 reset
  motor1_off();

  // Set encoder pins as inputs
  ESP32Encoder::useInternalWeakPullResistors = UP; // Enable the weak pull up
resistors
  encoder.attachHalfQuad(ENC_1A, ENC_1B); // Attache pins for use as encoder pins
  encoder.setCount(0);  // set starting count value after attaching
```

```
  encoder.clearCount ( );
  encoder2.attachHalfQuad(ENC_2A, ENC_2B); // Attache pins for use as encoder pins
  encoder2.setCount(0);  // set starting count value after attaching
  encoder2.clearCount ( );

  timer1 = timerBegin(1, 80, true);  // timer 1, MWDT clock period = 12.5 ns *
TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer1, &onTime1, true); // edge (not level) triggered
  timerAlarmWrite(timer1, 100000, true); // 100000 * 1 us = 100 ms, autoreload true
  // at least enable the timer alarms
  //timerAlarmEnable(timer0); // enable
  timerAlarmEnable(timer1); // enable


}
void loop(){
theta = enc_count1-revs*countsperturn;
counter1 = encoder.getCount( );
//buttonIsPressed = false;

last_state=state;

if (turntable_encoder_full_rev()) {
  turntable_encoder_full_rev_reset();
}

if (turntable_off_course()) {
  turntable_correction();
}

if (desired_turns_reached()==false) {
  latch_correction();
}

//if (lidar_limit_reached()){
//  emergency_shutoff();
//}
switch (state) {
    case 0:
      status="This is state 0, waiting for espnow";
      if(ingredient1==1 || ingredient2==1 || ingredient3==1 || ingredient4==1 ||
ingredient5==1 || ingredient6==1 || ingredient1==2 || ingredient2==2 ||
ingredient3==2 || ingredient4==2 || ingredient5==2 || ingredient6==2){
```

```
        led_on();
        state = 1;
        delay(5000);
      }
      if(commence == 2){
        state=7;
      }
      break;
    case 1:
      status="This is state 1, setup";
      led_on();
      if (ingredient_num <=6) {
        //EVENT 1: if button is pressed OR position error out of tolerance
        //SERVICE 1: switch state 1->2
        thetaDes=(ingredient_num-1)*thetaDes_interval;
        serving_to_go=ingredient_serving();
        state = 2;
      }
      else {
        ingredient_num=1;
        state = 7;
      }
      break;

    case 2:
      status="This is state 2, motor moving";
      motor1_off();
      //plotControlData();

      if (deltaT) {
        portENTER_CRITICAL(&timerMux1);
        deltaT = false;
        portEXIT_CRITICAL(&timerMux1);
        if(abs(theta-thetaDes)>count2_slow_tolerance){
          if(theta < thetaDes){
            omegaDes = 15;
          }
          else if(theta > thetaDes){
            omegaDes = -15;
          }
        }
        else if(abs(theta-thetaDes)<count2_slow_tolerance &&
abs(theta-thetaDes)>count2_stop_tolerance){
```

```
    if(theta < thetaDes){
      omegaDes = int(abs(float(thetaDes-theta))/float(countsperturn)*15) + 10;
    }
    else if(theta > thetaDes){
      omegaDes = -(int(abs(float(thetaDes-theta))/float(countsperturn)*15) +
10);
    }
  }

  else{
    omegaDes = 0;
              //EVENT 2: if postion error is not out of tolerance:
    //SERVICE 2: stop motor and switch case 2 -> 1
    //Serial.print("stopped");
    ledcWrite(pwmChannel2, 0);
    digitalWrite(motor2Pin1, LOW);
    digitalWrite(motor2Pin2, LOW);
    led_off();
    if (interrupted_from_case){
      interrupted_from_case=false;
      state=save_current_state;
    }
    else{
      state=3;
    }
  }

  //PI control on Duty cycle with respect to position
  e= omegaDes- omegaSpeed;
  sumE= sumE+e;
  if (abs(sumE)> KiMax){
    if (sumE>0){
      sumE= KiMax;
    }
    else{
      sumE= -KiMax;
    }
  }
  D= Kp*e+ Ki*sumE;
  //keep Duty cycle within nominal and maximum
  if (D > MAX_PWM_VOLTAGE) {
    D = MAX_PWM_VOLTAGE;
  }
```

```
    else if (D < -MAX_PWM_VOLTAGE) {
      D = -MAX_PWM_VOLTAGE;
    }
    //else if (D >0 && D< NOM_PWM_VOLTAGE) {
      //D = NOM_PWM_VOLTAGE;
    //}
    //else if (D <0 && D> -NOM_PWM_VOLTAGE) {
      //D = -NOM_PWM_VOLTAGE;
    //}

    //Map the D value to motor directionality

    if (D > 0)  {
        //Serial.print("forward");
        ledcWrite(pwmChannel2, D);
        digitalWrite(motor2Pin1, HIGH);
        digitalWrite(motor2Pin2, LOW);
    }
    else if (D < 0) {
      //Serial.print("backward");
      //FLIP ENCODER PINS SO SPEED AND D HAVE SAME SIGN
      ledcWrite(pwmChannel2, -D);
      digitalWrite(motor2Pin1, LOW);
      digitalWrite(motor2Pin2, HIGH);
    }
  }

  break;

case 3:
  status="This is state 3, turntable in position, switching to latch";

  //if turntable out of tolerance
  state=4;
  break;

case 4:
  status="This is state 4, setting up latch";
  if( serving_to_go>0){
    desired_turns1=-10;
    desired_counts1=desired_turns1*countsperturn1;
    state = 5;
  }
```

```
      else {
        ingredient_num=ingredient_num+1;
        state = 1;
      }

      break;

    case 5:
      status="This is state 5, latch moving";
      if(desired_turns_reached()==false){

dutyCycle1=int(float(abs(counter1-desired_counts1))/float(10*countsperturn1)*100)+150
;
        ledcWrite(pwmChannel1, dutyCycle1);
        if (counter1<=desired_counts1 + count1_tolerance){
          motor1_forward();
        }
        else if (counter1>=desired_counts1 - count1_tolerance){
          motor1_backward();
        }
      }
      else if(desired_turns_reached()){
        //EVENT: desired encoder count is reached
        //SERVICE: turn off led, switch case 2-> 3
        if (interrupted_from_case){
            interrupted_from_case_latch=false;
            state=save_current_state_latch;
        }
        else{
          Currenttime=millis();
          transition_time=Currenttime;
          state = 6;
        }
      }
      break;

    case 6:
      status="This is state 6, setting latch setpoint to 0";
      motor1_off();
      desired_turns1=0;
      desired_counts1=desired_turns1*countsperturn1;
      if(desired_turns_reached()==false){
        //EVENT: desired encoder count is not reached
```

```
        //SERVICE: switch case 3-> 2
        state=5;
      }
      else {
        //EVENT: desired encoder count is reached
        //SERVICE: turn off led, switch case 3-> 1
        Currenttime=millis();
        if (Currenttime-transition_time>3000){
        serving_to_go=serving_to_go-1;
        state = 4;
        }
      }
      break;

    case 7:
      led_off();
      thetaDes=6*thetaDes_interval;
      serving_to_go=0;
      status="This is state 7, process end";
      if(ingredient1==1 || ingredient2==1 || ingredient3==1 || ingredient4==1 ||
ingredient5==1 || ingredient6==1 || ingredient1==2 || ingredient2==2 ||
ingredient3==2 || ingredient4==2 || ingredient5==2 || ingredient6==2){
        commence = 1;
        ingredient1 = 0;
        ingredient2 = 0;
        ingredient3 = 0;
        ingredient4 = 0;
        ingredient5 = 0;
        ingredient6 = 0;
        //new = 1;
        incomingReadings.commence = commence;
        //incomingReadings.new = new;
        esp_err_t result = esp_now_send(broadcastAddress, (uint8_t*)
&incomingReadings, sizeof(incomingReadings));
        //Serial.println("Commence sent!");
        led_off();
        state=0;
      }
      break;

    case 8:
      status="This is state 8, emergency shutoff";
      Serial.println(status);
```

```
      //latch motor off
      motor1_off();
      led_on();
      break;
}
//if (change_state()){
  plotControlData();
//}

}

//bool lidar_limit_reached(){
//  return (tofSensor.readRangeSingleMillimeters()<100);
//}
void emergency_shutoff(){
  state=8;
}
bool turntable_encoder_full_rev() {
  if(theta==countsperturn){
    return true;
  }
  else{
    return false;
  }
}
void turntable_encoder_full_rev_reset(){
  thetaDes=thetaDes-countsperturn;
  revs=revs+1;
}

bool CheckForButtonPress() {
  if (buttonIsPressed == true){
    thetaDes=thetaDes+thetaDes_interval;//increase desired postion by the
thetaDes_interval (60 degrees)
    led_on();
    buttonIsPressed = false;
    return true;
  }
  else {
    return false;
  }
}
```

```
bool turntable_off_course(){
  if (abs(thetaDes- theta)>count2_stop_tolerance && (state > 2 || state==0)) {
    return true;
  }
  else{
    return false;
  }
}

void turntable_correction(){
  //SERVICE 1: switch state 1->2
  led_on();
  interrupted_from_case=true;
  save_current_state=state;
  state = 2;
}

bool change_state(){
  if (last_state !=state){
    return true;
   }
  else{
    return false;
   }
}

int condiment_num(){
  if (abs(thetaDes- theta)<count2_stop_tolerance){
    condiment=thetaDes/thetaDes_interval+1;
    if (condiment>6){
      turns=thetaDes/thetaDes_interval/6;
      condiment=condiment-6*turns;
    }
  }
  return condiment;
}

int ingredient_serving(){
  if (ingredient_num==1){
    serving=ingredient1;
  }
  else if (ingredient_num==2){
```

```
    serving=ingredient2;
  }
  else if (ingredient_num==3){
    serving=ingredient3;
  }
  else if (ingredient_num==4){
    serving=ingredient4;
  }
  else if (ingredient_num==5){
    serving=ingredient5;
  }
  else if (ingredient_num==6){
    serving=ingredient6;
  }
  return serving;
}
void plotControlData() {
  Serial.print("|Condiment:");
  Serial.print(condiment_num());
  Serial.print(" ");
  Serial.print("ingredient:");
  Serial.print( ingredient_num);
  Serial.print(" ");
  Serial.print("serving to go:");
  Serial.print( serving_to_go);
  Serial.print(" ");
  Serial.print("|Position:");
  Serial.print(theta);
  Serial.print(" ");
  Serial.print("Desired_Position:");
  Serial.print(thetaDes);
  Serial.print(" ");
  Serial.print("PWM_Duty:");
  Serial.print(D);
  Serial.print(" ");
  Serial.print(" | latch Direction: ");
  Serial.print(currentDir1);
  Serial.print(" ");
  Serial.print("counter1: ");
  Serial.print(counter1);
  Serial.print(" ");
  Serial.print("desired counts: ");
  Serial.print(desired_counts1);
```

```
  Serial.print(" ");
  Serial.print(" | status: ");
  Serial.println(status);
}

bool desired_turns_reached() {
  if (counter1<=desired_counts1 + count1_tolerance && counter1>=desired_counts1 -
count1_tolerance){
    return true;
  }
  else {
    return false;
  }
}
void latch_correction(){
  //SERVICE 1: switch state 1->2
  led_on();
  interrupted_from_case_latch=true;
  save_current_state_latch=state;
  state = 5;
}
void motor1_forward() {
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, HIGH);
  currentDir1="forward";
}
void motor1_backward() {
  digitalWrite(motor1Pin1, HIGH);
  digitalWrite(motor1Pin2, LOW);
  currentDir1="backward";
}
void motor1_off() {
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, LOW);
  currentDir1="stopped";
}


void led_on() {
  digitalWrite(LED_PIN, HIGH);
}

void led_off() {
```

```
  digitalWrite(LED_PIN, LOW);
}
```