Automated Cling Wrap Dispenser

Group 30: Sam Averitt, Tae Lee, Yuke Nakano, Todd Russell

University of California, Berkeley

Final Deliverables

Mechatronics Design

ME 102B

11 December 2022

I Opportunity

We wanted to create a device to aid the user in food preparation processes. Specifically, we were focused on automating the process of dispensing cling wrap. Our motivation in selecting this particular solution was to make the process of storing food both quicker and less frustrating (due to the sticky nature of cling wrap).

II High Level Strategy

2.1 *Strategy of Realized Product*

1. The user loads the main arm with cling wrap and places the object they want wrapped onto the center of the device. This placement is detected by a load cell, causing the system to enter an armed state.
2. While armed, the user is able to press a button that will cause an arm to rotate, covering the desired object in cling wrap. This wrapping process is then followed by an automated cutting process.
3. The user can then remove their wrapped object from the device. This removal is also detected by the load cell, causing the system to enter a secondary armed state.
4. While armed, the user is able to press the button again to reset the system to its original state.

2.2 *Initial Desired Functionality vs. Achieved Specifications*

We initially planned on using a timer to start the wrapping process *x* seconds after placement on the load cell was detected. However, we decided to use a button in order to make our product more adaptable to different situations. We were able to achieve all our other essential goals from our initial desired functionality: the main lever arm is driven by a single motor and has sufficient torque to pull/apply cling wrap onto the desired object. Moreover, we expanded the scope of our project to include the automation of the cutting process of the cling wrap (something that we initially categorized as a reach goal).

III Integrated Physical System

3.1 *Photos*

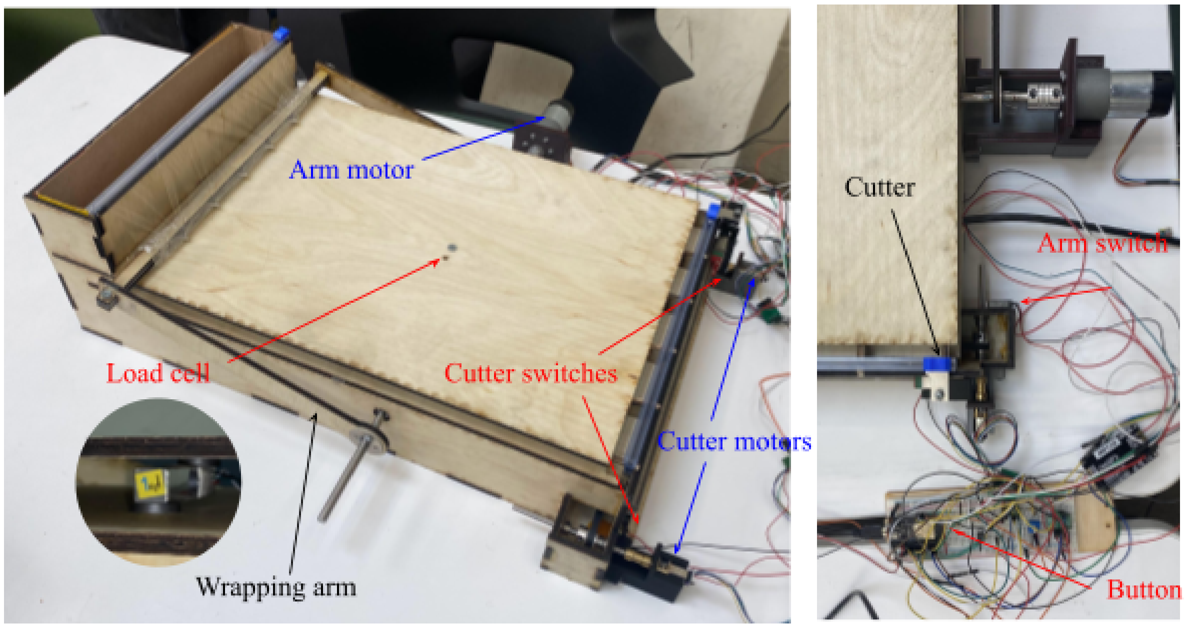


Figure 1 - Integrated physical device[1]

3.2 *Function Critical Decisions*

Our calculations were performed using the following assumptions:

1. 10 N of force is required to pull the cling wrap (this includes a built in buffer).
2. The moment of inertia of our arm is 20 $lb \cdot in^2 = 0.0059\ kg \cdot m^2$ (determined from SolidWorks).
3. Our arm should be able to withstand an angular acceleration of up to 10 rad/s^2 (also includes a built in buffer).

[1] Actuators shown in blue, sensors shown in red, moving parts shown in black

Motor 1

Torque to pull cling wrap - $\tau_p = r \times F = (0.267m)(10N) = 2.67Nm$

Torque to rotate arm - $\tau_i = I\alpha = \left(0.0059kg \cdot m^2\right)\left(10rad/s^2\right) = 0.059Nm$

Torque required of motor - $\tau_m = \tau_p + \tau_i = 2.67Nm + 0.059Nm = 2.729Nm$

The motor we selected has a stall torque of 49 $kg \cdot cm$ = 4.805 $N \cdot m$ when operating at 12 V. Following the rule of thumb of not exceeding 60% of the stall torque, the maximum torque that this motor can safely supply is $4.805Nm * 0.6 = 2.88Nm > 2.729Nm$.

Arm bearings

Our initial design was based around a weaker motor and the utilization of a gear ratio. The calculations we performed gave a maximum radial load of approximately 201.5 N. The majority of this radial load was due to the use of gears; the radial load exerted on the bearings by the arm itself was relatively small. After performing these calculations, we changed our design to use a stronger motor and a flexible shaft coupler, eliminating the use of gears and substantially reducing the radial load experienced by the bearings. Although we ensured our bearings were rated for radial loads within the range of our initial calculations, this design change allowed us to prioritize the geometry (8 mm ID) and cost reduction of the bearings over their radial load specifications.

Motor 2/flexible transmission

Because this was a low force/torque application, we prioritized prototyping and an iterative design approach over performing torque/radial load calculations. After running tests with our initial design, we realized that in order to consistently cut the cling wrap we would need to exceed the stall torque of the motor in our lab kit. To resolve this issue, we drove the pulley system with 2 lab kit motors (one on either end), both of which were connected to the same logic pin on the ESP32. This was able to provide a sufficient cutting force for our application. Another substantial design modification that we had to make was the inclusion of a rail to minimize the torque experienced by the cutting arm itself.

IV Software/Subsystems

4.1 Circuit Diagram

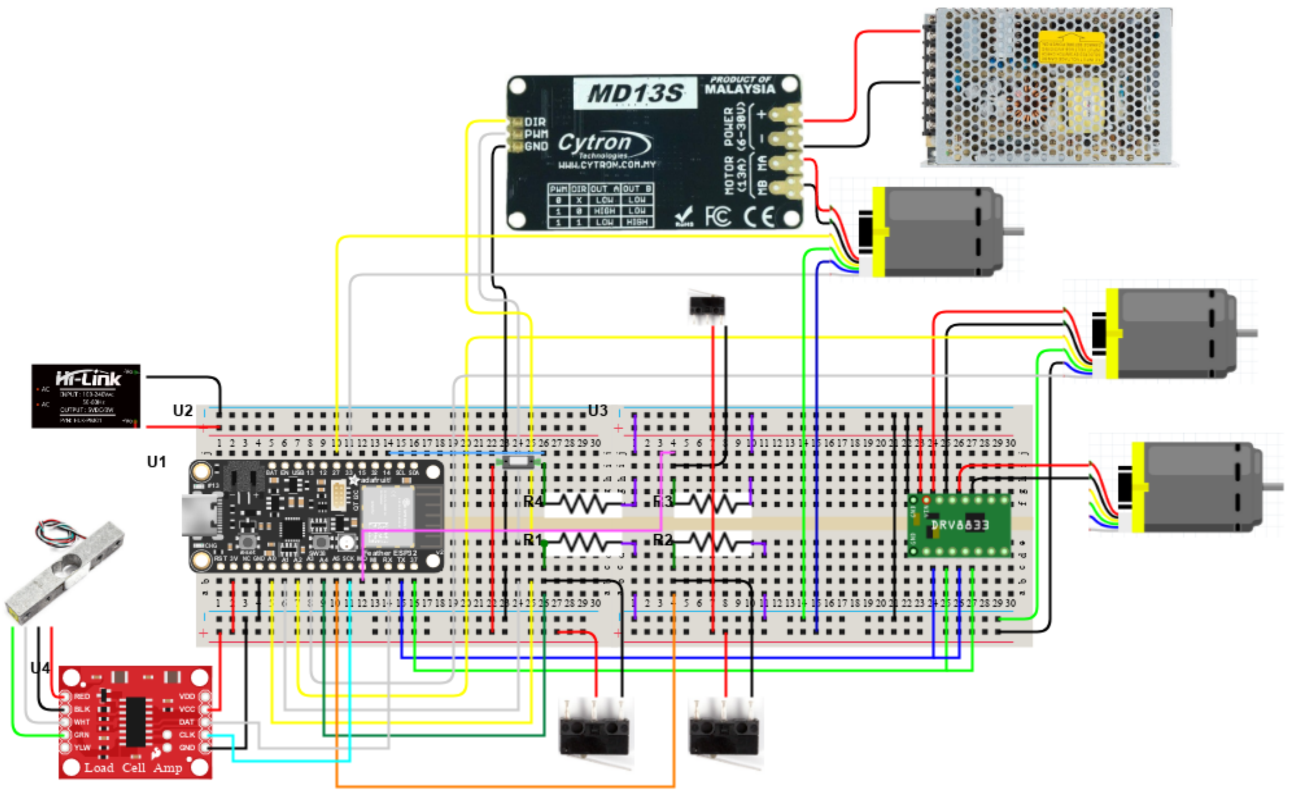
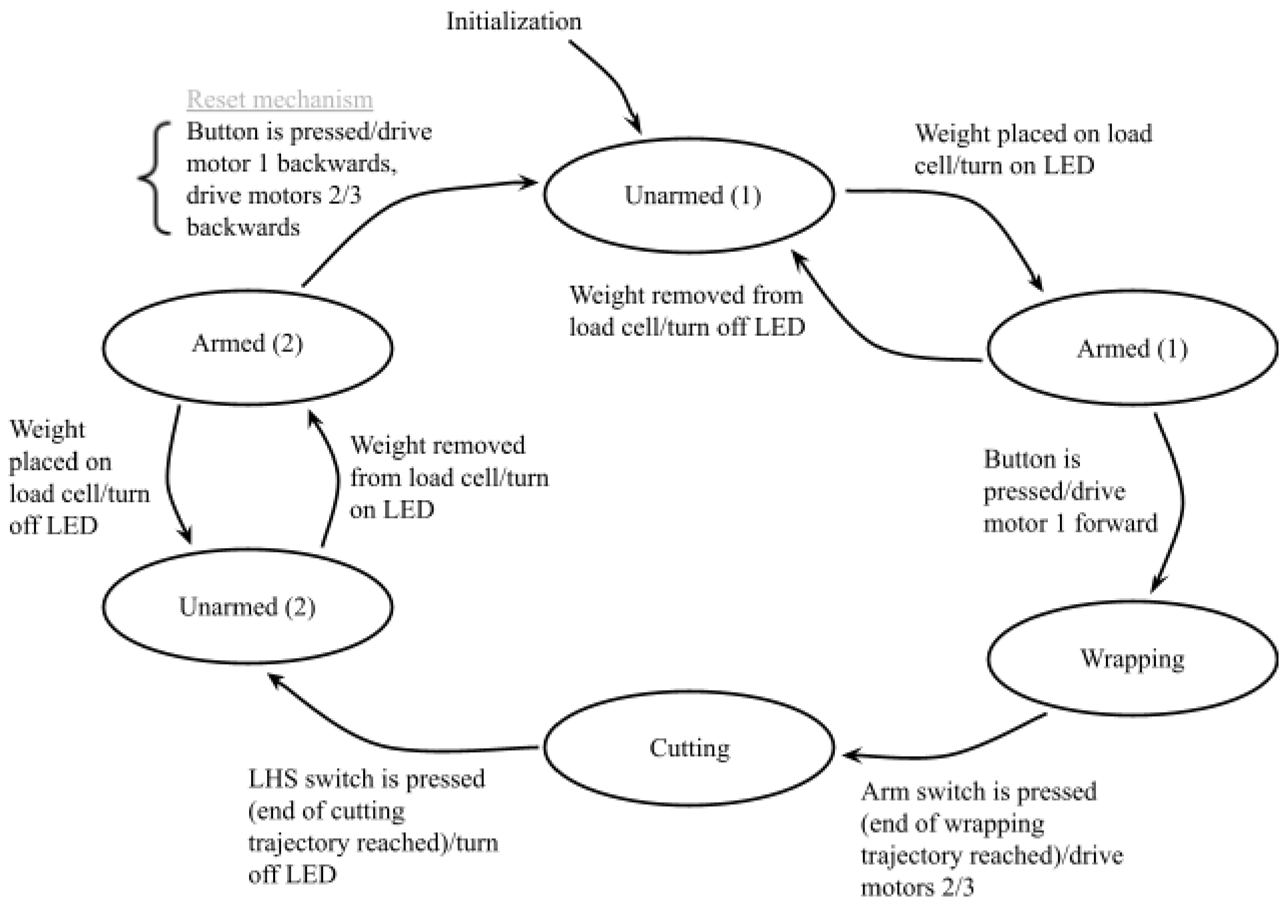


Figure 2 - Circuit diagram

4.2 State Transition Diagram



Initialization
Reset mechanism
Button is pressed/drive motor 1 backwards, drive motors 2/3 backwards
Unarmed (1)
Weight placed on load cell/turn on LED
Weight removed from load cell/turn off LED
Armed (2)
Armed (1)
Weight placed on load cell/turn off LED
Weight removed from load cell/turn on LED
Button is pressed/drive motor 1 forward
Unarmed (2)
Wrapping
Cutting
LHS switch is pressed (end of cutting trajectory reached)/turn off LED
Arm switch is pressed (end of wrapping trajectory reached)/drive motors 2/3

Figure 3- State transition diagram[2]

V Reflections

5.1 Strategies that worked well

Going to office hours and the machine shop to ask questions, especially during the designing stages of our project, were both incredibly helpful resources. We were able to get really good feedback on how to improve our design while still keeping it manufacturable. Furthermore, whenever a set of deliverables was due, our group met consistently throughout the week preceding the deadline. This worked well in that it gave us sufficient time to address/debug any issues that arose.

5.2 What we wish we had done differently

Starting the manufacturing/prototyping process earlier in the semester would have been really helpful. We ran into a lot of unexpected obstacles in the week before functionality evaluations and had to rush implementing some last minute modifications. These obstacles could have been found and addressed sooner/more thoroughly had we started prototyping earlier in the semester.

[2] Note that the wrapping/cutting states in the diagram are implemented with if statements rather than switch cases in the code shown in the Appendix below.

VI Appendix

6.1 BOM

Part	Description	Link/supplier	Cost	Quantity	Total
1/4" plywood (18" x 30")	Laser cut for base, arm, cutter, and transmission (gear) assembly	Jacobs	$6.25	6	$37.50
Load cell	Analog input	Lab kit	$0.00	1	$0.00
Pushbutton switch	Digital input	Lab kit	$0.00	1	$0.00
Spacers	For load cell deflection	3D print	$5.00 (for filament)		$5.00
Motor housings	To hold motors in place	3D print			
Cutters	2 on either end to cut cling wrap	Amazon	$10.99	1 (pack of 4)	$12.12
8mm keyed shaft (500mm)	To rotate U shaped arm	McMaster	$30.92	1	$30.92
2mm key for shaft	To secure U shaped arm to shaft	McMaster	$9.61	1 (pack of 5)	$9.61
Motor 1	Rotate keyed shaft/lever arm	Pololu	$51.95	1	$51.95
Motor 2	To rotate pulley/flexible transmission	Lab kit	$0.00	1	$0.00
L298N motor driver	To drive motor	Borrowed (Tom)	$0.00	1	$0.00
12V power supply	External power supply for motor	Borrowed (Tom)	$0.00	1	$0.00
Ball Bearings	To support 8mm shaft	McMaster	$8.52	2	$17.04
Shims	Prevents inner/outer races from touching	McMaster	$9.17	1 (pack of 10)	$9.17
Belleville washers	To preload ball bearings	McMaster	$4.00	1 (pack of 10)	$4.00
Collar	Provides axial constraint on 8mm shaft for bearings	McMaster	$5.45	2	$10.90
Flexible shaft coupler	Connect motor shaft to shaft w/ input gear	Amazon	$9.99	1	$9.99
Pulley kit	Main flexible transmission	Amazon	$16.99	1	$16.99
5mm dia shaft	To hold rotational components	Amazon	$9.59	1 (pack of 5)	$9.59
Shaft coupler	To support two axels of varying sizes	Amazon	$8.99	1 (pack of 8)	$8.99
Shaft Collar	To hold the shims and washers in place	McMaster	$1.83	4	$7.32
Flanged Ball Bearing	To assist rotational motion	Amazon	$9.99	1 (pack of 10)	$9.99
Railing	To provide support to cutting arm	Amazon	$17.64	1	$19.64
Hardware (M3 nuts, screws, etc.)	Secure components together (more in depth summary here)	ACE	-	-	$24.75
Wood glue	To secure plywood pieces together	ACE	$7.71	1	$7.71
Stainless Steel Shafts	100mmx5mm Stainless Steel Round Shafts	McMaster	9.59	1	$9.59

Table 1 - Bill of Materials

6.2 CAD

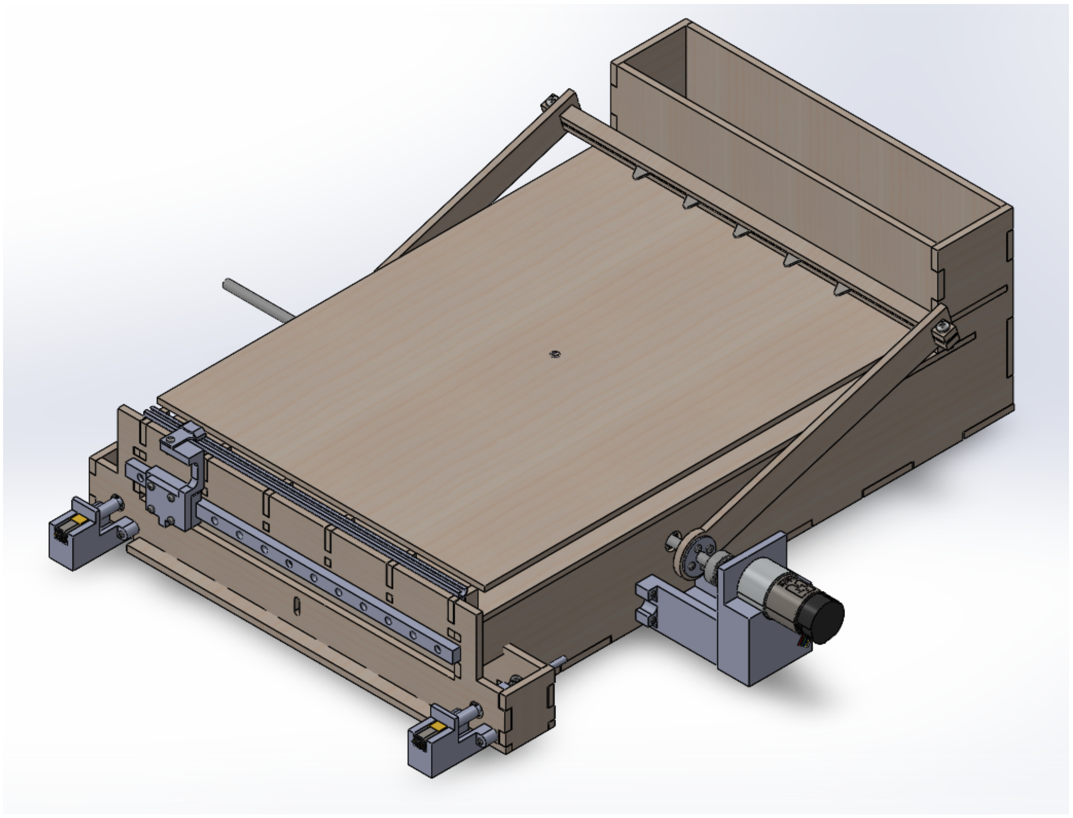


Figure 4 - Full CAD, isometric view

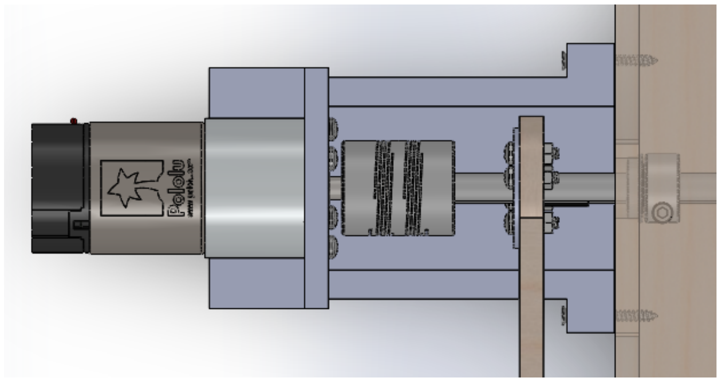


Figure 5 - Arm transmission

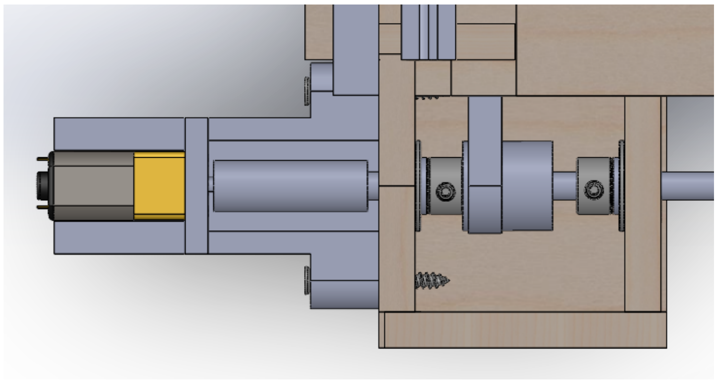


Figure 6 - Pulley transmission (mirrored on either side)

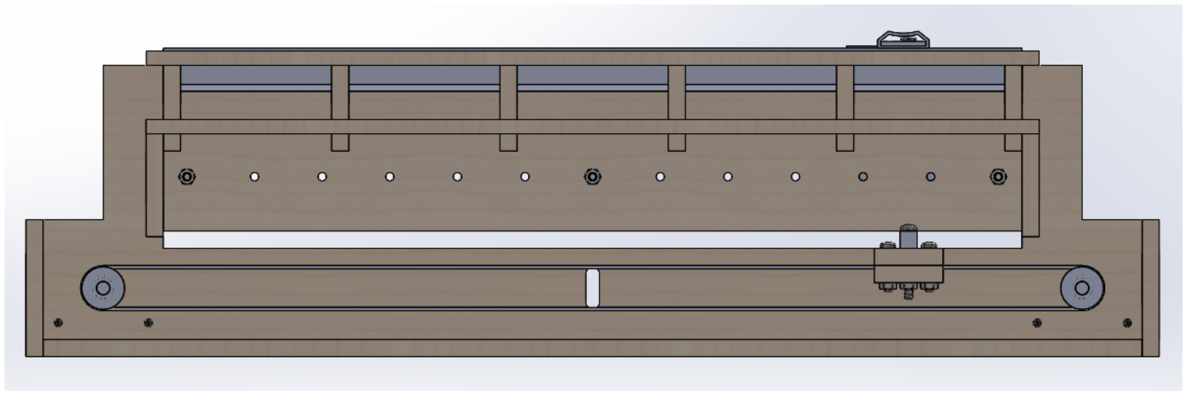


Figure 7 - Pulley cross section view

6.3 Code

```
#include <Arduino.h>
#include <ESP32Encoder.h>
#include "HX711.h"

// set up pins
#define LED_PIN 13 // builtin LED pin number
#define BTN 14  // declare the button ED pin number
#define DIR 26 // declare the direction pin number (motor1)
#define PWM 25 // declare the PWM pin number (motor1)
#define BIN_1 17 // declare motor2 values
#define BIN_2 21 // declare motor2 values
#define SWR 4 // declare RHS switch
#define SWL 18 // declare LHS switch
#define SWA 36 // declare arm switch

//Define status
#define CASE 1
#define CASE 2
#define CASE 3
#define CASE 4
#define CASE 5

#define calibration_factor -7050.0 // for load cell

#define DOUT 16
#define CLK 5

HX711 scale;


// Setup variables ------------------------------------------------------------
// Values for arm rotation angle and total time (CHANGE AS NEEDED)
float armRotation = 165; // Degrees (120?)
int totalTimeArm = 5; // Seconds (5?)
int slicerRotation = 250; // Units (250)
int totalTimeSlicer = 15; // Seconds (4?)

// Values for controller performance (CHANGE AS NEEDED)
float KpA = 1;
float KiA = .5;
float KpS = 1;
float KiS = .1;
volatile int omegaMax = 10;
volatile int D_max = 175;
int totalError = 10;

// Value for load cell threshold (CHANGE AS NEEDED)
const int threshold = 20;
const int pwmChannel = 0;
int val = 0;
```

```
// Values not to change
int thetaArm = 0;
int thetaSlicer = 0;
float thetaDesArm = 0;
int thetaDesSlicer = 0;
int thetaMaxArm = 4550;
int thetaMaxSlicer = 4550;
int D = 0;
volatile int DarmSpeed = 0;
volatile int DarmPosition = 0;
int DSlicer = 0;
int thetaFinalArm = 0;
int thetaFinalSlicer = 0;

volatile bool switch_state_slicer_RHS = false;
volatile bool switch_state_slicer_LHS = false;
volatile bool switch_state_arm = false;

int state = 1;

// Speed control
int omegaSpeed = 0;
int dir = 1;
int KpAS = 60;   // TUNE THESE VALUES TO CHANGE CONTROLLER PERFORMANCE
int KiAS = 5;
int IMax = 0;
float sumError = 0;

// Set slicer motor values
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;

// Encoder interrupt variables
volatile int countArm = 0; // encoder count Arm
volatile int countSlicer = 0; // encoder count Slicer
volatile bool interruptCounter = false; // check timer interrupt 1
volatile bool deltaT = false;     // check timer interrupt 2
int totalInterrupts = 0;   // counts the number of triggering of the alarm
hw_timer_t * timer0 = NULL;
hw_timer_t * timer1 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;

const int MAX_PWM_VOLTAGE = 255;
const int NOM_PWM_VOLTAGE_ARM = 100;
const int NOM_PWM_VOLTAGE_SLICER = 150;

// Button interrupt variables
volatile bool buttonIsPressed = false;
int timepassed = 0;

hw_timer_t * timer = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;
```

```
// Driver setcup code -------------------------------------------------------
// From https://github.com/CytronTechnologies/CytronMotorDriver
enum MODE {
  PWM_DIR,
  PWM_PWM,
};

class CytronMD
{
  public:
    CytronMD(MODE mode, uint8_t pin1, uint8_t pin2);
    void setSpeed(int16_t speed);

  protected:
    MODE _mode;
    uint8_t _pin1;
    uint8_t _pin2;
};

CytronMD::CytronMD(MODE mode, uint8_t pin1, uint8_t pin2)
{
  _mode = mode;
  _pin1 = pin1;
  _pin2 = pin2;

  pinMode(_pin1, OUTPUT);
  pinMode(_pin2, OUTPUT);

  digitalWrite(_pin1, LOW);
  digitalWrite(_pin2, LOW);
}
```

```
void CytronMD::setSpeed(int16_t speed)
{
  // Make sure the speed is within the limit.
  if (speed > 255) {
    speed = 255;
  } else if (speed < -255) {
    speed = -255;
  }

  // Set the speed and direction.
  switch (_mode) {
    case PWM_DIR:
      if (speed >= 0) {
        analogWrite(_pin1, speed);
        digitalWrite(_pin2, LOW);
      } else {
        analogWrite(_pin1, -speed);
        digitalWrite(_pin2, HIGH);
      }
      break;

    case PWM_PWM:
      if (speed >= 0) {
        analogWrite(_pin1, speed);
        analogWrite(_pin2, 0);
      } else {
        analogWrite(_pin1, 0);
        analogWrite(_pin2, -speed);
      }
      break;
  }
// Configure the motor driver --------------------------------------------------
CytronMD motor(PWM_DIR, PWM, DIR);

ESP32Encoder encoderArm;
ESP32Encoder encoderSlicer;

// Initialization --------------------------------------------------------------
void IRAM_ATTR isr() {  // the function to be called when interrupt is triggered
  buttonIsPressed = true;
}

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

void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux1);
  countArm = encoderArm.getCount( );
  countSlicer = encoderSlicer.getCount( );
  encoderArm.clearCount ( );
  encoderSlicer.clearCount ( );
  deltaT = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux1);
}
```

```
// Initialize Timer
void TimerInterruptInit() {
  timer = timerBegin(0, 80, true); // divides the frequency by the prescaler: 80,000,000 / 80 = 1,000,000 tics / sec
}

// Setup code (runs once) -------------------------------------------------------
void setup() {
  // setup switches
  pinMode(SWR, INPUT); // configure RHS switch as input
  pinMode(SWL, INPUT); // configure LHS switch as input
  pinMode(SWA, INPUT); // configure arm switch as input

  // configure LED PWM functionalitites
  ledcSetup(ledChannel_1, freq, resolution);
  ledcSetup(ledChannel_2, freq, resolution);

  // attach the channel to the GPIO to be controlled
  ledcAttachPin(BIN_1, ledChannel_1);
  ledcAttachPin(BIN_2, ledChannel_2);

  // Timer setup
  pinMode(BTN, INPUT);  // configures the specified pin to behave either as an input or an output
  attachInterrupt(BTN, isr, RISING);
  Serial.begin(115200);
  TimerInterruptInit(); // Initiates timer interrupt

 // Encoder and motor setup
 Serial.begin(115200);
 ESP32Encoder::useInternalWeakPullResistors = UP; // Enable the weak pull up resistors
 encoderArm.attachHalfQuad(33, 27); // Attach pins for use as encoder pins
 encoderSlicer.attachHalfQuad(39, 34); // Attach pins for use as encoder pins
 encoderArm.setCount(0);  // set starting count value after attaching
 encoderSlicer.setCount(0);  // set starting count value after attaching

 // initilize timer
 timer0 = timerBegin(0, 80, true);  // timer 0, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE
 timerAttachInterrupt(timer0, &onTime0, true); // edge (not level) triggered
 timerAlarmWrite(timer0, 20000000, true); // 2000000 * 1 us = 2 s, autoreload true

 timer1 = timerBegin(1, 80, true);  // timer 1, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE
 timerAttachInterrupt(timer1, &onTime1, true); // edge (not level) triggered
 timerAlarmWrite(timer1, 10000, true); // 10000 * 1 us = 10 ms, autoreload true

 // at least enable the timer alarms
 timerAlarmEnable(timer0); // enable
 timerAlarmEnable(timer1); // enable

 // Load cell setup
 pinMode(LED_PIN, OUTPUT);
 pinMode(DOUT, INPUT);
 ledcSetup(pwmChannel, freq, resolution);
 ledcAttachPin(DOUT, pwmChannel);
```

```
  Serial.begin(115200);
  scale.begin(DOUT,CLK);
  scale.set_scale(calibration_factor); // value is obtained by using the SparkFun_HX711_Calibration sketch
  scale.tare();  // assuming no weight on t/zhe scale at start up, reset the scale to 0

  // Reset timer
  timerStop(timer);
  timerWrite(timer, 0);
  timerStart(timer);

  // Move slicer to neutral position
  switch_state_slicer_RHS = digitalRead(SWR);
  if (switch_state_slicer_RHS == false) { // slicer not in neutral position
    while (switch_state_slicer_RHS == false & timerReadSeconds(timer) < totalTimeSlicer) {
      RotateSlicerCW();
    }
  }

  // Turn motor off
  ledcWrite(ledChannel_1, LOW);
  ledcWrite(ledChannel_2, LOW);
  thetaDesSlicer = 0;

  // Reset timer
  timerStop(timer);
  timerWrite(timer, 0);
  timerStart(timer);

  // Reset button
  buttonIsPressed = false;
}
```

```
// Main code (runs repeatedly)-----------------------------------------------
void loop() {
  Serial.println(state);

  // get values from load cell
  val = scale.get_units(); // returns a float
  Serial.print("Reading: ");
  Serial.print(scale.get_units(), 1); //scale.get_units() returns a float
  Serial.print(" grams"); //You can change this to kg but you'll need to refactor the calibration_factor
  Serial.println();

  switch (state) {
    case 1: // unarmed(1)
    if (CheckWeight() == true) {
      led_on();
      buttonIsPressed = false;
      state = 2;
    }
    break;

    case 2: // armed(1)
    if (CheckWeight() == false) {
      led_off();
      state = 1;
    }
    else if (CheckForButtonPress() == true) {
      ButtonResponse();
      state = 3;
    }
    break;

   case 3: // wrapping and cutting
   switch_state_slicer_LHS = digitalRead(SWL);
   if (switch_state_slicer_LHS == true) { // end of wrapping trajectory
     led_off();
     state = 4;
   }
   break;

   case 4: // unarmed(2)
   if (CheckWeight() == false) {
     buttonIsPressed = false;
     led_on();
     state = 5;
   }
   break;

   case 5: // armed(2)
   if (CheckWeight() == true) {
     led_off();
     state = 4;
   }
   else if (CheckForButtonPress() == true) {
     ButtonResponse();
     led_off();
     state = 1;
   }
   break;
  }
}
```

```
// Event checkers -----------------------------------------------------------
bool CheckForButtonPress() {
  if (buttonIsPressed == true & timerReadMilis(timer) > 1000) {
    return true;
  }
  else {
    return false;
  }
}

bool CheckWeight() { // checking for distance event
  if (val >= threshold){
    return true;
  }
  else if (val < threshold){
    return false;
  }
}

// Services -----------------------------------------------------------------
// turn led on/off to indicate armed/unarmed respectively
void led_on() {
  digitalWrite(LED_PIN, HIGH);
}

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



// calls RotateArm and RotateSlice in response to button event
void ButtonResponse() {
  buttonIsPressed = false;
  timerStop(timer);
  timerWrite(timer, 0);
  timerStart(timer);
  thetaArm += countArm;

  switch_state_arm = digitalRead(SWA);

  // Determine current position of arm and set desired end position
  if (switch_state_arm == false and thetaArm < 0.5 * 4550 / 360 * armRotation) {
    thetaFinalArm = 4550 / 360 * armRotation;
  }
  else {
    thetaFinalArm = 5;
  }
```

```
// Calculate desired angle for arm
thetaDesArm = map(thetaFinalArm, 0, 4095, 0, thetaMaxArm);

// Rotate Arm
while (timerReadSeconds(timer) < totalTimeArm) {
  RotateArm();
}

// Turn off motor
motor.setSpeed(0);
thetaDesArm = 0;

// Reset timer
timerStop(timer);
timerWrite(timer, 0);
timerStart(timer);

thetaSlicer += countSlicer;

switch_state_slicer_RHS = digitalRead(SWR);
switch_state_slicer_LHS = digitalRead(SWL);

// Determine current position of slicer, set desired end position, and move slicer
if (switch_state_slicer_RHS == true) { // slicer in neutral position
  thetaFinalSlicer = 4550 / 360 * slicerRotation;
  // Calculate desired angle for arm
  thetaDesSlicer = map(thetaFinalSlicer, 0, 4095, 0, thetaMaxSlicer);


  // Move slicer to extended position if arm is in correct position
  //if (switch_state_arm == true) {
  while (switch_state_slicer_LHS == false & timerReadSeconds(timer) < totalTimeSlicer) {
    RotateSlicerCCW();
  //  }
  }
}

else if (switch_state_slicer_LHS == true) { // slicer extended
  thetaFinalSlicer = 0;
  // Calculate desired angle for arm
  thetaDesSlicer = map(thetaFinalSlicer, 0, 4095, 0, thetaMaxSlicer);

  // Move slicer to neutral position
  while (switch_state_slicer_RHS == false & timerReadSeconds(timer) < totalTimeSlicer) {
    RotateSlicerCW();
  }
}

else { // slicer in middle
  thetaFinalSlicer = 0;
  // Calculate desired angle for arm
  thetaDesSlicer = map(thetaFinalSlicer, 0, 4095, 0, thetaMaxSlicer);

  // Move slicer to neutral position
  while (switch_state_slicer_RHS == false & timerReadSeconds(timer) < totalTimeSlicer) {
    RotateSlicerCW();
  }
}
```

```
  // Turn motor off
  ledcWrite(ledChannel_1, LOW);
  ledcWrite(ledChannel_2, LOW);
  thetaDesSlicer = 0;

  // Reset timer
  timerStop(timer);
  timerWrite(timer, 0);
  timerStart(timer);

  // Reset button
  buttonIsPressed = false;
}

void RotateArm() {
  if (deltaT) {
      portENTER_CRITICAL(&timerMux1);
      deltaT = false;
      portEXIT_CRITICAL(&timerMux1);

      thetaArm += countArm;

      // Check switches
      switch_state_arm = digitalRead(SWA);
      if (thetaDesArm * 36 / 455 > 0.5 * armRotation & switch_state_arm == true) {
        D = 0;
      }
      else {
        float e = (thetaDesArm - thetaArm) / 10;

        if (e > 50 / KpA) {
          e = 50 / KpA;
        }
        sumError += e;
        if (sumError > 1000) {
          sumError = 1000;
        }
        if (sumError < -1000) {
          sumError = -1000;
        }

        D = KpA * e + float(KiA) / float(10) * float(sumError);
      }
      //Ensure that you don't go past the maximum nominal command
      if (D > NOM_PWM_VOLTAGE_ARM) {
          D = NOM_PWM_VOLTAGE_ARM;
      }
      else if (D < -NOM_PWM_VOLTAGE_ARM) {
          D = -NOM_PWM_VOLTAGE_ARM;
      }

      // Avoid low PWM values that don't actually move the arm
      if (abs(D) < 15) {
        D = 0;
      }

      //Map the D value to motor directionality
      motor.setSpeed(D);

      plotControlDataArm();
  }
}
```

```
void RotateSlicerCCW() {
  if (deltaT) {
      portENTER_CRITICAL(&timerMux1);
      deltaT = false;
      portEXIT_CRITICAL(&timerMux1);

      // check switches
      switch_state_slicer_RHS = digitalRead(SWR);
      switch_state_slicer_LHS = digitalRead(SWL);

      // Put power to motor
      ledcWrite(ledChannel_1, LOW);
      ledcWrite(ledChannel_2, D_max);

      // Plot
      plotSwitches();
  }
}

void RotateSlicerCW() {
  if (deltaT) {
      portENTER_CRITICAL(&timerMux1);
      deltaT = false;
      portEXIT_CRITICAL(&timerMux1);

      // check switches
      switch_state_slicer_RHS = digitalRead(SWR);
      switch_state_slicer_LHS = digitalRead(SWL);

      // Put power to motor
      ledcWrite(ledChannel_1, D_max);
      ledcWrite(ledChannel_2, LOW);

      // Plot
      plotSwitches();
  }
}

void plotControlDataSlicer() {
  Serial.println("Position_Slicer[deg], Desired_Position_Slicer[deg], PWM_Duty_Slicer");
  Serial.print(thetaSlicer * 36 / 455);
  Serial.print(" ");
  Serial.print(thetaDesSlicer * 36 / 455);
  Serial.print(" ");
  Serial.println(DSlicer);
}

void plotControlDataArm() {
  Serial.println("Position_Arm[deg], Desired_Position_Arm[deg], PWM_Duty_Arm");
  Serial.print(thetaArm * 36 / 455);
  Serial.print(" ");
  Serial.print(thetaDesArm * 36 / 455);
  Serial.print(" ");
  Serial.println(D);
  //Serial.print(" ");
  //Serial.println(timerReadMilis(timer))/100;
}

void plotSwitches() {
  switch_state_slicer_RHS = digitalRead(SWR);
  switch_state_slicer_LHS = digitalRead(SWL);
  Serial.println("RHS, LHS");
  Serial.print(switch_state_slicer_RHS);
  Serial.print(" ");
  Serial.println(switch_state_slicer_LHS);
}
```