Arrakis - The Auto Spice Dispenser **Group 11**

Jacob Gottesman, Alvin So, Nikhil Gupta, Amy Jiang

Opportunity:

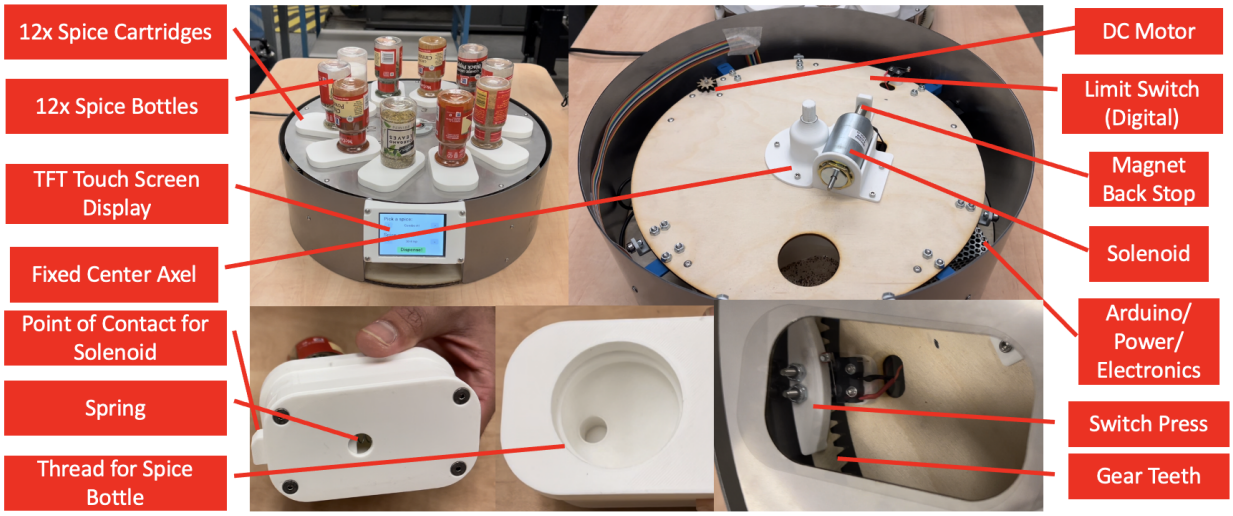
Overall, cooking is a basic need for everyone and for those who have tried, it's common to find that your work space gets messy quickly throughout the process. Some will find themselves missing spices half way through the cooking process while others will forget the recipe. The worst scenario is overseasoning your dishes, spilling your spices or having the incorrect spice ratios. As you can see, many things can go wrong during the process. Therefore, our goal is to make the process more enjoyable by simplifying the process and reducing the probability of making a mistake. The customer base we are targeting includes almost everyone ranging from beginners such as college students to professionals in restaurants. For beginners, mistakes are common and having automated tools can help prevent them from getting overwhelmed while making the process more fun. On the other hand, automated tools help professionals with efficiency by allowing food to be served faster and preventing mistakes from occurring when under pressure with non stop orders during rush hours.

Strategy:

Our strategy for this opportunity is to design a spice dispenser that will create mixtures based on volumetric values of the ingredients in the recipe. Initially, we wanted the user to add their input and save spice ratios via a potentiometer and button setup paired with a display but instead we decided to use a touchscreen for user convenience. To store the spices, we planned to use transparent, refillable spice cartridges with twistable lids. Considering that the user may want to change the spices on the machine without emptying the spice cartridges, we implemented threaded housings that match common spice bottles instead of reusable spice cartridges giving the advantage of easy replacement or refilling of spices. There will be multiple spice housings spaced evenly around a circular base. In order to zero the position at the same spot when starting the machine, we used a limit switch that clicks at a specific position, acting as the zero point.

To dispense, the base will rotate and stop when the desired spice cartridge is above the tray where the cup is. Then, the spice will be dispensed using a solenoid instead of having an archimedes screw and a motor at each spice cartridge which was our initial costly plan. Each cycle of the solenoid will dispense a fixed amount of spice by pushing a compartment in the housing that contains a small increment of spice. The compartment will slide out and drop the spice into the cup. Then, a spring attached to the compartment will push the compartment back into place. The solenoid will conduct x number of cycles to reach the required amount of spice. The number of cycles will equal to the amount of spice divided by the volumetric increments of spice which is the size of the housing compartment.

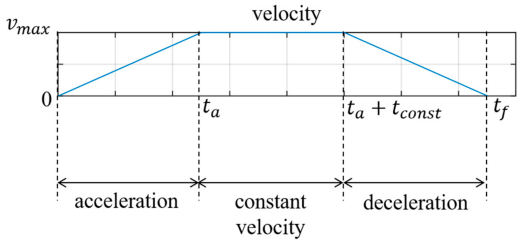
Fully Assembled Device:



Functional Critical Decisions:

<u>Motor Sizing & Math:</u>

The goal used to select the particular DC motor for the project was that we wanted the dispenser to be able to do a full rotation in 4 seconds. Following a trapezoidal velocity profile and equal acceleration and deceleration times t_1 with constant velocity time t_2, the equations dictating the acceleration and maximum velocity are:



$$2\pi = \int_0^4 \omega \, dt \quad \rightarrow \quad 2\pi = t_1 * \omega_{max} + t_2 * \omega_{max}$$

$$t_2 = t - 2t_1 = 4 - 2t_1 \rightarrow \omega_{max} = \alpha * t_1$$

$$2\pi = \alpha t_1^2 + (4 - 2t_1)\alpha t_1 = \alpha t_1^2 - 2\alpha t_1^2 + 4\alpha t_1 = -\alpha t_1^2 + 4\alpha t_1 \rightarrow \alpha = 2\pi/(-t_1^2 + 4t_1)$$

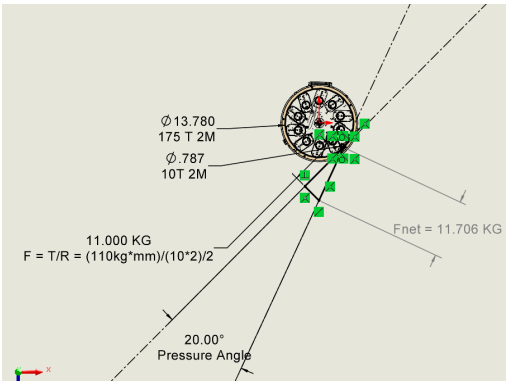
We can now generate a set of values dependent of t_1 knowing that

$$\tau = I\alpha \text{ with I = 104.02 lbs*in}^2 = 0.0304403767 \text{ kg*m}^2 \text{ (From CAD)}$$

$$P = \tau\omega$$

T1	Accel (rad/s*s)	V max (rad/s)	V Max (RPM)	V Max (RPM) of Motor	Torque (kg*m)	Torque (kg*cm)	Torque After Gear Ratio of 17.5:1	Fudge Factor of 1.5
0.2	7.48	1.50	14.29	250.00	0.23	22.77	1.30	1.95

The Pololu 25D 34:1 Metal Gearmotor is capable of of producing a the torque and speed requirements with a 17.5:1 reduction for the t_1 of 0.2 seconds.

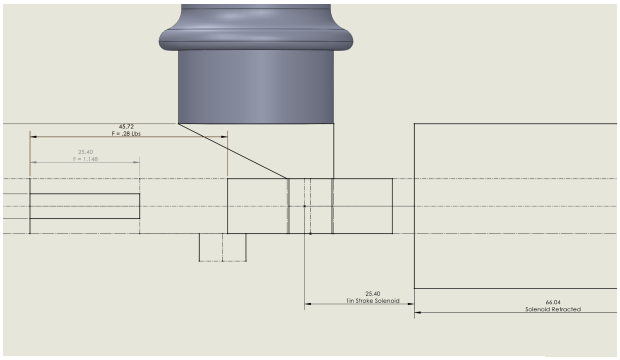


<u>Gear Tooth / Bearing Load Calculations</u>

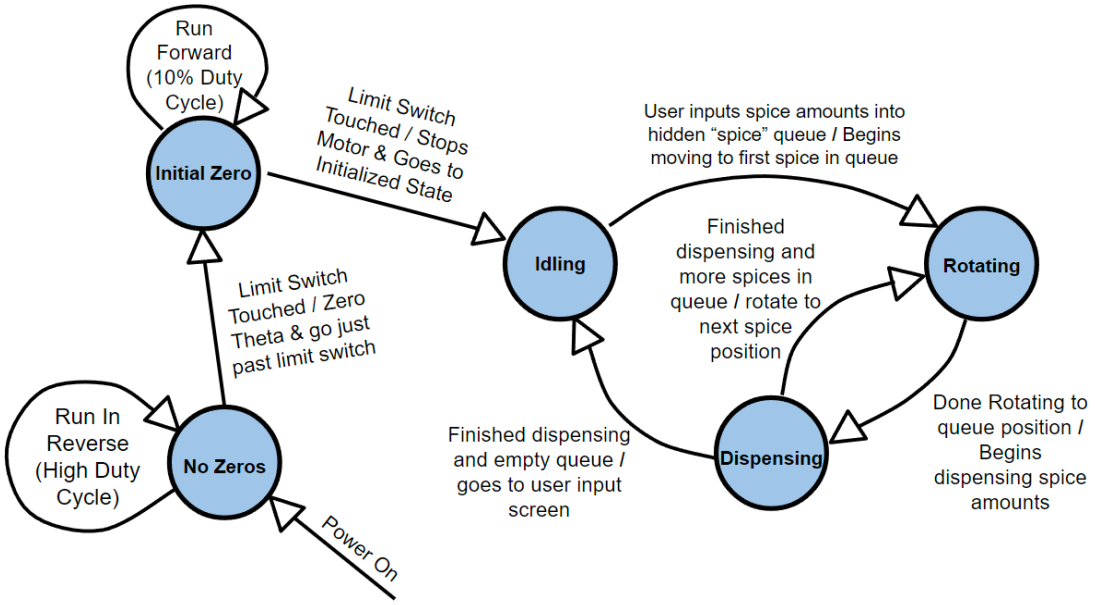
For a brief moment, our motor will be stalled (aka 0 RPM) before it begins accelerating. This would apply a torque of 110 kg*mm to the motor. Given a 20° pressure angle with a pinion gear of 10T and an internal spur gear of 175T. The net force on the teeth of the gears is 11.706 kg. This would create a reaction/radial force of 11.706 kg at the bearing which is under its maximum dynamic load of 140 KG and even a static load of 32 kg. No max radial was provided by the manufacturer for our motor.

Solenoid Sizing:

In the pushed-in position, the spring exerts a force of 1.148 lbs. In the released out position, the force exerts a force of 0.28 lbs. In the fully exerted position, our solenoid exerts a force of 1.75 lbs, thus this solenoid works for our application.



Updated Circuit Diagram:



Reflection:

Overall, starting the project early and having the work splitted ahead of time allowed us to have ample amount of time to debug and resolve any design issues that arose in the late stages of making the machine. To prevent any major issues from arising during manufacturing and get design feedback, we found it best to ask for help and suggestions from Jacob's engineering specialists as well as Tom in Hesse. Getting feedback will prevent design issues that will cause a setback in progress and potentially issues with finishing the project depending on at what stage the issue is realized.

As far as design learning opportunities, if we were to redesign the project knowing what we know now, instead of using a solenoid for actuation, we would have used either a linear actuator or a servo. Solenoids are great for small actuations but trying to get a 0.7 in actuation forced us to use a very big, loud, and unnecessary solenoid. Additionally, when originally doing the force calculations for the solenoid, we had only considered the static state and hadn't considered the momentum/impulse of the solenoid into the force calculations. This also led to use using a bigger solenoid than we had needed.

Appendices:
Apendix A. (Bill of Materials)
COTS

Name	Supplier	Part #	# of Parts	Packages	Cost Per Ind/Box	Total
Female Threaded Hex Standoff, Aluminum, 1/4" Hex, 2-3/4" Long, 6-32 Thread	McMaster	91780A644	9	9	1.76	15.84
Female Threaded Hex Standoff Aluminum, 1/4" Hex, 1-3/4" Long, 6-32 Thread	McMaster	91780A642	5	5	1.13	5.65
Ball Bearing, Flanged, Open with Ring, for 1/2" Shaft, 1-1/8" Housing ID	McMaster	6383K234	1	1	11.3	11.3
Compression Spring, 2" Long, 0.3" OD, 0.256" ID	McMaster	9657K384	10	2	7.26	14.52
Push-on External Retaining Rings for 4 mm OD, Black-Phosphate 1060-1090 Spring Stee	McMaster	92133A105	1	1	5.38	5.38
Ball Bearing, Open, Trade Number R2, for 1/8" Shaft Diameter	McMaster	60355K501	3	3	7.03	21.09
IEC Connector with Power Switch, Screw on, C14, 10A UL, DPST-NO	McMaster	5428N11	1	1	27.32	27.32
18-8 Stainless Steel Button Head Hex Drive Screw, 6-32 Thread Size, 3/8" Long	McMaster	92949A146	25	0	0	0
18-8 Stainless Steel Hex Drive Flat Head Screw, 82 Degree Countersink Angle, 6-32 Thread Size, 9/16" Long	McMaster	92210A149	17	1	7.89	7.89
18-8 Stainless Steel Hex Drive Flat Head Screw, 82 Degree Countersink Angle, 6-32 Thread Size, 1" Long	McMaster	92210A153	54	1	9.59	9.59
18-8 Stainless Steel Socket Head Screw, 6-32 Thread Size, 1/2" Long	McMaster	92196A148	19	1	8.72	8.72
18-8 Stainless Steel Button Head Hex Drive Screw, 5-40 Thread Size, 1/2" L	McMaster	92949A135	3	1	6.25	6.25
Super-Corrosion-Resistant 316 Stainless Steel Socket Head Screw, 2-56 Thread Size, 7/8" Long	McMaster	92185A095	2	1	5.61	5.61
Black-Oxide Alloy Steel Socket Head	McMaster	91864A	4	0	0	0

Screw, 4-40 Thread Size, 7/16" Long		081				
18-8 Stainless Steel Button Head Hex Drive Screw, 6-32 Thread Size, 1-3/4" Long	McMaster	92949A334	2	1	10.3	10.3
18-8 Stainless Steel Button Head Hex Drive Screw, 6-32 Thread Size, 1" Long	McMaster	92949A153	2	0	0	0
Black-Oxide Alloy Steel Hex Drive Flat Head Screw, 90 Degree Countersink Angle, M3 x 0.50 mm Thread, 8 mm Long	McMaster	91294A128	4	1	5.82	5.82
Low-Strength Steel Nylon-Insert Locknut, Zinc-Plated, 4-40 Thread Size	McMaster	90631A005	5	0	0	0
Steel Hex Nut, Medium-Strength, Class 8, M3 x 0.5 mm Thread	McMaster	90592A085	2	1	2.62	2.62
Low-Strength Steel Nylon-Insert Locknut, Zinc-Plated, 6-32 Thread Size	McMaster	90631A007	39	0	0	0
Low-Strength Steel Hex Nut, Zinc-Plated, 6-32 Thread Size	McMaster	90480A007	48	1	1.49	1.49
Low-Strength Steel Hex Nut, Zinc-Plated, 2-56 Thread Size	McMaster	90480A003	2	0	0	0
Low-Strength Steel Nylon-Insert Locknut, Zinc-Plated, 5-40 Thread Size	McMaster	90631A006	3	1	4.2	4.2
Sealed Linear Solenoid, Intermittent, Push, 1" Stroke, 68 oz. Force	McMaster	69905K621	1	1	60.4	60.4
Hairpin Cotter Pin, 1050-1095 Steel for 1/4"-1/2" Pin Diameter, 3/32" Wire Diameter	McMaster	92375A440	1	0	0	0
McCormick Bottles	Safeway	NA	10	10	0	0
25d-metal-gearmotor-34-47-encoder	Polulu	4844	1	1	48.95	48.95
DRV8874 DC Motor Driver	Polulu	4035	1	1	9.95	9.95
3.5inch LCD Display Module SPI TFT ILI9488 65K Color Expandable Screen	Amazon	ILI9488	1	1	22.5	22.5
S-120-12 PSU	Tom :)	NA	1	1	0	0
Limit Switch	Tom :)	NA	1	0	0	0
				Total	**$305.39**	

Raw Materials:

Material	**Size**	Supplier	**Qty**	**Cost**	**Total Cost:**
1/4" Plywood	24"x48"	Jacobs	1	13.32	**$96.32**
1/8" Plywood	18"x30"	Jacobs	1	2.22	
1/8" 6061 AL	24"x24"	Jacobs	1	45.03	
0.035" Mild Steel	24'"x48"	Jacobs	1	15.75	
White PLA	1KG	Amazon	1	20	

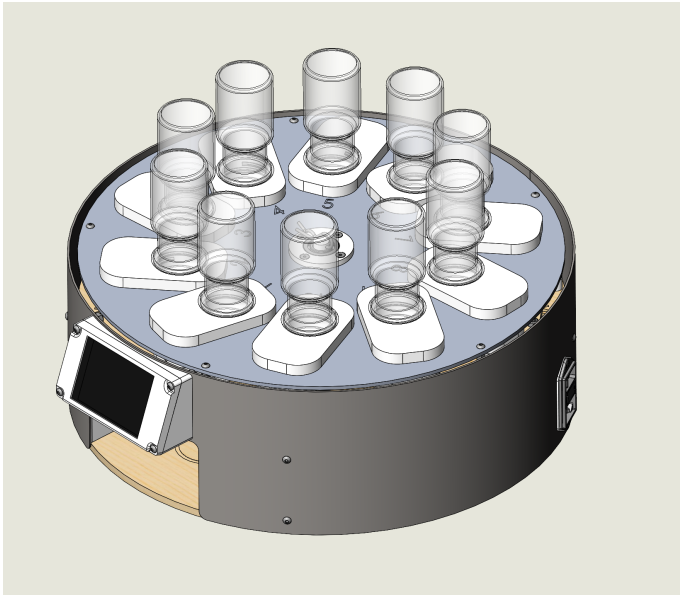
Fabricated Materials:

Name	**Material**	**Fabrication Method**	**Qty**
Axle & Solenoid Mount	PLA	3D Print	1
Bearing Clamp	PLA	3D Print	1
Bearing Mount	PLA	3D Print	1
Spice Cartridge Mount Disk	1/8" 6061 AL	Fablight	1
Base Plate	Plywood	Laser Cutter	1
Intermediate Plate	Plywood	Laser Cutter	1
175T 2M Internal Spur Gear	Plywood	Laser Cutter	1
Upper Shell Mount	PLA	3D Print	4
Lower Shell Mount	PLA	3D Print	4
Power Input Mount	PLA	3D Print	1
Inner Shell	PLA	3D Print	1
Outer Shell	1/32" Mild Steel	Fablight	1
Display Mount Outer	PLA	3D Print	1
Display Mount Inner	PLA	3D Print	1
Support Bearing Mount	PLA	3D Print	3
10T 2M Pinion	Plywood	Laser Cutter	1
Spice Cartridge Upper	PLA	3D Print	10
Spice Cartridge Middle	PLA	3D Print	10
Spice Cartridge Lower	PLA	3D Print	10
Spice Cartridge Slider	PLA	3D Print	10
Limit Switch Spacer	PLA	3D Print	1

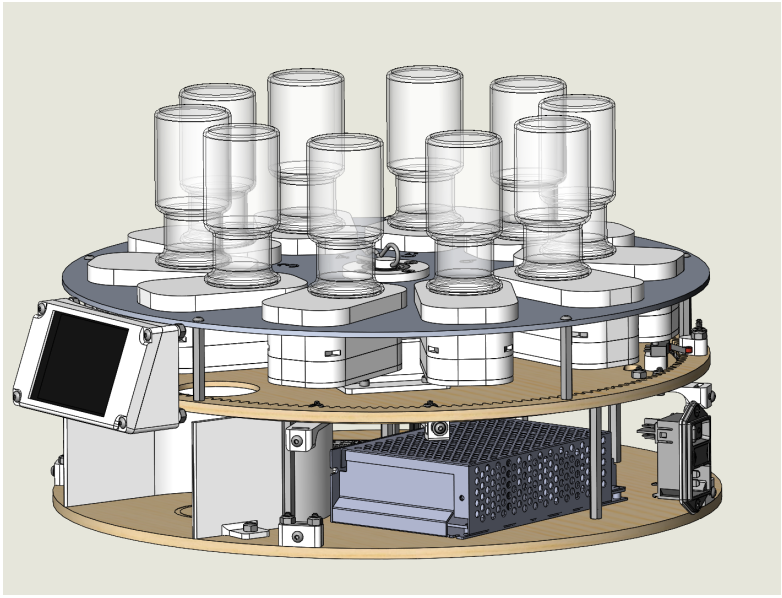
Limit Switch Pole	PLA	3D Print	1
Support Bearing Spacer	PLA	3D Print	3
Shell Connector	PLA	3D Print	2
Nut Spacer	PLA	3D Print	4

Appendix B: (CAD)

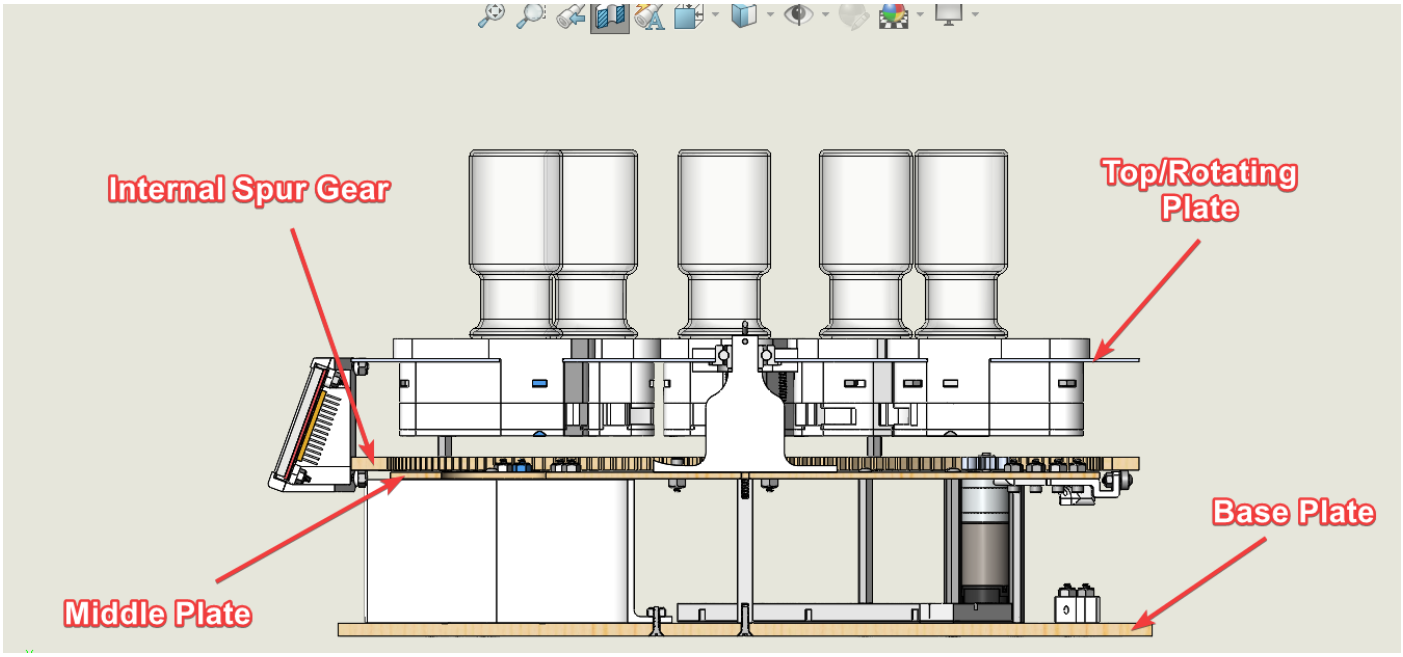
Isometric View of Full Assembly



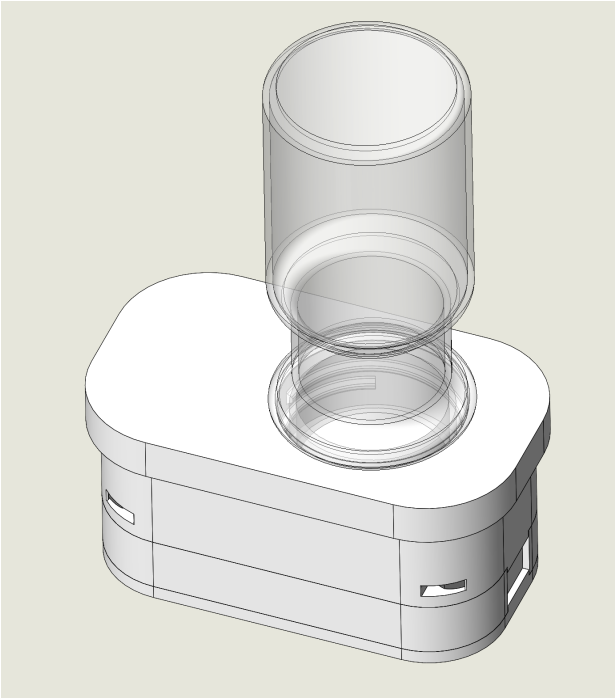
Shell/Casing/Cover Removed View



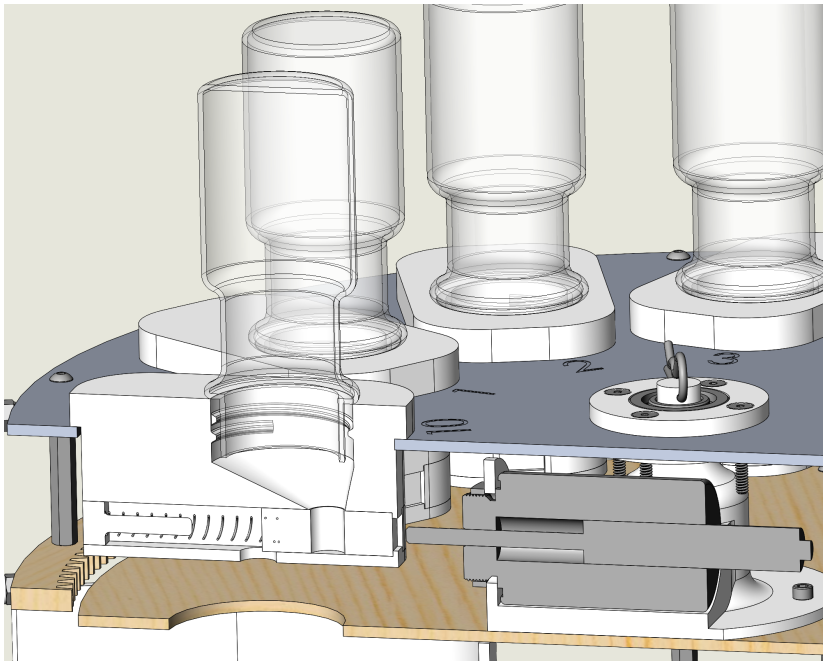
Section View - Full Assembly



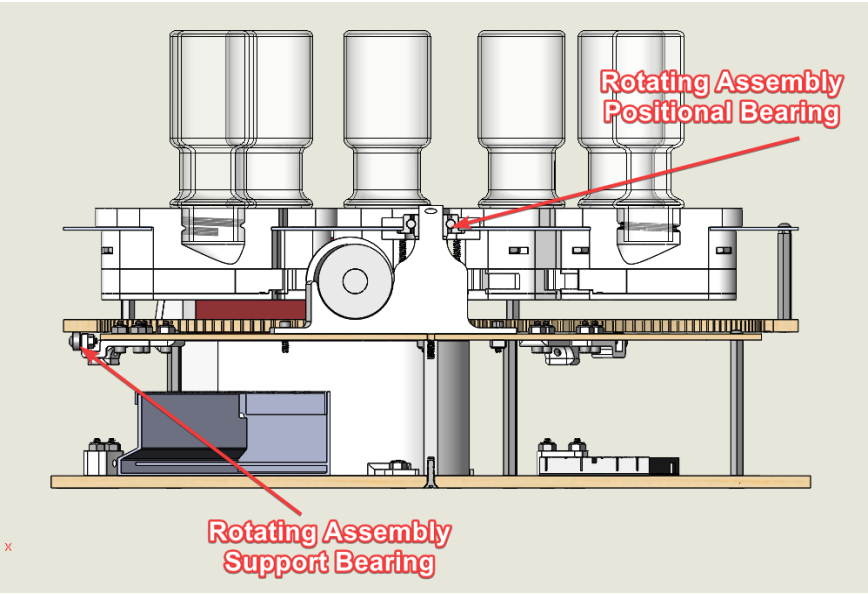
Spice Cartridge Isometric



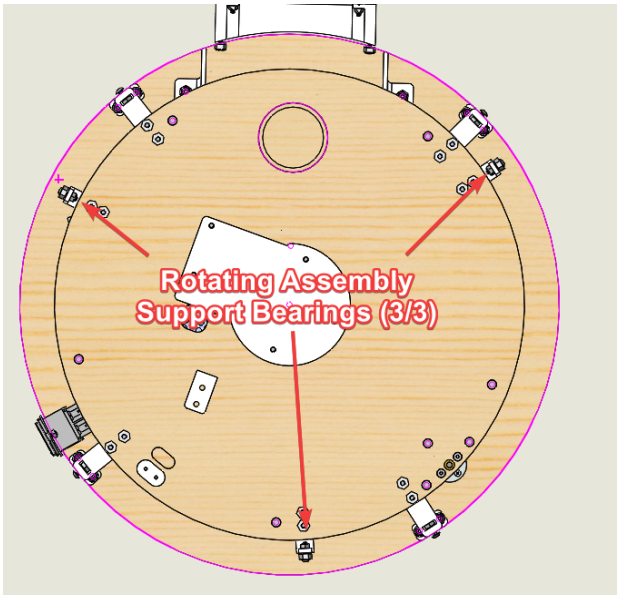
Section View - Solenoid/Cartridge Slider Interaction



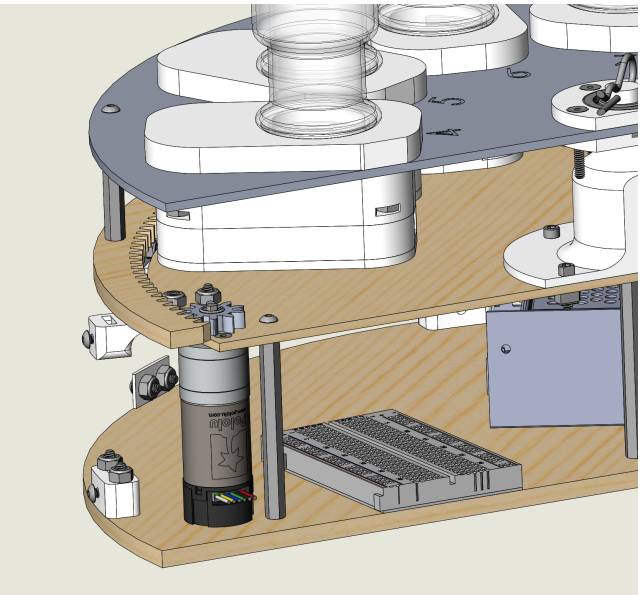
Section View - Axle Bearing + Support Bearing (⅓)



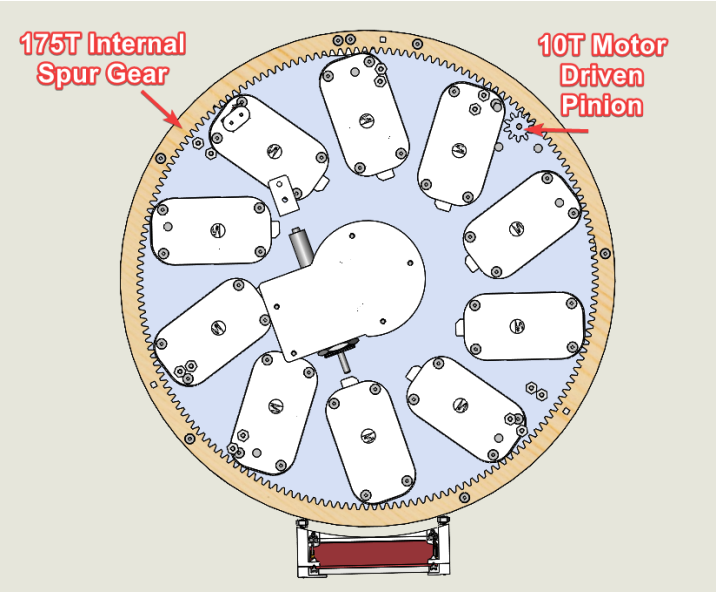
Section View - Support Bearings



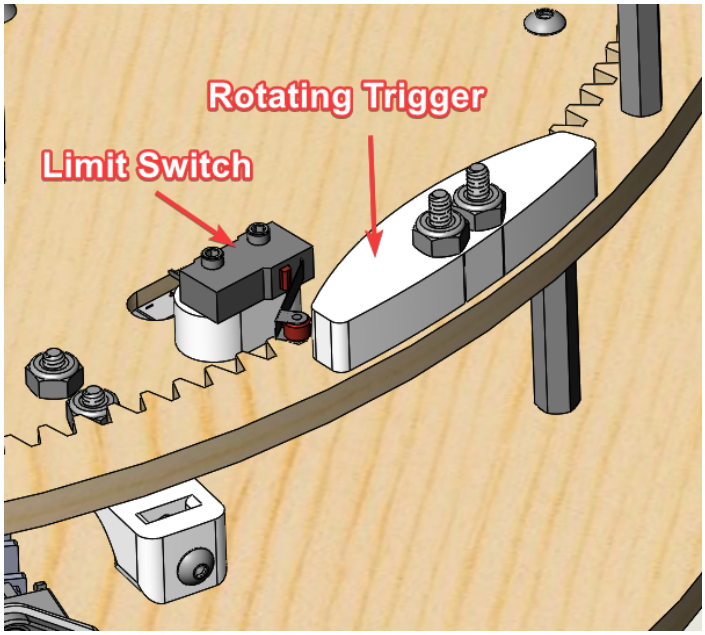
Section View - Motion Transmission



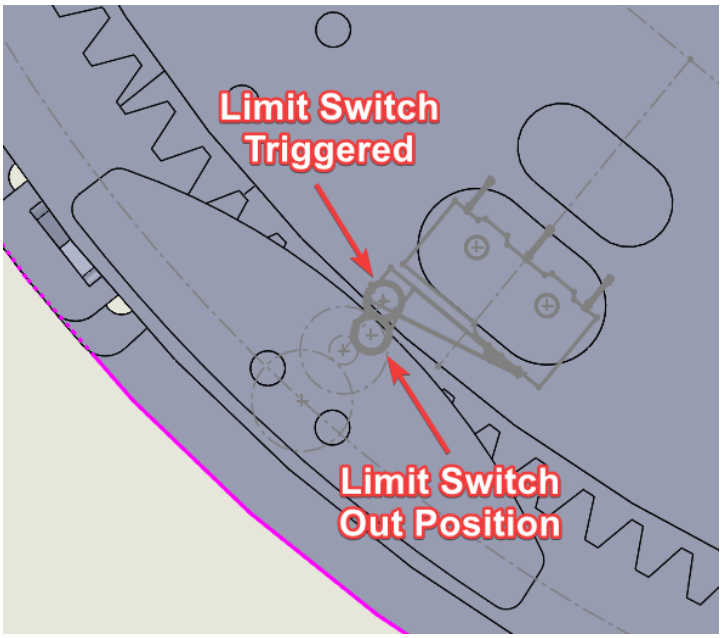
Rotating Assembly Bottom View



Limit Switch Assembly View



Limit Switch Planning Sketch



Apppendix C: (Code)

```
#include <ESP32Encoder.h>
#include <ezButton.h>
#include <SPI.h>
#include <TFT_eSPI.h>
#include "bitmap.h"
#include "Free_Fonts.h"
#define BIN_1 26
#define BIN_2 25
#define SOLENOID_PIN 13

ezButton limitSwitch(17);
TFT_eSPI tft = TFT_eSPI();
ESP32Encoder encoder;

//Global Variables ----------------------------

//States
int state = 0;
int substate = 0;
int substate2 = 0;

//Position
int theta = 0;
int thetaMax = 14286;
int offset = 1560;
int increment = 1429;
int D = 0;
int pos_counter = 0;
bool positioncheck = false; //Used as an event checker

//PID Variables
int thetaDes;
int error;
float Kp = 1.2;

//user input

//int indexes = 4;
int idxSpice = 1;
int idxAmt = 1;
int positions[] = {1,0,0,0};
int amounts[] = {1,0,0,0};
```

```
int indexes = 1;
int p;

//dispensing input
int pos_index = 0;
int pos;
bool disp_state = false;
int disp = 0;

//helper data structures for drawing
struct square{
int xStart;
int yStart;
int width;
int height;
int16_t color;
};
struct roundedSquare {
int xStart;
int yStart;
int width;
int height;
byte cornerRadius;
int16_t color;
};
void drawFillSquare(square toDraw){
tft.fillRect(toDraw.xStart, toDraw.yStart, toDraw.width, toDraw.height, toDraw.
color);
}
void drawFillRoundRect(roundedSquare toDraw){
tft.fillRoundRect(
toDraw.xStart,
toDraw.yStart,
toDraw.width,
toDraw.height,
toDraw.cornerRadius,
toDraw.color
);
}

//button data structures
roundedSquare leftSpice = {
35,
65,
```

```
60,
45,

10,
0xE71A
};
roundedSquare rightSpice = {
385,
65,
60,
45,
10,
0xE71A
};
roundedSquare lessAmt = {
35,
190,
60,
45,
10,
0xE71A
};
roundedSquare moreAmt = {
385,
190,
60,
45,
10,
0xE71A
};
roundedSquare dispense = {
129,
256,
220,
50,
10,
0x8EED
};

//text bg data structures
square spiceNameBg = {
leftSpice.xStart + leftSpice.width,
leftSpice.yStart,
rightSpice.xStart - (leftSpice.xStart + leftSpice.width),
```

```
leftSpice.height,

TFT_WHITE
};
square spiceAmtBg = {
lessAmt.xStart + lessAmt.width,
lessAmt.yStart,
moreAmt.xStart - (lessAmt.xStart + lessAmt.width),
lessAmt.height,
TFT_WHITE
};

//button drawing functions
void leftSpiceBtn() {
drawFillRoundRect(leftSpice);
tft.setFreeFont(FSSB12);
tft.setTextColor(TFT_BLACK, 0xE71A);
tft.setTextDatum(TL_DATUM);
tft.drawString("<", leftSpice.xStart + 25, 75);
}
void rightSpiceBtn() {
drawFillRoundRect(rightSpice);
tft.setFreeFont(FSSB12);
tft.setTextColor(TFT_BLACK, 0xE71A);
tft.setTextDatum(TL_DATUM);
tft.drawString(">", rightSpice.xStart + 25, 75);
}
void lessAmtBtn() {
drawFillRoundRect(lessAmt);
tft.setFreeFont(FSSB12);
tft.setTextColor(TFT_BLACK, 0xE71A);
tft.setTextDatum(MC_DATUM);
tft.drawString("-", lessAmt.xStart + lessAmt.width / 2, lessAmt.yStart + lessAmt.
height / 2);
}
void delLessAmtBtn() {
tft.fillRect(lessAmt.xStart, lessAmt.yStart, lessAmt.width, lessAmt.height,
TFT_WHITE);
}
void delMoreAmtBtn() {
tft.fillRect(moreAmt.xStart, moreAmt.yStart, moreAmt.width, moreAmt.height,
TFT_WHITE);
}
```

```
void moreAmtBtn() {
drawFillRoundRect(moreAmt);
tft.setFreeFont(FSSB12);
tft.setTextColor(TFT_BLACK, 0xE71A);
tft.setTextDatum(MC_DATUM);
tft.drawString("+", moreAmt.xStart + moreAmt.width / 2, moreAmt.yStart + moreAmt.
height / 2);
}
void delAmts() {
tft.fillRect(0, 150, 480, 170, TFT_WHITE);
}
void dispenseBtn() {
drawFillRoundRect(dispense);
tft.setFreeFont(FSSB18);
tft.setTextColor(TFT_BLACK, 0x8EED);
tft.setTextDatum(TC_DATUM);
tft.drawString("Dispense!", dispense.xStart + dispense.width / 2, dispense.yStart
+ 10);
}

//text display functions
void spiceName() {
drawFillSquare(spiceNameBg);
tft.setFreeFont(FSSB12);
tft.setTextColor(TFT_BLACK, TFT_WHITE);
tft.setTextDatum(MC_DATUM);
String spiceName;
if (idxSpice > 10) {
spiceName = "Combo #" + String(idxSpice - 10);
} else {
spiceName = "Spice #" + String(idxSpice);
}
tft.drawString(spiceName, tft.getViewportWidth() / 2, leftSpice.yStart + leftSpice.
height / 2);
}
void spiceAmt() {
drawFillSquare(spiceAmtBg);
tft.setFreeFont(FSSB12);
tft.setTextColor(TFT_BLACK, TFT_WHITE);

tft.setTextDatum(MC_DATUM);
String spiceAmt;
if (idxSpice == 11) {
spiceAmt = String(idxAmt * 10) + "/4 tsp";
```

```
} else if (idxSpice == 12) {
spiceAmt = String(idxAmt * 2) + "/4 tsp";
} else {
spiceAmt = String(idxAmt) + "/4 tsp";
}
tft.drawString(spiceAmt, tft.getViewportWidth() / 2, lessAmt.yStart + lessAmt.
height / 2);
}

//draw initial screen
void initialScreen() {
tft.fillScreen(TFT_WHITE);
tft.setFreeFont(FSSB24);
tft.setTextColor(TFT_BLACK, TFT_WHITE);
tft.setTextDatum(MC_DATUM);
tft.drawString("Initializing...", tft.getViewportWidth() / 2, tft.
getViewportHeight() / 2);
}

//draw main menu
void mainMenu() {
tft.fillScreen(TFT_WHITE);
leftSpiceBtn();
rightSpiceBtn();
if (idxAmt > 1) {
lessAmtBtn();
}
moreAmtBtn();
tft.setFreeFont(FSSB18);
tft.setTextColor(TFT_BLACK, TFT_WHITE);
tft.setTextDatum(TL_DATUM);
tft.drawString("Pick a spice:", 30, 25);
tft.drawString("Spice amount:", 30, 150);
spiceName();
spiceAmt();

dispenseBtn();
}

//touch points
uint16_t t_x = 0, t_y = 0;

//Encoder Timer ----------------------------
volatile int count = 0; // encoder count
```

```
volatile bool deltaT_motor = false; // event - check timer interrupt 1
volatile bool deltaT_sol = false; // event - check timer interrupt 0
volatile bool debounceT = false; // event - check timer interrupt 2
int sol_counter = 0;
hw_timer_t * timer1 = NULL; //encoder timer
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;
hw_timer_t * timer2 = NULL; // touch timer
portMUX_TYPE timerMux2 = portMUX_INITIALIZER_UNLOCKED;

// setting PWM properties ----------------------------
const int freq = 20000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 200;
const int MIN_PWM_VOLTAGE = 160;

//Initialization -----------------------------------
void IRAM_ATTR onTime1() {
portENTER_CRITICAL_ISR(&timerMux1);

//encoder stuff
count = encoder.getCount( );
encoder.clearCount ( );
deltaT_motor = true; // the function to be called when timer interrupt is triggered

//solenoid stuff
sol_counter = sol_counter + 1;
if (sol_counter > 100){
sol_counter = 0;
deltaT_sol = true;
}
portEXIT_CRITICAL_ISR(&timerMux1);
}

void IRAM_ATTR onTime2() {
portENTER_CRITICAL_ISR(&timerMux2);
debounceT = true; // the function to be called when timer interrupt is triggered

//Serial.println("timer trig");
portEXIT_CRITICAL_ISR(&timerMux2);
}

//functions for TOUCH TIMER ONLY
```

```
void start_timer() {
Serial.println("timer start");
timerWrite(timer2, 0);
timerStart(timer2);
}
void stop_timer() {
timerStop(timer2);
}

//ARDUINO FUNCTIONS ------------------------------
void setup() {
limitSwitch.setDebounceTime(5);
Serial.begin(115200);

//touchscreen set up
tft.setRotation(1);
tft.init();
initialScreen();

//Encoder Set up
ESP32Encoder::useInternalWeakPullResistors = UP; // Enable the weak pull up
resistors
encoder.attachHalfQuad(33, 27); // Attache pins for use as encoder pins
encoder.setCount(0); // set starting count value after attaching

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

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

// initilize timer
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, 10000, true); // 10000 * 1 us = 10 ms, autoreload true

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

//initialize touch delay timer
timer2 = timerBegin(0, 80, true); // timer 0, MWDT clock period = 12.5 ns *
TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
```

```
timerAttachInterrupt(timer2, &onTime2, true); // edge (not level) triggered
timerAlarmWrite(timer2, 40000, true); // 40000 * 1 us = 50 ms, autoreload true
timerAlarmEnable(timer2);
stop_timer();
pinMode(SOLENOID_PIN, OUTPUT); //solenoid
digitalWrite(SOLENOID_PIN, LOW);
pinMode(BIN_1, OUTPUT); //Motor Direction
}
void loop() {
switch (state) {
case 0 : //Non-initialized State
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
limitSwitch.loop(); //Turns on Limit Switch Detection
switch (substate2) {
case 0: //Initial Zeroing
----------------------------------------------------------------------
if(limitSwitch.isPressed()){
theta = 0; //Sets Zero
substate2 = 1; //Goes to final zeroing
break;
}
if (deltaT_motor) {
updateTheta();
D = -254;
runMotor(D);
plotControlData();
}
break;
case 1: // Zero Fine Tuning
----------------------------------------------------------------------

if (positioncheck){
if(limitSwitch.isPressed()){
theta = 0; //Sets Zero
runMotor(LOW); //Turns of Motor
state = 1; //Goes to Initialized State
mainMenu();
break;
}
if (deltaT_motor) {
updateTheta();
D = 50;
runMotor(D);
```

```
plotControlData();
}
break;
}else{
goTo(-825); //Moves past limit switch
plotControlData();
if (theta < -800){
positioncheck = true; //ready to do fine zeroing
}
}
break;
}
break;
case 1 : //Initialized State
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
switch (substate) {
case 0 : //User Input State or Idling
-----------------------------------------------------------------------

/**User input code goes here
change the positions[], amounts[], and int indexes to change cycle pattern
example:
positions[] = {1,3,8};
amounts[] = {2,3,1}
indexes = 3 (just how many elements in the above arrays)
these conditions will dispense 2x of spice 1, 3x of spice 3, 1x of spice 8
in that order
**/

if (checkTouch() == true && checkTimer() == true) {
processTouch();
}
break;

case 1 : //Rotating State
-----------------------------------------------------------------------
if (deltaT_motor) {
pos = positions[pos_index];
thetaDes = -increment*(pos-1) + offset;
goTo(thetaDes);
if (check4Pos(thetaDes, 10)){
runMotor(LOW);
substate = 2;
```

```
deltaT_sol = false;
sol_counter = 0;
break;
}
plotControlData();
}

break;
case 2 : //Dispensing State
--------------------------------------------------------------------
if (deltaT_sol){
deltaT_sol = false;
sol_counter = 0;
if (not disp_state and disp < amounts[pos_index]) {
digitalWrite(SOLENOID_PIN, HIGH);
disp_state = true;
Serial.println("Solenoid ON");
} else if (disp < amounts[pos_index]){
digitalWrite(SOLENOID_PIN, LOW);
Serial.println("Solenoid OFF");
disp_state = false;
disp = disp + 1;
} else {

disp = disp + 1;
}
if (disp > amounts[pos_index]){
pos_index = pos_index + 1;
disp = 0;
if (pos_index == indexes){
substate = 0;
pos_index = 0;
mainMenu();
//Serial.println("Done");
} else {
substate = 1;
}
}
}
break;
default: //Substate Error
--------------------------------------------------------------------
break;
}
```

```
break;
default: //Main State Error
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
//Serial.println("Main State Error");
break;
}
}
//touchscreen event checkers
bool checkTouch() {
if(tft.getTouch(&t_x, &t_y)) {
if (!timerStarted(timer2)) {
start_timer();
}
return true;
}
else {
return false;
}
}
bool checkTimer() {

if (debounceT == true) {
debounceT = false;
stop_timer();
return true;
}
else {
return false;
}
}
//touchscreen service
void processTouch(){
if(tft.getTouch(&t_x, &t_y)){
//Here we check if the touch is contained within our imaginary box
Serial.println(String(t_x) + ", " + String(t_y));
if(t_x > leftSpice.xStart && t_x < leftSpice.xStart + leftSpice.width && t_y
< 320 - leftSpice.yStart && t_y > 320 - (leftSpice.yStart + leftSpice.height)){
//Serial.println("left spice");
idxSpice -= 1;
if (idxSpice < 1) {
idxSpice = 12;
}
spiceName();
```

```
}
if(t_x > rightSpice.xStart && t_x < rightSpice.xStart + rightSpice.width &&
t_y < 320 - rightSpice.yStart && t_y > 320 - (rightSpice.yStart + rightSpice.
height)){
idxSpice += 1;
if (idxSpice > 12) {
idxSpice = 1;
}
spiceName();
}
//if (idxSpice == 11 || idxSpice == 12) {
//delLessAmtBtn();
//delMoreAmtBtn();
//spiceAmt();
//}

//if (idxSpice < 11) {
spiceAmt();
if(t_x > lessAmt.xStart && t_x < lessAmt.xStart + lessAmt.width && t_y <
320 - lessAmt.yStart && t_y > 320 - (lessAmt.yStart + lessAmt.height) && idxAmt > 1){
//Serial.println("less amt");
idxAmt -= 1;
spiceAmt();
}
if(t_x > rightSpice.xStart && t_x < moreAmt.xStart + moreAmt.width && t_y
< 320 - moreAmt.yStart && t_y > 320 - (moreAmt.yStart + moreAmt.height)){
//Serial.println("more amt");
idxAmt += 1;
spiceAmt();
}
if (idxAmt > 1) {
lessAmtBtn();
}
else {
delLessAmtBtn();
}
moreAmtBtn();
//}
if(t_x > dispense.xStart && t_x < dispense.xStart + dispense.width && t_y <
320 - dispense.yStart && t_y > 320 - (dispense.yStart + dispense.height)) {
setDispenseOutputs();
substate = 1;
dispensingImg();
Serial.println("dispensing...");
```

```
Serial.println(amounts[0]);
Serial.println(positions[0]);
Serial.println(indexes);
}
}
}
//helper functions for dispensing
void setDispenseOutputs() {
if (idxSpice <= 10) {
positions[0] = idxSpice;
amounts[0] = idxAmt;
indexes = 1;
}
else if (idxSpice == 11) {

positions[0] = 1;
positions[1] = 6;
positions[2] = 4;
positions[3] = 9;
amounts[0] = idxAmt*1;
amounts[1] = idxAmt*3;
amounts[2] = idxAmt*2;
amounts[3] = idxAmt*4;
indexes = 4;
} else {
positions[0] = 2;
positions[1] = 7;
amounts[0] = idxAmt*1;
amounts[1] = idxAmt*1;
indexes = 2;
}
}
void dispensingImg() {
tft.setSwapBytes(true); // Swap the byte order for pushImage() - corrects
endianness
tft.fillScreen(TFT_WHITE);
tft.pushImage(0, 0, 480, 320, salt_bae);
}
//Other functions
-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-
+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+=-+
=-+=
//Service
//Moves to position theta Des
```

```
void goTo(int thetaDes){
if (deltaT_motor) {
updateTheta();
error = thetaDes - theta;
D = Kp*error;
//Ensure that you don't go past the maximum possible command
if (D > MAX_PWM_VOLTAGE) {
D = MAX_PWM_VOLTAGE;
}

else if (D < -MAX_PWM_VOLTAGE) {
D = -MAX_PWM_VOLTAGE;
}
//Map the D value to motor directionality
//FLIP ENCODER PINS SO SPEED AND D HAVE SAME SIGN
runMotor(D);
}
}
//Service
//Updates the theta with count from timer
void updateTheta(){
portENTER_CRITICAL(&timerMux1);
deltaT_motor = false;
portEXIT_CRITICAL(&timerMux1);
theta += count;
}
//Service
//Sets motor to speed D
void runMotor(int D){
if (D > 0) {
digitalWrite(BIN_1, HIGH);
ledcWrite(ledChannel_2, D);
}
else if (D < 0) {
digitalWrite(BIN_1, LOW);
ledcWrite(ledChannel_2, -D);
}
else {
digitalWrite(BIN_1, LOW);
ledcWrite(ledChannel_2, LOW);
}
}
//Event Checker
//checks to see if theta is in allowable error
```

```
//if so, counter amount of times, and then returns true
bool check4Pos(int Des, int counter){
if (abs(Des - theta) < 10){
pos_counter = pos_counter + 1;
}
if (pos_counter > counter){

pos_counter = 0;
return true;
}
return false;
}
//Service
//Plots Control Data
void plotControlData() {
//Serial.println("Position, Desired_Position, PWM_Duty");
//Serial.print(theta);
//Serial.print(" ");
//Serial.print(thetaDes);
//Serial.print(" ");
//Serial.println(D);
```