Automatic Drink Dispenser

Mia Galatis, Ming Chin, Can Aydin, and Ali Ahmed
Dept. of Mechanical Engineering, University of California at Berkeley
MECENG 102B: Mechatronics Design
December 19, 2024

Opportunity: An automated drink dispenser for cocktails that offers a unique opportunity to make the process of mixing drinks quicker, more efficient, and standardized – perfect for home bartenders and party hosts. Our system would combine convenience, precision, and customization, transforming the at-home cocktail experience into something fun and effortless for the user.

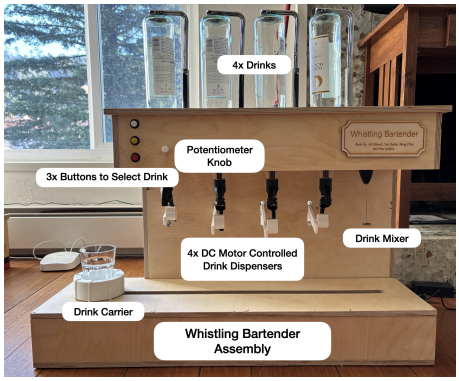
High-level Strategy: Our final project is composed of three main systems– dispensing, mixing, and a linear motion system. We designed and manufactured a 0.5-inch plywood housing that holds all our components together for a clean, professional look. The system is controlled with an ESP32 and a PCA9685 PWM extension board to add more PWM output pins to drive our DC motors.

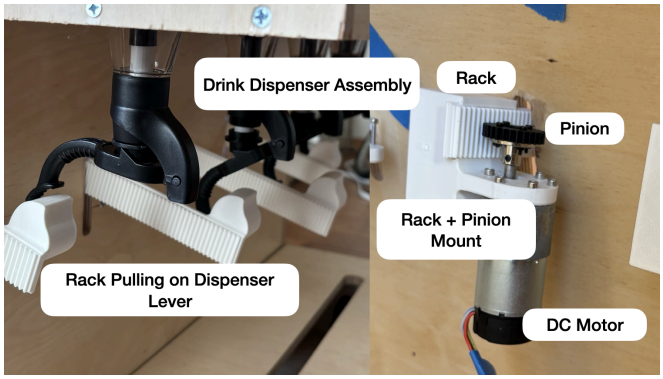
Dispensing: This system is composed of four DC brush motors attached to a 3D printed rack and pinion mechanism that would pull a lever of the drink dispensers to dispense the liquid.

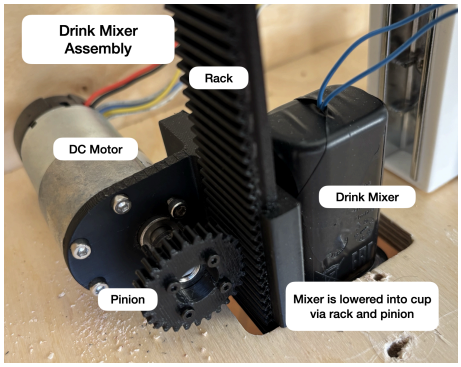
Mixing Assembly: The mixer also utilizes a DC brush motor that is attached to a rack and pinion mechanism. This moves the frother up and down to get the tip of the frother in and out of the cup. The frother itself consists of a simple brushed DC motor that is connected to a relay board and 5V power.

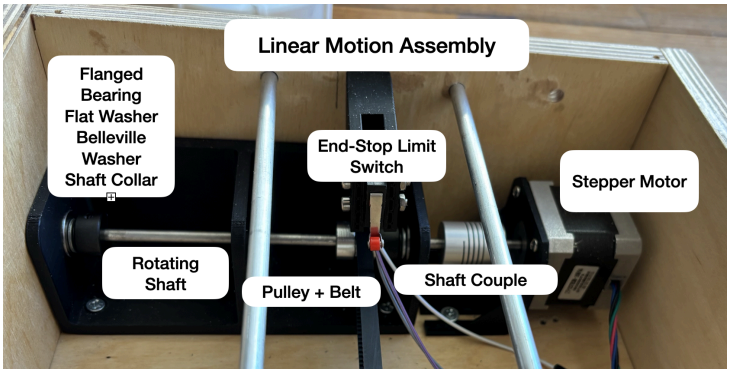
Linear motion system: The carriage assembly moves the cup from dispenser to dispenser. The carrier system is composed of two linear rods, a driving and driven pulley, shaft collars, a belt tensioning system, linear ball bearings, a cup holder, and a carriage.

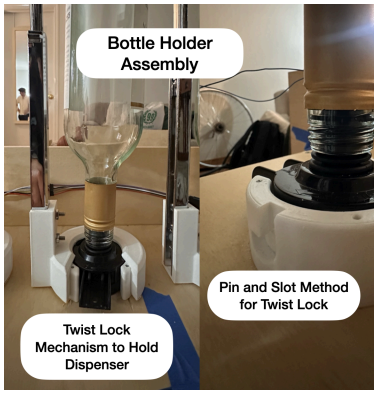
Integrated Design Diagram:

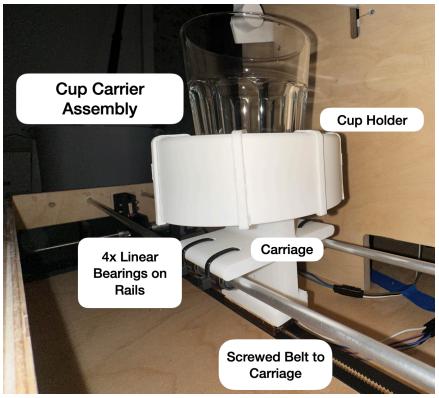


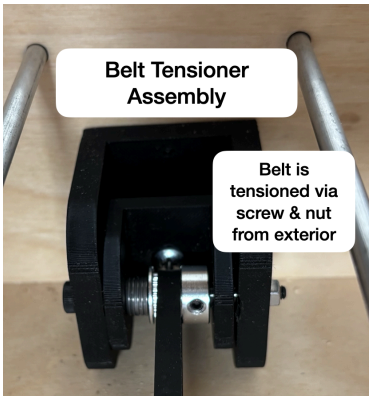




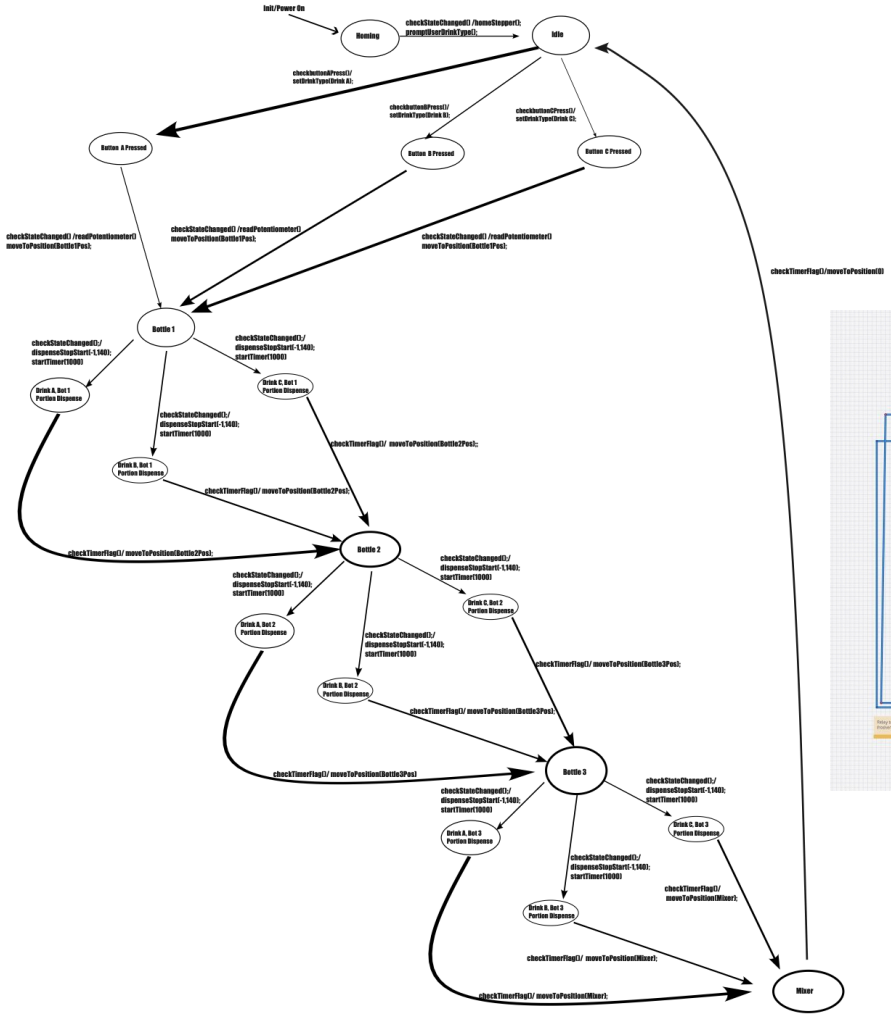


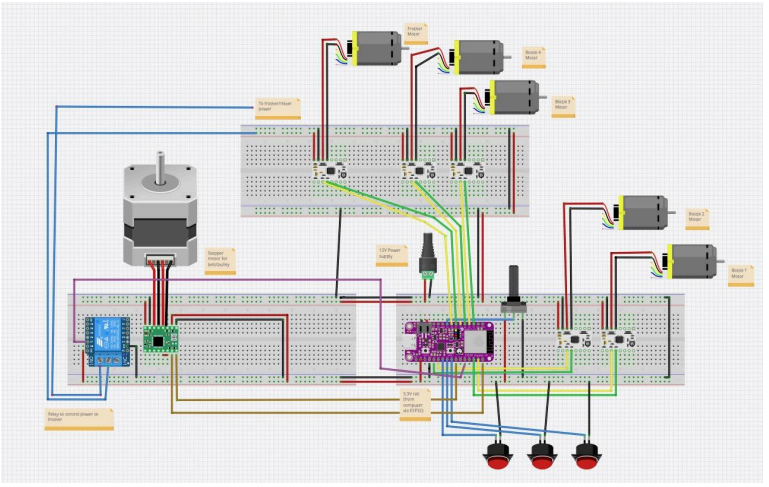






Final Circuit and State Diagram





Critical Design Decisions:

One of the biggest function-critical decisions we had to make was whether the chosen DC brush motors would have enough torque to pull on the dispenser lever to disengage the stopper. To this end, we calculated motor torque to ensure that the motor we would be using was strong enough. After we calculated motor torque, we purchased one motor and designed a test stand that would emulate the real conditions for physical testing.

$Given\ in\ motor\ spec\ sheet\ and\ Bearing\ specs$
$Bore\ diameter\ = 5.00mm$
$Outer\ diameter\ = 10.2mm$
$Dynamic\ load\ capacity: 90\ lbs\ = 400.3399\ N$
$Static\ load\ capacity: 35\ lbs\ = 155.68776\ N$
$Maximum\ speed: 71000\ rpm$

$T = Motor\ torque \times Gear\ ratio$
$T = 0.63743Nm \times 6.3 = 4.015746Nm$
$n_{out} = \frac{490\ rpm}{6.3} = 77.778\ rpm$

$Tangential\ force: F_t = \frac{T_m}{radius} = \frac{0.63743\ Nm}{0.003m} = 212.47667\ N$
$Bearing\ force: F_{bearing} = 212.47667N \times tan(20°) = 77.3352\ N$

$Load\ comparison$

$F_{bearing} = 77.34\ N < 400.3399\ N \quad (Dynamic\ load)$
$F_{bearing} = 77.34\ N < 155.68776\ N \quad (Static\ load)$

We need to consider a factor of safety to ensure that we are "overestimating" the force load on the bearings for a worst-case scenario. Therefore,

$Assume\ FOS: 77.34\ N \times 1.5 = 116.01\ N$

Even with a 1.5 FOS, the bearings are still able to withstand the torque of the motor

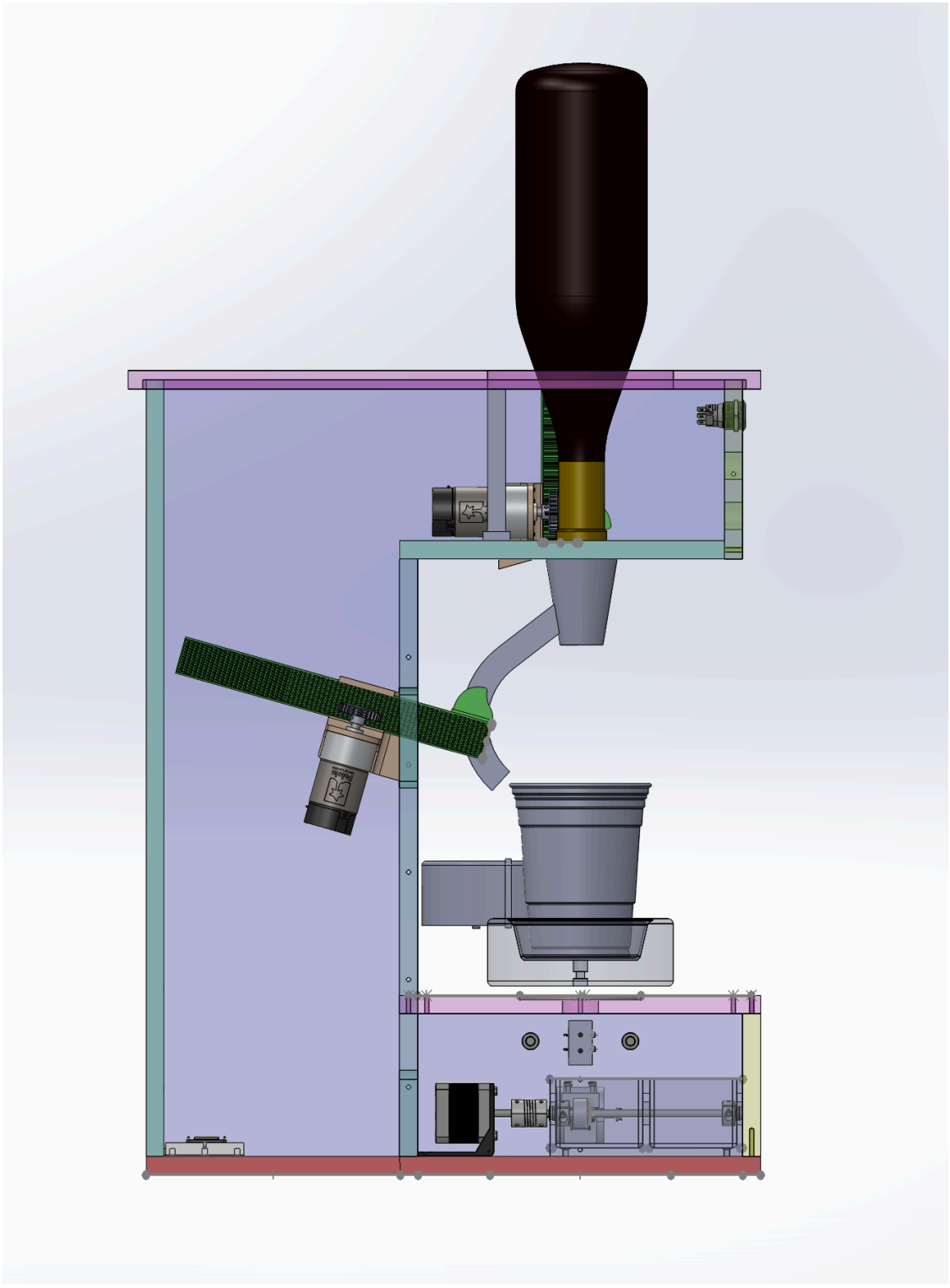
Reflection: One of our major fallbacks during this project was being too ambitious during the onset of the design process. For example, our initial design utilized a total of six bottles. We decided to scale down to four bottles so we would not run out of GPIO pins for the DC motors. Even with this consideration in mind, we still struggled to accommodate all the motors & buttons as many IO pins on the ESP32 were actually reserved pins with specific functions. We would recommend creating a preliminary design and circuit diagram while trying to choose projects.

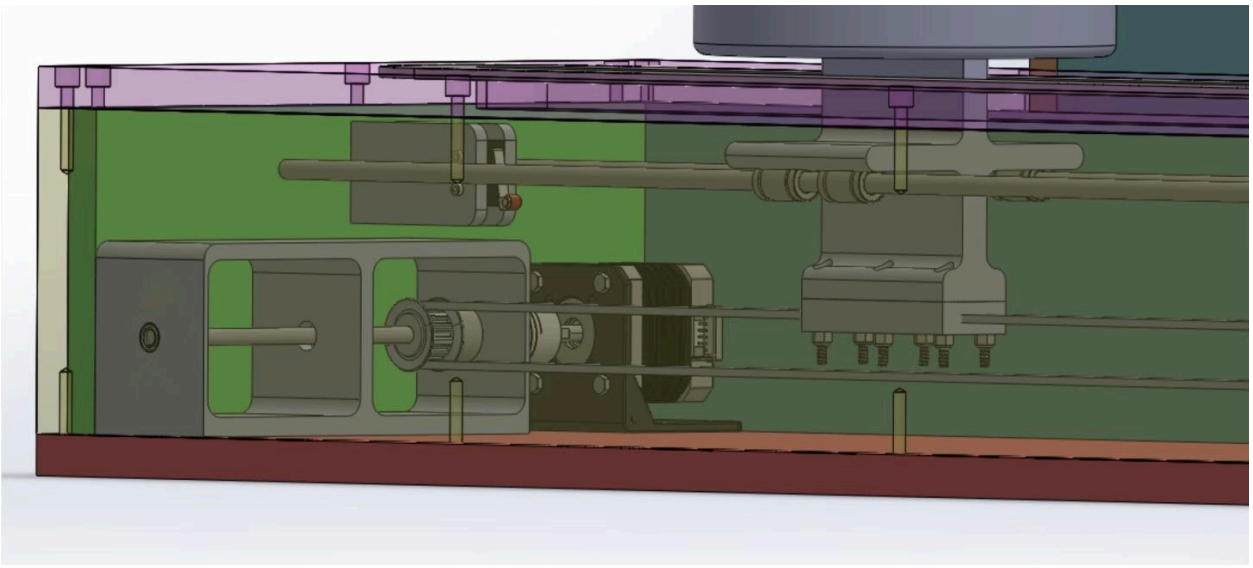
Appendix A: Bill of Material (BOM)

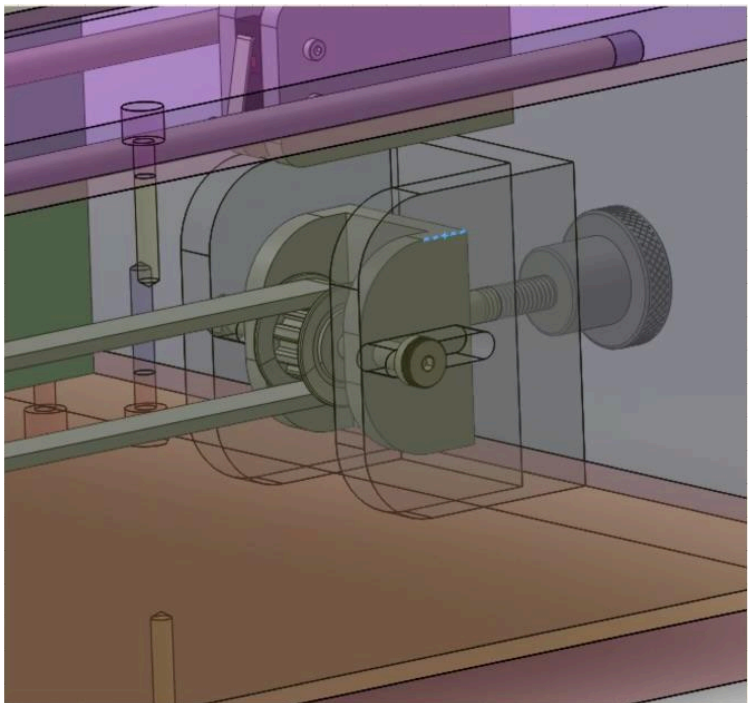
ME102B Drink Dispenser Bill of Material				
Part Number/Type	Quantity	Unit Cost	Total Cost	Source/Link
4’x8’ Birch Plywood	1.5	$87.63	$131.45	https://store.jacobshall.org/products/1-2-plywood-shopbot
Linear Ball Bearings	1	$8.59	$8.59	https://www.amazon.com/dp/B0C3R38HJ7?ref=ppx_yo2ov_dt_b_fed_asin_title
12V AC Power Supply	1	$9.99	$9.99	https://www.amazon.com/dp/B07GFFG1BQ?ref=ppx_yo2ov_dt_b_fed_asin_title
6-Bottle Revolving Liquor Dispenser	1	$42.49	$42.49	https://www.amazon.com/dp/B00WRXG08Y?ref=ppx_yo2ov_dt_b_fed_asin_title
5M GT2 Timing Belt 6mm Width	1	$9.99	$9.99	https://www.amazon.com/dp/B0CK21WWGV?ref=ppx_yo2ov_dt_b_fed_asin_title
12v to 5v 5A Converter Step-Down Power Supply	1	$14.99	$14.99	https://www.amazon.com/dp/B09C4HPNJ8?ref=ppx_yo2ov_dt_b_fed_asin_title
Flexible Couplings	1	$8.99	$8.99	https://www.amazon.com/dp/B07DC2CV6T?ref=ppx_yo2ov_dt_b_fed_asin_title
16MM Momentary Push Button on Off Switch	1	$11.49	$11.49	https://www.amazon.com/dp/B08SKJ6V7Z?ref=ppx_yo2ov_dt_b_fed_asin_title
6mm Flange Coupling Connector	1	$8.99	$8.99	https://www.amazon.com/dp/B08334N261?ref=ppx_yo2ov_dt_b_fed_asin_title&th=1
Micro Limit Switch	1	$6.60	$6.60	https://www.amazon.com/dp/B07X142VGC?ref=ppx_yo2ov_dt_b_fed_asin_title
DC Geared-Down Motor 37Dx72.5L mm 6V/12V, with 64 CPR Encoder	5	$33.99	$169.95	https://www.amazon.com/dp/B08X3CDZRF?ref=ppx_yo2ov_dt_b_fed_asin_title&th=1

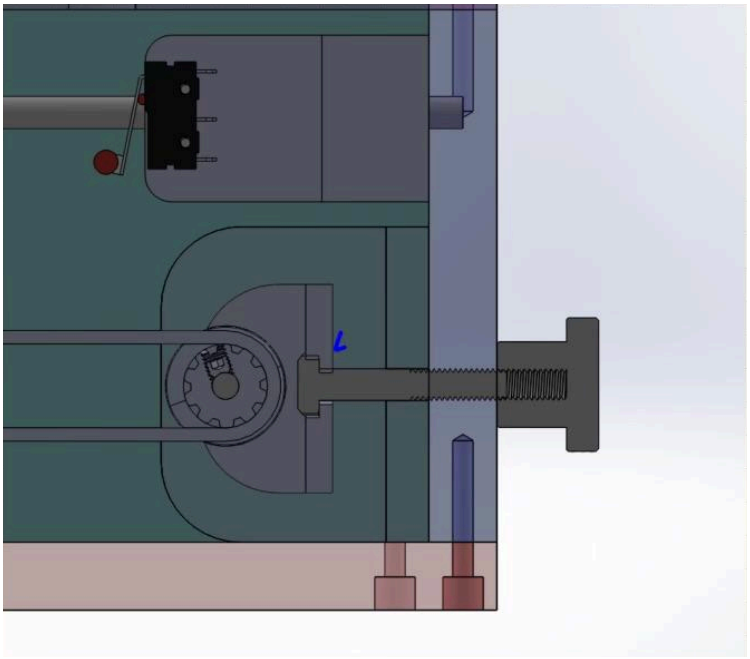
ME102B Drink Dispenser Bill of Material				
Part Number/Type	Quantity	Unit Cost	Total Cost	Source/Link
3D Printer Filament, PLA	1	$29.75	$29.75	https://www.amazon.com/dp/B0C6T3Y6SW?ref=ppx_yo2ov_dt_b_fed_asin_title&th=1
DRV8833 Dual Motor Driver Carrier	3	$9.95	$29.85	https://www.pololu.com/product/2130
Wine Bottles	3	$5.80	$17.39	
Stainless steel spring lock washer for M5 screws	1 pack	$12.13	$12.13	https://www.mcmaster.com/91477A123/
5mm clamping shaft collar	2	$7.21	$14.42	https://www.mcmaster.com/57445K22/
200mm carbon steel rotary shaft	1	$5.58	$5.58	https://www.mcmaster.com/1327K511/
M6 Knurled grip knob	1	$4.39	$4.39	https://www.mcmaster.com/60765K332/
M5 Alloy steel shoulder screw	1	$6.44	$6.44	https://www.mcmaster.com/92981A779/
6mm x 3ft 6061 aluminum rod	4	$3.04	$12.16	https://www.mcmaster.com/4634T32-4634T323/
M6 wood screws	1 pack	$7.99	$7.99	https://www.acehardware.com/departments/hardware/screws-and-anchors/wood-screws/56144
120pcs Multicolored Dupont Wire 40pin Male to Female	1	$6.98	$6.98	https://amazon.com
Potentiometer	1			Borrowed from lab
ESP32	4			From lab kits
PCA9685	2			Borrowed from friend
		TOTAL	$570.60	

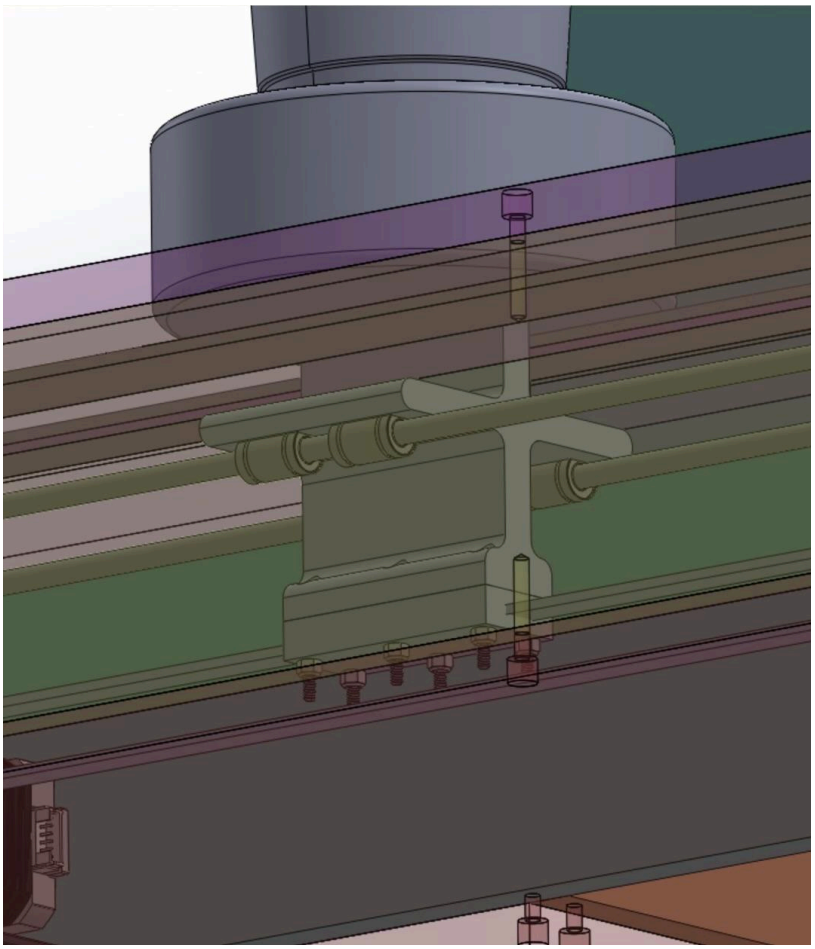
Appendix B: CAD Images











Appendix C: Operating Code

Select Board

showcase_WIP.ino

```ino
#include <Arduino.h>
#include <AccelStepper.h>
#include <ESP32Encoder.h>
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

//#define BIN_1 19
//#define BIN_2 21
//#define LED_PIN 32  //for dc motor control

//Motor 1 PWM channel 0 1
//Motor 2 PWM channel 2 3
//Motor 3 PWM channel 4 5
//Mixer PWM channel 6 7
#define LED_2 12
#define LED_3 13
#define MIXER_LED 19

ESP32Encoder encoder1;
ESP32Encoder encoder2;
ESP32Encoder encoder3;
ESP32Encoder encoder4;

Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver(0x49);

// Pin Definitions
#define BUTTON_A_PIN 4  // Drink A button
//#define BUTTON_B_PIN  // Drink B button
//#define BUTTON_C_PIN 4 // Drink C button
#define POT_PIN 34          // Potentiometer pin
#define LIMIT_SWITCH_PIN 5  // Limit switch pin
//#define LED_1_PIN 26  // LED for portionMultiplier = 1
//#define LED_2_PIN 25  // LED for portionMultiplier = 2
//#define LED_3_PIN 34 // LED for portionMultiplier = 3
#define DIR_PIN 32
#define STEP_PIN 15

// Define motor steps per revolution (e.g., 200 for typical NEMA 17)
#define STEPS_PER_REV 200
#define MICROSTEPPING 1  // Set your microstepping value (e.g., 16 for 1/16 microstep)

// Create AccelStepper object
AccelStepper stepper(AccelStepper::DRIVER, STEP_PIN, DIR_PIN);

int bottle1Pos = -1500;
int bottle2Pos = -2800;
```

Ln 1, Col 1 No board selected

Select Board

showcase_WIP.ino

```ino
int bottle3Pos = -4000;
int bottle4Pos = -5100;

int frotherPos = -6000;




// Variables
//homing
bool homingCompleted = false;

//dc motor pi control
int theta = 0;       // Current position from encoder
int thetaDes = 0;    // Desired position (calculated from input)
int thetaMax = 512;  // Maximum position value (adjust as needed)
int D = 0;           // PWM duty cycle command
int deadZone = 5;
int duty = 0;

// PI Controller Gains
float Kp = 0.5;
float Ki = 0.2;
float errorSum = 0;       // Accumulated error for integral term
const float IMax = 60.0;  // Anti-windup limit for integral term

volatile int count1 = 0;  //count for encoder, PI control
volatile int count2 = 0;
volatile int count3 = 0;
volatile int count4 = 0;
volatile int activeMotor = 0;

volatile bool deltaT = false;
hw_timer_t* timer1 = NULL;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;

// PWM properties
const int freq = 100000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;

//debounce
#define DEBOUNCE_TIME 900  //debounce time in ms
```

Ln 1, Col 1 No board selected

Select Board

showcase_WIP.ino

```
//debounce
#define DEBOUNCE_TIME 900  //debounce time in ms

int potValueChosen = 0;


unsigned long lastPressA = 0;
unsigned long lastPressB = 0;
unsigned long lastPressC = 0;

// State data structure
enum State {
  INIT,
  HOMING,
  IDLE,
  BUTTON_A_PRESSED,
  //BUTTON_B_PRESSED,
  //BUTTON_C_PRESSED,
  BOTTLE_1,
  DRINK_A_BOT_1,
  DRINK_B_BOT_1,
  DRINK_C_BOT_1,
  BOTTLE_2,
  DRINK_A_BOT_2,
  DRINK_B_BOT_2,
  DRINK_C_BOT_2,
  BOTTLE_3,
  DRINK_A_BOT_3,
  DRINK_B_BOT_3,
  DRINK_C_BOT_3,
  MIXER
};
State currentState = INIT;

// Drink selection substates
enum DrinkState { DRINK_A,
                  DRINK_B,
                  DRINK_C };
DrinkState drinkState;

// timer variables
volatile bool TIMERflag = false;
hw_timer_t* timer0 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;

// Flags for Ssnsors
```

Ln 1, Col 1 No board selected

Select Board

showcase_WIP.ino

```
// Flags for Ssnsors
volatile bool BUTTON_A_flag = false;
volatile bool BUTTON_B_flag = false;
volatile bool BUTTON_C_flag = false;
volatile bool LIMIT_SWITCH_flag = false;
// global flag
bool stateChangedFlag = true;  // Set to true when transitioning to a new state, used to make serial prints only print once


//button/switch ISR functions

void IRAM_ATTR isrButtonA() {
  unsigned long currentTime = millis();
  if (currentTime - lastPressA > DEBOUNCE_TIME) {
    BUTTON_A_flag = true;
    lastPressA = currentTime;
    Serial.println("Button A flag set to true");
  }
}

//void IRAM_ATTR isrButtonB() {
//  unsigned long currentTime = millis();
//  if (currentTime - lastPressB > DEBOUNCE_TIME) {
//    BUTTON_B_flag = true;
//    lastPressB = currentTime;
//    Serial.println("Button B flag set to true");
//  }
//}

//void IRAM_ATTR isrButtonC() {
//  unsigned long currentTime = millis();
//  if (currentTime - lastPressC > DEBOUNCE_TIME) {
//    BUTTON_C_flag = true;
//    lastPressC = currentTime;
//   Serial.println("Button C flag set to true");
//  }
//}

void IRAM_ATTR isrLimitSwitch() {
  LIMIT_SWITCH_flag = true;  // Set flag for limit switch
}

void IRAM_ATTR onTime() {

  //Serial.println("");
```

Ln 1, Col 1 No board selected

Select Board

showcase_WIP.ino

```
}

void IRAM_ATTR onTime() {

  //Serial.println("");
  portENTER_CRITICAL_ISR(&timerMux0);
  TIMERflag = true;
  portEXIT_CRITICAL_ISR(&timerMux0);
  timerStop(timer0);
  //Serial.println("Timer Interrupt Triggered!");
}

void IRAM_ATTR onTime1() {

  portENTER_CRITICAL_ISR(&timerMux1);
  switch (activeMotor) {
    case 1:
      count1 = encoder1.getCount();
      encoder1.clearCount();
      break;
    case 2:
      count2 = encoder2.getCount();
      encoder2.clearCount();
      break;
    case 3:
      count3 = encoder3.getCount();
      encoder3.clearCount();
      break;
    case 4:
      count4 = encoder4.getCount();
      encoder4.clearCount();
      break;
    default:
      // No active motor or invalid number, do nothing
      break;
  }

  deltaT = true;
  portEXIT_CRITICAL_ISR(&timerMux1);
}

// Portion Timings (in milliseconds)
unsigned long drinkAPortionTime = 3000;
unsigned long drinkBPortionTime = 800;
unsigned long drinkCPortionTime = 1000;

// Potentiometer Value
```

Ln 1, Col 1 × No board selected

Select Board

showcase_WIP.ino

```
// Potentiometer Value
int portionMultiplier = 1;

// Functions
void startTimer(unsigned long time);
bool checkTimerFlag();
bool checkbuttonAPressed();
bool checkbuttonBPressed();
bool checkbuttonCPressed();
bool checkStateChanged();
void stopTimer();
void setPositionRef();
void promptUserDrinkType();
void startPotTimer();
int readPotentiometer();
void setDrinkType(DrinkState chosenDrink);
void dispenseStopStart(int direction, int angle);
void moveToPosition(int position);
void mixDrink();
void updateLEDs();
void IRAM_ATTR onTime();
void IRAM_ATTR isrLimitSwitch();
void IRAM_ATTR isrButtonA();
void IRAM_ATTR isrButtonB();
void IRAM_ATTR isrButtonC();
void homeStepper();
long angleToSteps(float angle);

void setup() {

  Serial.begin(115200);
  //stateChangedFlag = true;
  unsigned long startupTime = millis();
  while (millis() - startupTime < 1000) {
    // Non-blocking  delay for stabilization, peripherals
  }

  // Initialize I2C communication on SDA (GPIO 21) and SCL (GPIO 22)
  Wire.begin(22, 20);

  // Initialize PCA9685
  pwm.begin();
  pwm.setPWMFreq(1500); // Set frequency to 1kHz for LED control
  Serial.println("PCA9685 Initialized.");

  pwm.setPWM(0, 0, 0);
```

Ln 1, Col 1 × No board selected

Select Board

showcase_WIP.ino

```
  pwm.setPWM(0, 0, 0);
  pwm.setPWM(1, 0, 0);
  pwm.setPWM(2, 0, 0);
  pwm.setPWM(3, 0, 0);
  pwm.setPWM(4, 0, 0);
  pwm.setPWM(5, 0, 0);
  pwm.setPWM(6, 0, 0);
  pwm.setPWM(7, 0, 0);

  pinMode(BUTTON_A_PIN, INPUT_PULLUP);
  //pinMode(BUTTON_B_PIN, INPUT_PULLUP);
  //pinMode(BUTTON_C_PIN, INPUT_PULLUP);
  pinMode(POT_PIN, INPUT);
  pinMode(LIMIT_SWITCH_PIN, INPUT);
  //pinMode(LED_1_PIN, OUTPUT);
  //pinMode(LED_2_PIN, OUTPUT);
  //pinMode(LED_3_PIN, OUTPUT);
 // pinMode(LED_PIN, OUTPUT);
  //digitalWrite(LED_PIN, LOW);
pinMode(LED_2, OUTPUT);
pinMode(LED_3, OUTPUT);
pinMode(MIXER_LED, OUTPUT);

  ESP32Encoder::useInternalWeakPullResistors = puType::up;

  encoder1.attachHalfQuad(21, 7);
  encoder1.setCount(0);
  encoder2.attachHalfQuad(27, 33);
  encoder2.setCount(0);
  encoder3.attachHalfQuad(13, 12);
  encoder3.setCount(0);
  encoder4.attachHalfQuad(8, 14);
  encoder4.setCount(0);

  //ledcSetup(ledChannel_1, freq, resolution);
  //ledcAttachPin(BIN_1, ledChannel_1);
  //ledcSetup(ledChannel_2, freq, resolution);
  //ledcAttachPin(BIN_2, ledChannel_2);

  // Correct Stepper motor positions to be in terms of steps
  bottle1Pos = angleToSteps(-1500);
  bottle2Pos = angleToSteps(-2800);
  bottle3Pos = angleToSteps(-4000);
  bottle4Pos = angleToSteps(-5100);
```

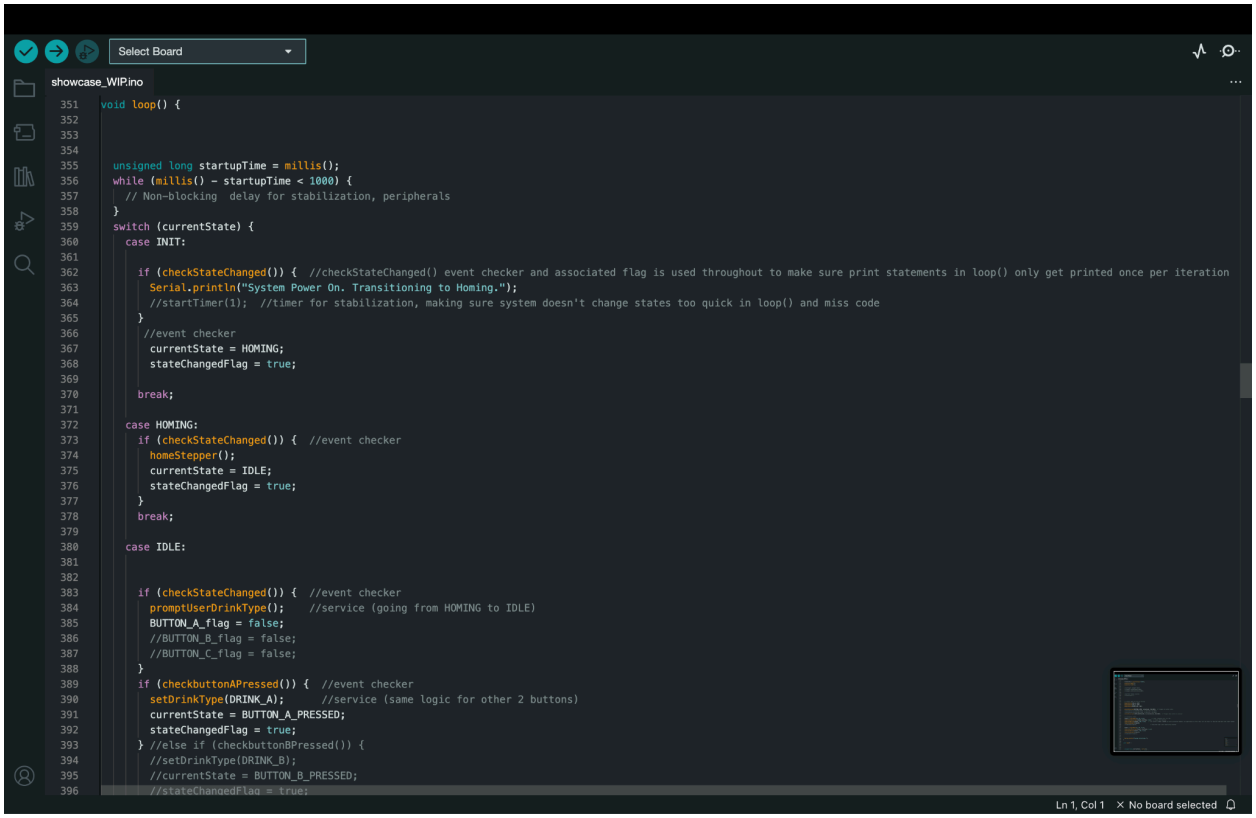
Ln 1, Col 1 × No board selected

Select Board

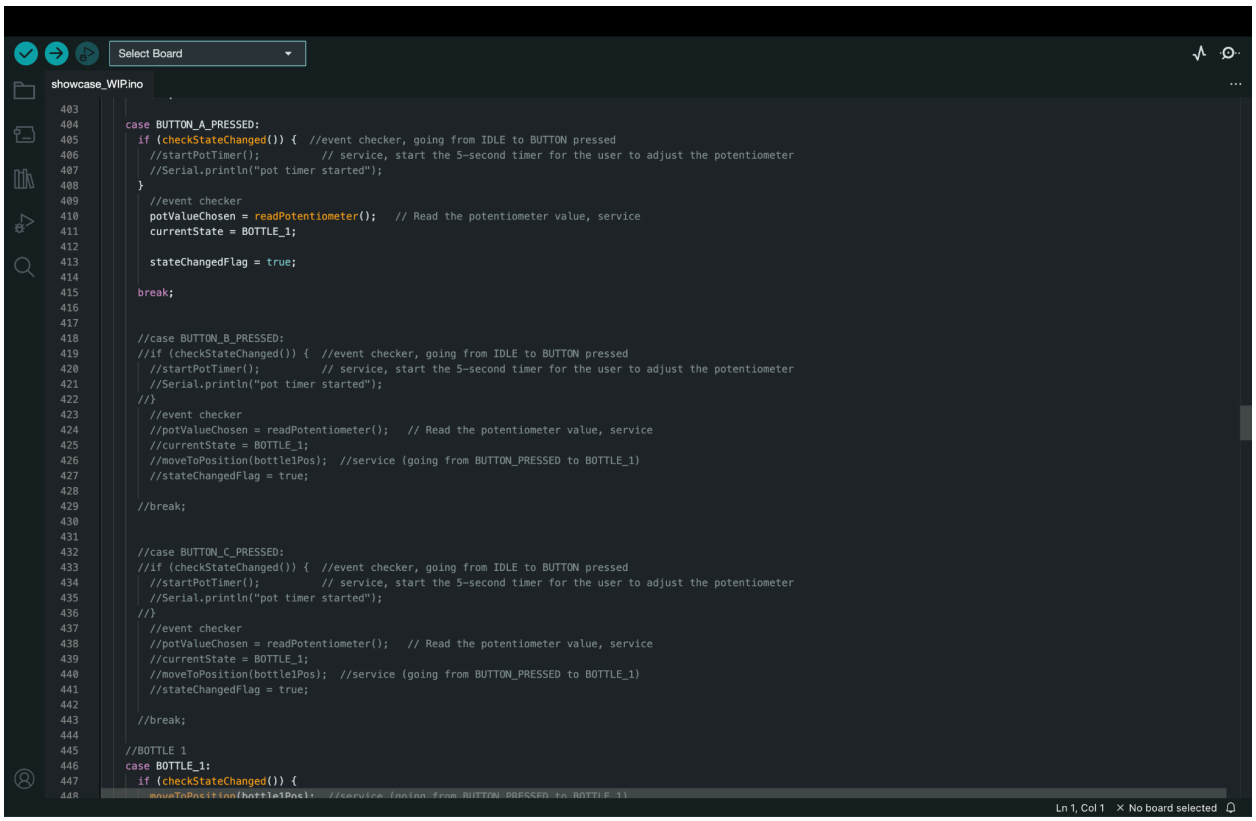
showcase_WIP.ino

```
  frotherPos = angleToSteps(-6000);
  disableCore0WDT();
  disableCore1WDT();

  //configure stepper motor
  //stepper.setMaxSpeed(1200);
  //stepper.setAcceleration(500);

  //perform homing routine
  //homeStepper();



  // Ensure LEDs are off at startup
  digitalWrite(LED_2, LOW);
  digitalWrite(LED_3, LOW);
  digitalWrite(MIXER_LED, LOW);

  attachInterrupt(BUTTON_A_PIN, isrButtonA, FALLING);  // Trigger on button press
  //attachInterrupt(BUTTON_B_PIN, isrButtonB, FALLING);
  //attachInterrupt(BUTTON_C_PIN, isrButtonC, FALLING);
  attachInterrupt(LIMIT_SWITCH_PIN, isrLimitSwitch, FALLING);  // Trigger when switch is pressed

  // Initialize Timer
  timer0 = timerBegin(0, 80, true);           // Timer frequency set to 1 MHz
  timerAttachInterrupt(timer0, &onTime, true);  // Attach ISR function
  timerAlarmWrite(timer0, 100, true);    // Set alarm to 500ms (500000 us) with unlimited repeats, no signficance to this time, will be reset for desired time each time timer needed
  timerAlarmEnable(timer0);
  //timerStop(timer0);                      // Stop the timer until explicitly started

  timer1 = timerBegin(1, 80, true);
  timerAttachInterrupt(timer1, &onTime1, true);
  timerAlarmWrite(timer1, 100, true);
  timerAlarmEnable(timer1);
  //timerStart(timer1);


  Serial.println("System Initialized.");
}

void loop() {



  unsigned long startupTime = millis();
```

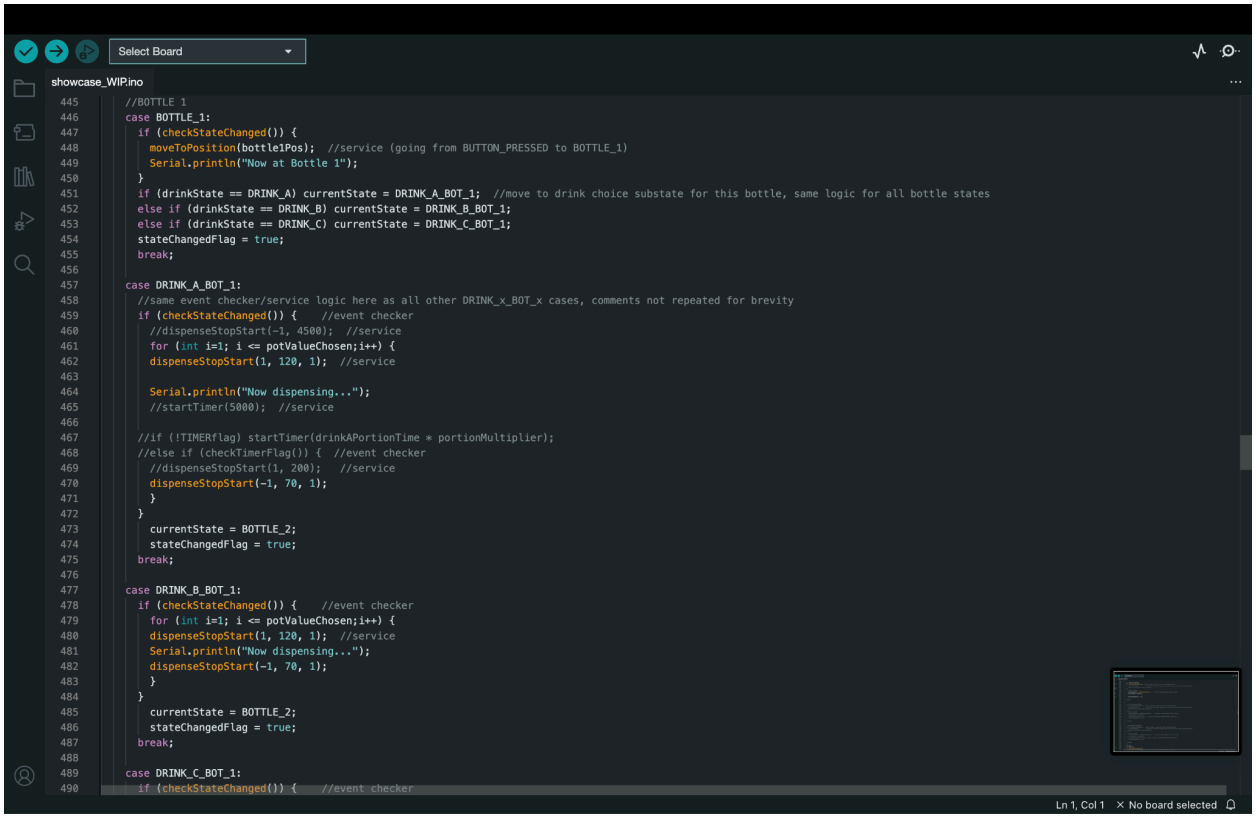
Ln 1, Col 1 × No board selected



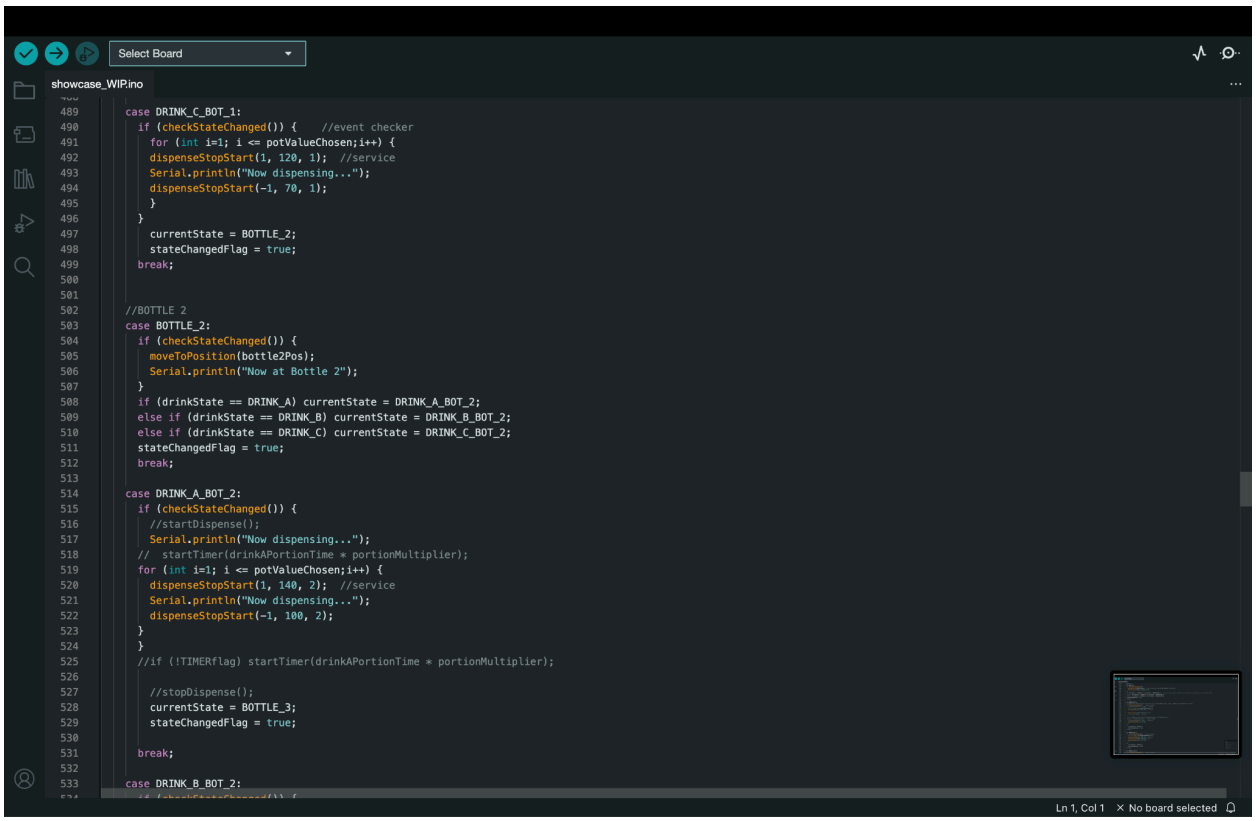

```
void loop() {



  unsigned long startupTime = millis();
  while (millis() - startupTime < 1000) {
    // Non-blocking  delay for stabilization, peripherals
  }
  switch (currentState) {
    case INIT:

      if (checkStateChanged()) {  //checkStateChanged() event checker and associated flag is used throughout to make sure print statements in loop() only get printed once per iteration
        Serial.println("System Power On. Transitioning to Homing.");
        //startTimer(1);  //timer for stabilization, making sure system doesn't change states too quick in loop() and miss code
      }
       //event checker
        currentState = HOMING;
        stateChangedFlag = true;

      break;

    case HOMING:
      if (checkStateChanged()) {  //event checker
        homeStepper();
        currentState = IDLE;
        stateChangedFlag = true;
      }
      break;

    case IDLE:


      if (checkStateChanged()) {  //event checker
        promptUserDrinkType();    //service (going from HOMING to IDLE)
        BUTTON_A_flag = false;
        //BUTTON_B_flag = false;
        //BUTTON_C_flag = false;
      }
      if (checkbuttonAPressed()) {  //event checker
        setDrinkType(DRINK_A);      //service (same logic for other 2 buttons)
        currentState = BUTTON_A_PRESSED;
        stateChangedFlag = true;
      } //else if (checkbuttonBPressed()) {
        //setDrinkType(DRINK_B);
        //currentState = BUTTON_B_PRESSED;
        //stateChangedFlag = true;
```



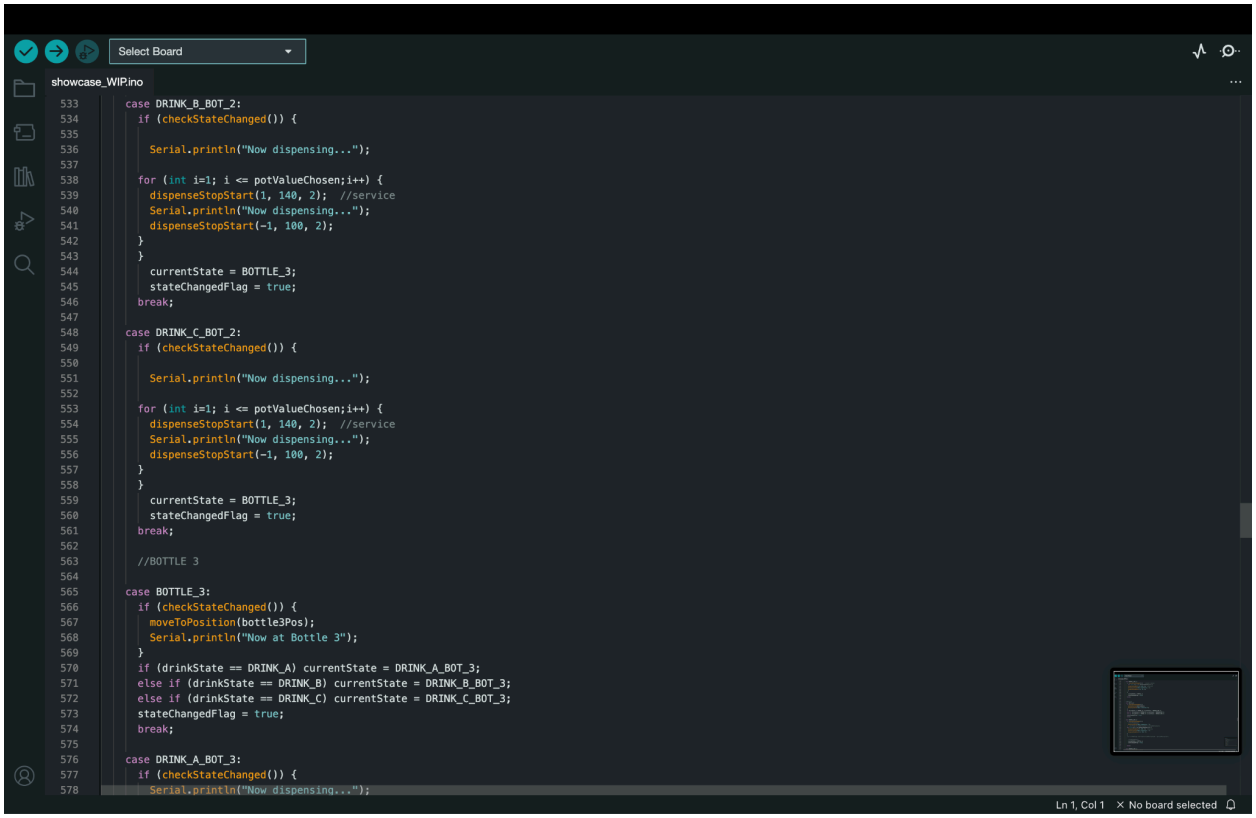

```
    case BUTTON_A_PRESSED:
      if (checkStateChanged()) {  //event checker, going from IDLE to BUTTON pressed
        //startPotTimer();          // service, start the 5-second timer for the user to adjust the potentiometer
        //Serial.println("pot timer started");
      }
        //event checker
        potValueChosen = readPotentiometer();   // Read the potentiometer value, service
        currentState = BOTTLE_1;

        stateChangedFlag = true;

      break;


    //case BUTTON_B_PRESSED:
    //if (checkStateChanged()) {  //event checker, going from IDLE to BUTTON pressed
      //startPotTimer();          // service, start the 5-second timer for the user to adjust the potentiometer
      //Serial.println("pot timer started");
    //}
      //event checker
      //potValueChosen = readPotentiometer();   // Read the potentiometer value, service
      //currentState = BOTTLE_1;
      //moveToPosition(bottle1Pos);  //service (going from BUTTON_PRESSED to BOTTLE_1)
      //stateChangedFlag = true;

    //break;


    //case BUTTON_C_PRESSED:
    //if (checkStateChanged()) {  //event checker, going from IDLE to BUTTON pressed
      //startPotTimer();          // service, start the 5-second timer for the user to adjust the potentiometer
      //Serial.println("pot timer started");
    //}
      //event checker
      //potValueChosen = readPotentiometer();   // Read the potentiometer value, service
      //currentState = BOTTLE_1;
      //moveToPosition(bottle1Pos);  //service (going from BUTTON_PRESSED to BOTTLE_1)
      //stateChangedFlag = true;

    //break;

  //BOTTLE 1
  case BOTTLE_1:
    if (checkStateChanged()) {
      moveToPosition(bottle1Pos);  //service (going from BUTTON_PRESSED to BOTTLE_1)
```



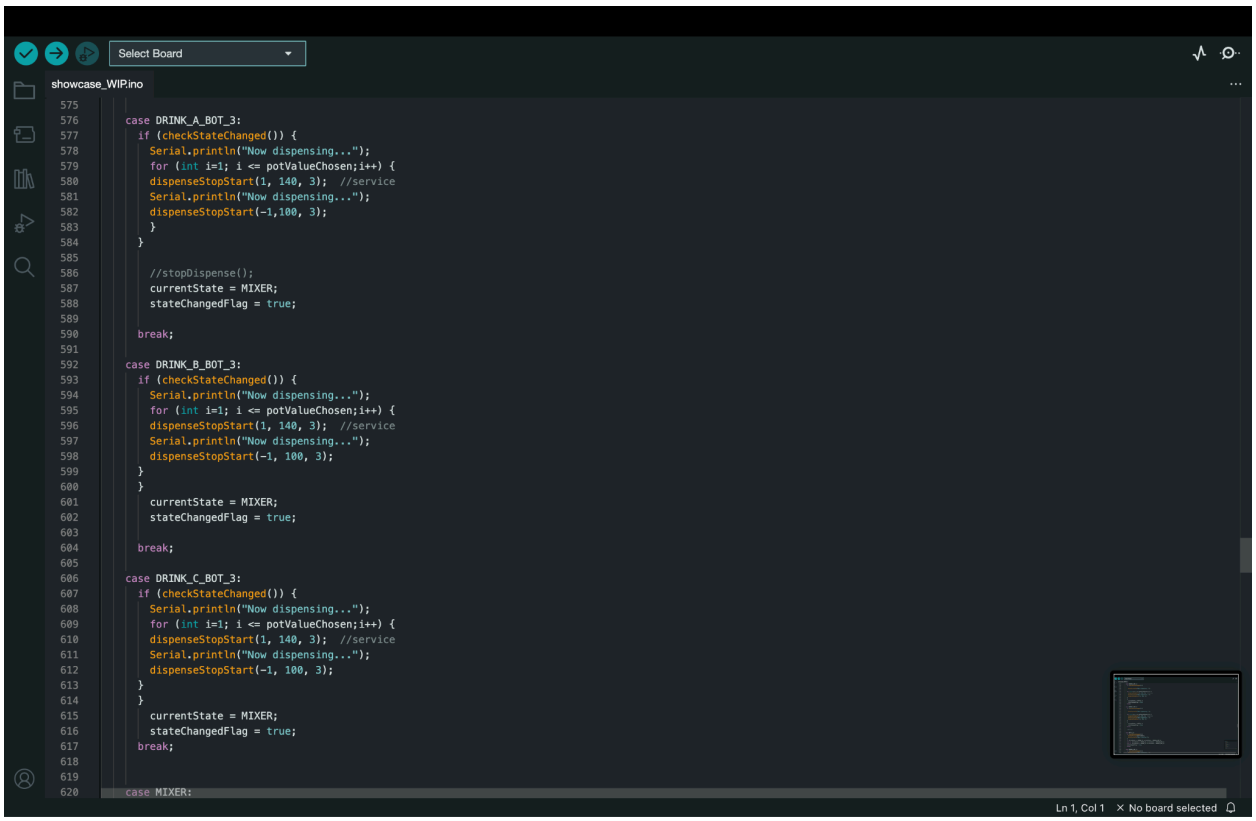

```
//BOTTLE 1
case BOTTLE_1:
  if (checkStateChanged()) {
    moveToPosition(bottle1Pos);  //service (going from BUTTON_PRESSED to BOTTLE_1)
    Serial.println("Now at Bottle 1");
  }
  if (drinkState == DRINK_A) currentState = DRINK_A_BOT_1;  //move to drink choice substate for this bottle, same logic for all bottle states
  else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_1;
  else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_1;
  stateChangedFlag = true;
  break;

case DRINK_A_BOT_1:
  //same event checker/service logic here as all other DRINK_x_BOT_x cases, comments not repeated for brevity
  if (checkStateChanged()) {    //event checker
    //dispenseStopStart(-1, 4500);  //service
    for (int i=1; i <= potValueChosen;i++) {
    dispenseStopStart(1, 120, 1);  //service

    Serial.println("Now dispensing...");
    //startTimer(5000);  //service

  //if (!TIMERflag) startTimer(drinkAPortionTime * portionMultiplier);
  //else if (checkTimerFlag()) {  //event checker
    //dispenseStopStart(1, 200);   //service
    dispenseStopStart(-1, 70, 1);
    }
  }
    currentState = BOTTLE_2;
    stateChangedFlag = true;
  break;

case DRINK_B_BOT_1:
  if (checkStateChanged()) {    //event checker
    for (int i=1; i <= potValueChosen;i++) {
    dispenseStopStart(1, 120, 1);  //service
    Serial.println("Now dispensing...");
    dispenseStopStart(-1, 70, 1);
    }
  }
    currentState = BOTTLE_2;
    stateChangedFlag = true;
  break;

case DRINK_C_BOT_1:
  if (checkStateChanged()) {    //event checker
```




```
case DRINK_C_BOT_1:
  if (checkStateChanged()) {    //event checker
    for (int i=1; i <= potValueChosen;i++) {
    dispenseStopStart(1, 120, 1);  //service
    Serial.println("Now dispensing...");
    dispenseStopStart(-1, 70, 1);
    }
  }
    currentState = BOTTLE_2;
    stateChangedFlag = true;
  break;


//BOTTLE 2
case BOTTLE_2:
  if (checkStateChanged()) {
    moveToPosition(bottle2Pos);
    Serial.println("Now at Bottle 2");
  }
  if (drinkState == DRINK_A) currentState = DRINK_A_BOT_2;
  else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_2;
  else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_2;
  stateChangedFlag = true;
  break;

case DRINK_A_BOT_2:
  if (checkStateChanged()) {
    //startDispense();
    Serial.println("Now dispensing...");
  //  startTimer(drinkAPortionTime * portionMultiplier);
  for (int i=1; i <= potValueChosen;i++) {
    dispenseStopStart(1, 140, 2);  //service
    Serial.println("Now dispensing...");
    dispenseStopStart(-1, 100, 2);
  }
  }
  //if (!TIMERflag) startTimer(drinkAPortionTime * portionMultiplier);

    //stopDispense();
    currentState = BOTTLE_3;
    stateChangedFlag = true;

  break;

case DRINK_B_BOT_2:
```




```
case DRINK_B_BOT_2:
  if (checkStateChanged()) {

    Serial.println("Now dispensing...");

  for (int i=1; i <= potValueChosen;i++) {
    dispenseStopStart(1, 140, 2);  //service
    Serial.println("Now dispensing...");
    dispenseStopStart(-1, 100, 2);
  }
  }
    currentState = BOTTLE_3;
    stateChangedFlag = true;
  break;

case DRINK_C_BOT_2:
  if (checkStateChanged()) {

    Serial.println("Now dispensing...");

  for (int i=1; i <= potValueChosen;i++) {
    dispenseStopStart(1, 140, 2);  //service
    Serial.println("Now dispensing...");
    dispenseStopStart(-1, 100, 2);
  }
  }
    currentState = BOTTLE_3;
    stateChangedFlag = true;
  break;

  //BOTTLE 3

case BOTTLE_3:
  if (checkStateChanged()) {
    moveToPosition(bottle3Pos);
    Serial.println("Now at Bottle 3");
  }
  if (drinkState == DRINK_A) currentState = DRINK_A_BOT_3;
  else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_3;
  else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_3;
  stateChangedFlag = true;
  break;

case DRINK_A_BOT_3:
  if (checkStateChanged()) {
    Serial.println("Now dispensing...");
```




```
case DRINK_A_BOT_3:
  if (checkStateChanged()) {
    Serial.println("Now dispensing...");
    for (int i=1; i <= potValueChosen;i++) {
    dispenseStopStart(1, 140, 3);  //service
    Serial.println("Now dispensing...");
    dispenseStopStart(-1,100, 3);
    }
  }

    //stopDispense();
    currentState = MIXER;
    stateChangedFlag = true;

  break;

case DRINK_B_BOT_3:
  if (checkStateChanged()) {
    Serial.println("Now dispensing...");
    for (int i=1; i <= potValueChosen;i++) {
    dispenseStopStart(1, 140, 3);  //service
    Serial.println("Now dispensing...");
    dispenseStopStart(-1, 100, 3);
  }
  }
    currentState = MIXER;
    stateChangedFlag = true;

  break;

case DRINK_C_BOT_3:
  if (checkStateChanged()) {
    Serial.println("Now dispensing...");
    for (int i=1; i <= potValueChosen;i++) {
    dispenseStopStart(1, 140, 3);  //service
    Serial.println("Now dispensing...");
    dispenseStopStart(-1, 100, 3);
  }
  }
    currentState = MIXER;
    stateChangedFlag = true;
  break;


case MIXER:
```

Select Board

showcase_WIP.ino

```
    case MIXER:
      if (checkStateChanged()) {  //event checker             //service

        moveToPosition(frotherPos);
        Serial.println("Now at Mixer");

        dispenseStopStart(1, 120, 4);
        mixDrink();
        dispenseStopStart(-1, 120, 4);

        Serial.println("Resetting to Idle.");
      }
      moveToPosition(0);
      currentState = IDLE;
      stateChangedFlag = true;
      break;
  }
  }

// Functions
void startTimer(unsigned long time) {
  Serial.println("Starting Timer...");
  timerWrite(timer0, 0);                          // Reset the timer count
  timerAlarmWrite(timer0, time * 1000, false);  // Set alarm duration (ms to us, no auto-reload)
  TIMERflag = false;                              // Reset timer flag
  timerAlarmEnable(timer0);
  timerStart(timer0);

  //Serial.println("Timer Started"); // Debugging print
}

bool checkTimerFlag() {
  bool flagState = false; // Local variable to hold the flag state

  portENTER_CRITICAL_ISR(&timerMux0); // Enter critical section
  if (TIMERflag) {
    TIMERflag = false;    // Reset the flag
    flagState = true;     // Mark that the flag was set
  }
  portEXIT_CRITICAL_ISR(&timerMux0); // Exit critical section

  return flagState;       // Return whether the flag was set
}

void stopTimer() {
```

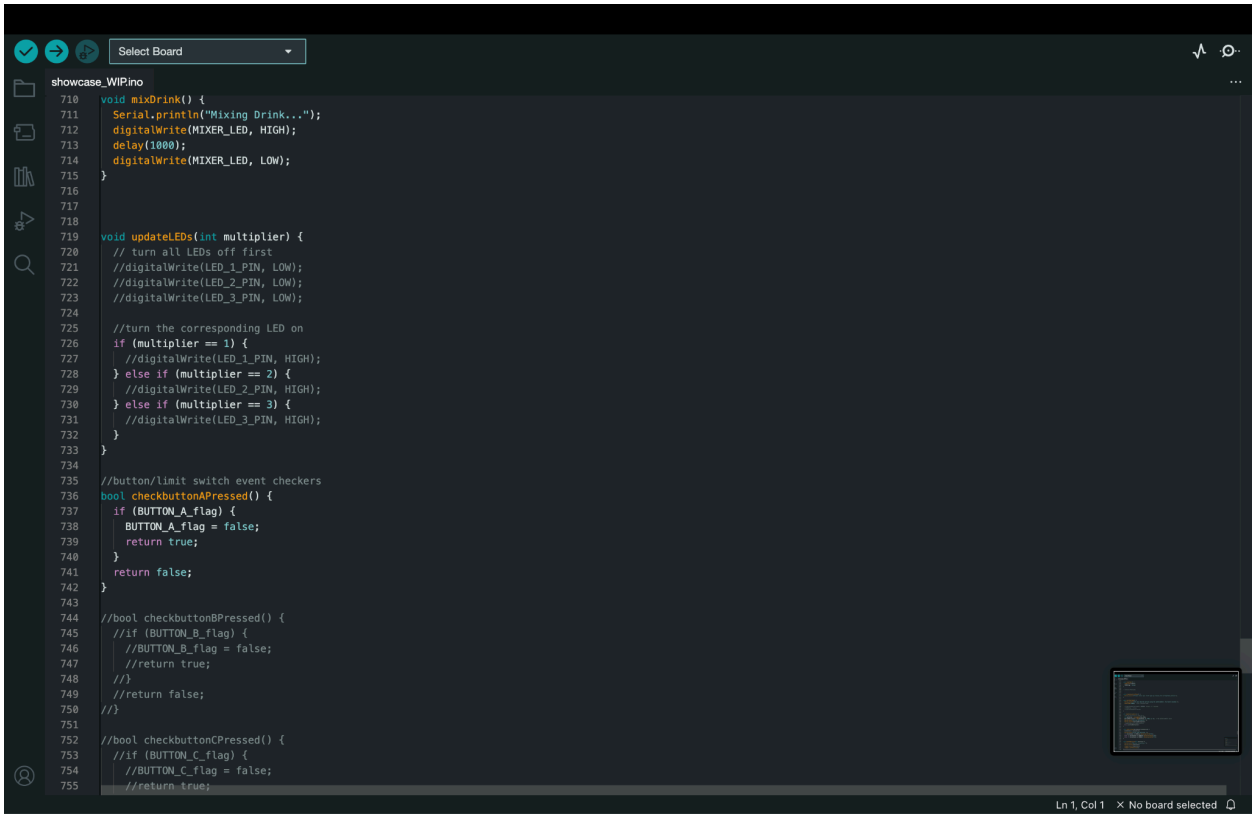
Ln 1, Col 1 No board selected

Select Board

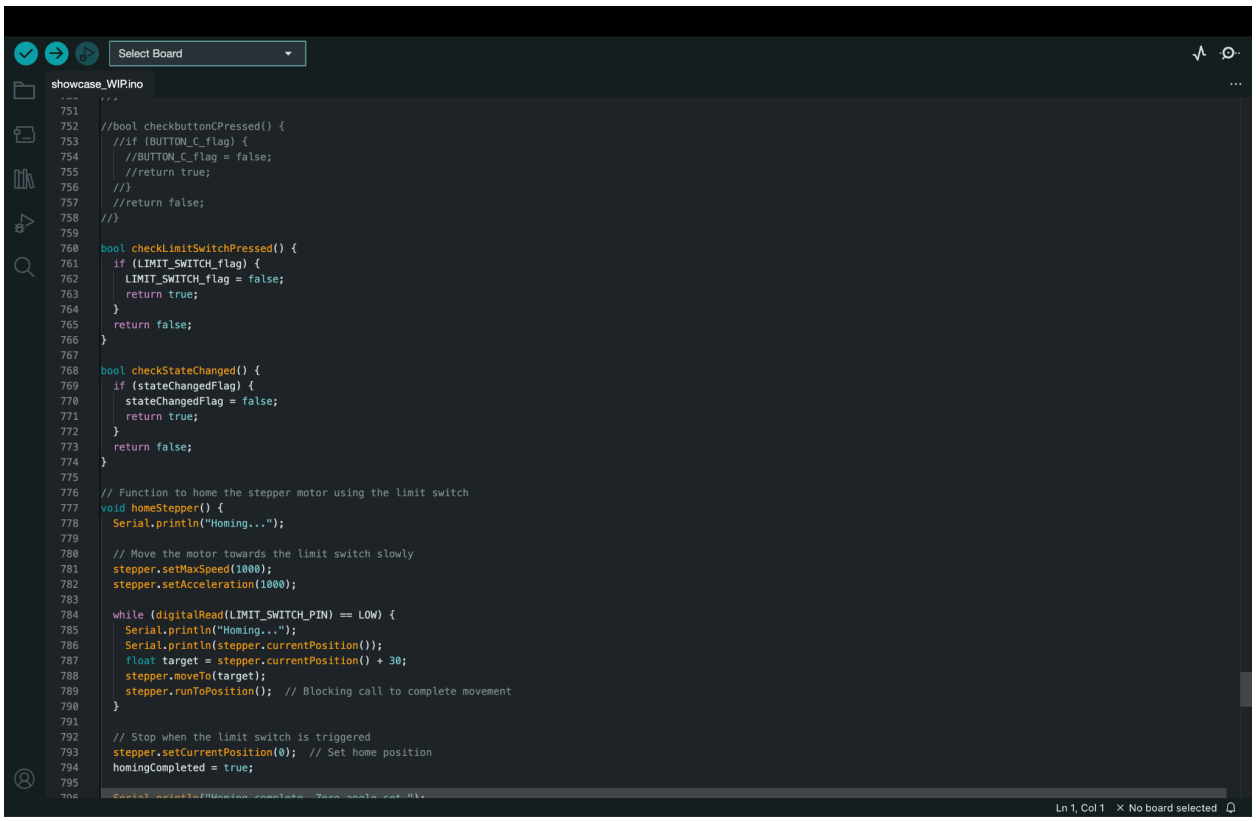
showcase_WIP.ino

```
void stopTimer() {
  timerStop(timer0);
  TIMERflag = false;
}

// Service Functions


void promptUserDrinkType() {
  Serial.println("Please select your drink type by pressing the corresponding button:");
}

void startPotTimer() {
  Serial.println("Set your desired portion using the potentiometer. You have 5 seconds.");
  startTimer(5000);  //set 5 second timer

  //timerAlarmWrite(timer0, 5000000, false); // 5 seconds
  //TIMERflag = false;
  //timerAlarmEnable(timer0);
}

int readPotentiometer() {
  //Serial.println("in read pot");
  int potValue = analogRead(POT_PIN);
  portionMultiplier = map(potValue, 0, 4095, 1, 3);  // Map potentiometer value
  Serial.print("Portion Multiplier: ");
  Serial.println(portionMultiplier);
  //updateLEDs(portionMultiplier);
 return portionMultiplier;
}

void setDrinkType(DrinkState chosenDrink) {
  drinkState = chosenDrink;
  Serial.print("Drink Type Selected: ");
  if (drinkState == DRINK_A) Serial.println("A");
  else if (drinkState == DRINK_B) Serial.println("B");
  else if (drinkState == DRINK_C) Serial.println("C");
}

void moveToPosition(int position) {
  Serial.print("Moving to position: ");
  Serial.println(position);
  stepper.moveTo(position);
  stepper.runToPosition();
}
```

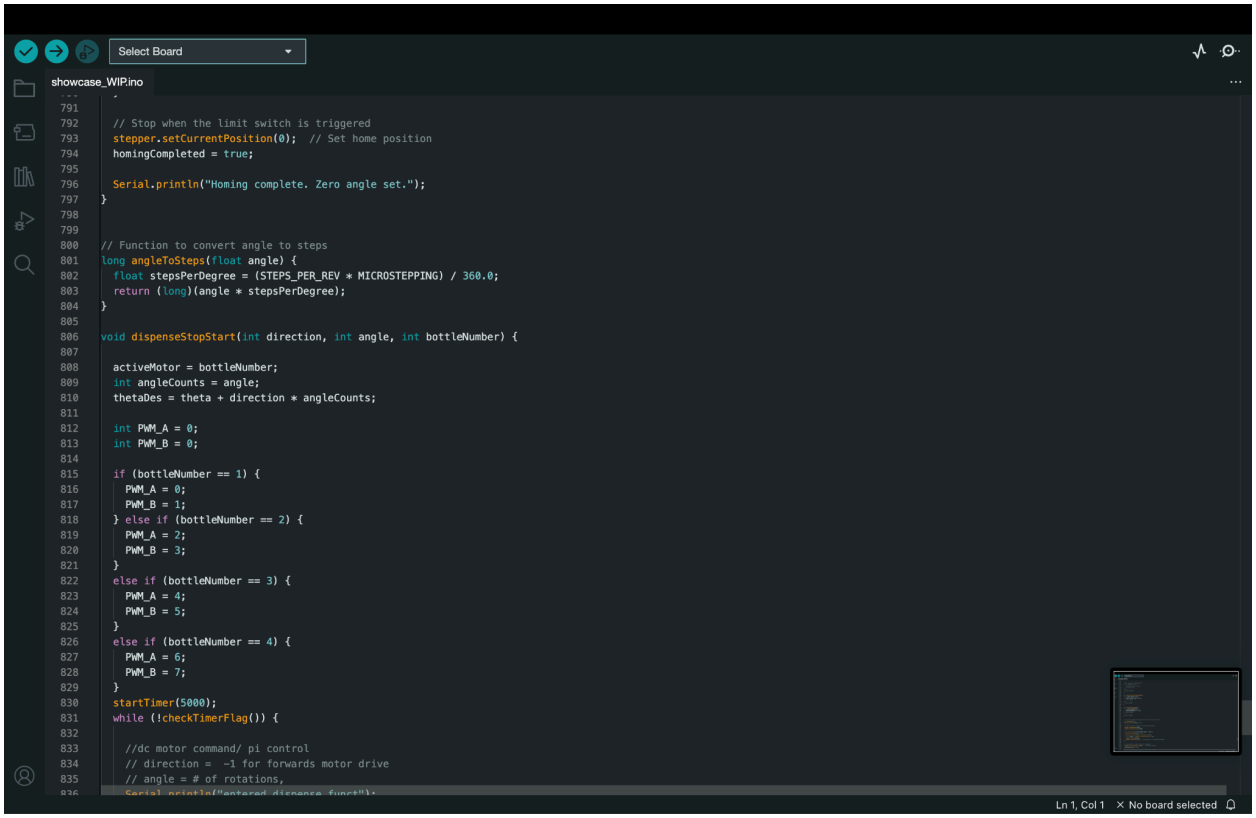
Ln 1, Col 1 No board selected




```
void mixDrink() {
  Serial.println("Mixing Drink...");
  digitalWrite(MIXER_LED, HIGH);
  delay(1000);
  digitalWrite(MIXER_LED, LOW);
}



void updateLEDs(int multiplier) {
  // turn all LEDs off first
  //digitalWrite(LED_1_PIN, LOW);
  //digitalWrite(LED_2_PIN, LOW);
  //digitalWrite(LED_3_PIN, LOW);

  //turn the corresponding LED on
  if (multiplier == 1) {
    //digitalWrite(LED_1_PIN, HIGH);
  } else if (multiplier == 2) {
    //digitalWrite(LED_2_PIN, HIGH);
  } else if (multiplier == 3) {
    //digitalWrite(LED_3_PIN, HIGH);
  }
}

//button/limit switch event checkers
bool checkbuttonAPressed() {
  if (BUTTON_A_flag) {
    BUTTON_A_flag = false;
    return true;
  }
  return false;
}

//bool checkbuttonBPressed() {
  //if (BUTTON_B_flag) {
    //BUTTON_B_flag = false;
    //return true;
  //}
  //return false;
//}

//bool checkbuttonCPressed() {
  //if (BUTTON_C_flag) {
    //BUTTON_C_flag = false;
    //return true;
```




```
//bool checkbuttonCPressed() {
  //if (BUTTON_C_flag) {
    //BUTTON_C_flag = false;
    //return true;
  //}
  //return false;
//}

bool checkLimitSwitchPressed() {
  if (LIMIT_SWITCH_flag) {
    LIMIT_SWITCH_flag = false;
    return true;
  }
  return false;
}

bool checkStateChanged() {
  if (stateChangedFlag) {
    stateChangedFlag = false;
    return true;
  }
  return false;
}

// Function to home the stepper motor using the limit switch
void homeStepper() {
  Serial.println("Homing...");

  // Move the motor towards the limit switch slowly
  stepper.setMaxSpeed(1000);
  stepper.setAcceleration(1000);

  while (digitalRead(LIMIT_SWITCH_PIN) == LOW) {
    Serial.println("Homing...");
    Serial.println(stepper.currentPosition());
    float target = stepper.currentPosition() + 30;
    stepper.moveTo(target);
    stepper.runToPosition();  // Blocking call to complete movement
  }

  // Stop when the limit switch is triggered
  stepper.setCurrentPosition(0);  // Set home position
  homingCompleted = true;

  Serial.println("Homing complete. Zero angle set.");
```


Select Board
showcase_WIP.ino
// Stop when the limit switch is triggered
stepper.setCurrentPosition(0); // Set home position
homingCompleted = true;
Serial.println("Homing complete. Zero angle set.");
}
// Function to convert angle to steps
long angleToSteps(float angle) {
float stepsPerDegree = (STEPS_PER_REV * MICROSTEPPING) / 360.0;
return (long)(angle * stepsPerDegree);
}
void dispenseStopStart(int direction, int angle, int bottleNumber) {
activeMotor = bottleNumber;
int angleCounts = angle;
thetaDes = theta + direction * angleCounts;
int PWM_A = 0;
int PWM_B = 0;
if (bottleNumber == 1) {
PWM_A = 0;
PWM_B = 1;
} else if (bottleNumber == 2) {
PWM_A = 2;
PWM_B = 3;
}
else if (bottleNumber == 3) {
PWM_A = 4;
PWM_B = 5;
}
else if (bottleNumber == 4) {
PWM_A = 6;
PWM_B = 7;
}
startTimer(5000);
while (!checkTimerFlag()) {
//dc motor command/ pi control
// direction = -1 for forwards motor drive
// angle = # of rotations,
Ln 1, Col 1
No board selected

Select Board

showcase_WIP.ino

```
836     Serial.println("entered dispense funct");
837
838     if (deltaT) {
839       portENTER_CRITICAL(&timerMux1);
840       deltaT = false;
841       int currentCount = 0;
842
843       switch (activeMotor) {
844         case 1: currentCount = count1; break;
845         case 2: currentCount = count2; break;
846         case 3: currentCount = count3; break;
847         case 4: currentCount = count4; break;
848       }
849       portEXIT_CRITICAL(&timerMux1);
850
851       // Update position
852       theta += currentCount;
853
854       Serial.print("Theta: ");
855       Serial.println(theta);
856
857       // A6 CONTROL SECTION: PI Control with Anti-Windup
858       int error = thetaDes - theta;
859
860
861
862       // Accumulate the error for the integral term
863       errorSum += error;
864
865       // Anti-Windup: Clamp the integral sum to prevent wind-up
866       if (errorSum > IMax) {
867         errorSum = IMax;
868       } else if (errorSum < -IMax) {
869         errorSum = -IMax;
870       }
871
872       // PI control equation: D = Kp * error + Ki * errorSum
873       D = Kp * error + Ki * errorSum;
874
875       // Clamp the PWM command to the allowed range
876       if (D > MAX_PWM_VOLTAGE) {
877         D = MAX_PWM_VOLTAGE;
878       } else if (D < -MAX_PWM_VOLTAGE) {
879         D = -MAX_PWM_VOLTAGE;
880       }
881
```

Ln 1, Col 1 ✕ No board selected

Select Board

showcase_WIP.ino

```
870       }
871
872       // PI control equation: D = Kp * error + Ki * errorSum
873       D = Kp * error + Ki * errorSum;
874
875       // Clamp the PWM command to the allowed range
876       if (D > MAX_PWM_VOLTAGE) {
877         D = MAX_PWM_VOLTAGE;
878       } else if (D < -MAX_PWM_VOLTAGE) {
879         D = -MAX_PWM_VOLTAGE;
880       }
881
882       duty = map(abs(D), 0, MAX_PWM_VOLTAGE, 0, 4095);
883
884       // Control motor direction and speed based on D
885       if (D > 0) {
886
887         Serial.print("D > 0: ");
888         Serial.println(D);
889           pwm.setPWM(PWM_A, 0, duty);
890           pwm.setPWM(PWM_B, 0, 0);
891
892       } else if (D < 0) {
893
894         Serial.print("D < 0: ");
895         Serial.println(D);
896
897         pwm.setPWM(PWM_A, 0, 0);
898         pwm.setPWM(PWM_B, 0, duty);
899
900       } else {
901
902         Serial.print("D = 0: ");
903         Serial.println(D);
904         pwm.setPWM(PWM_A, 0, 0);
905         pwm.setPWM(PWM_B, 0, 0);
906       }
907     }
908
909     yield();
910   }
911         pwm.setPWM(PWM_A, 0, 0);
912         pwm.setPWM(PWM_B, 0, 0);
913 }
914
```

Ln 1, Col 1 ✕ No board selected