



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.

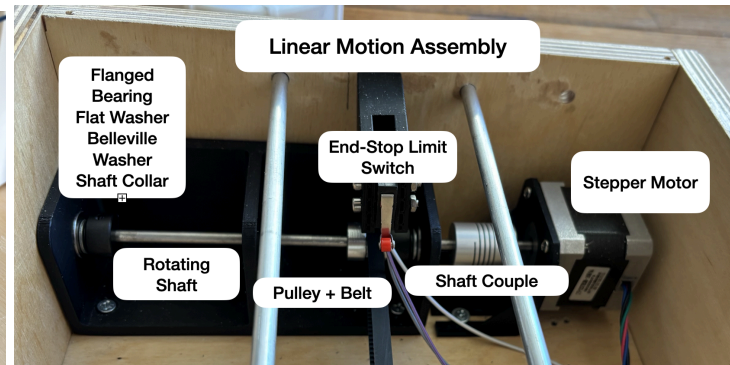
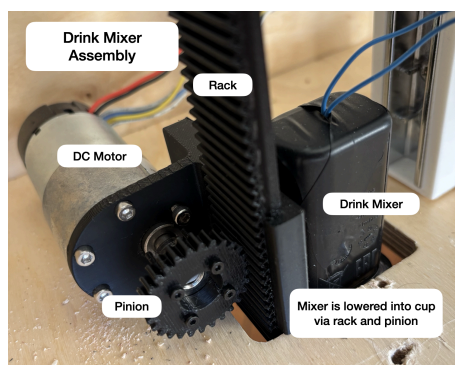
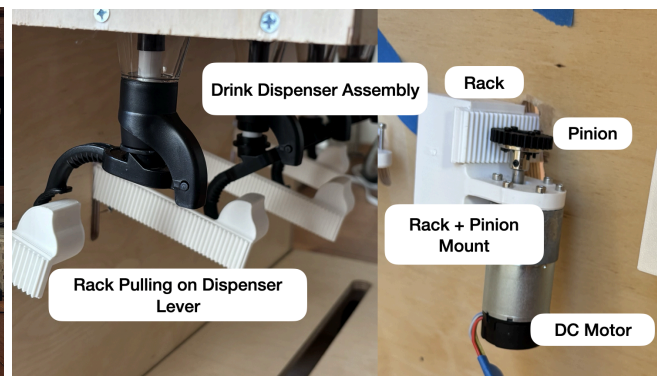
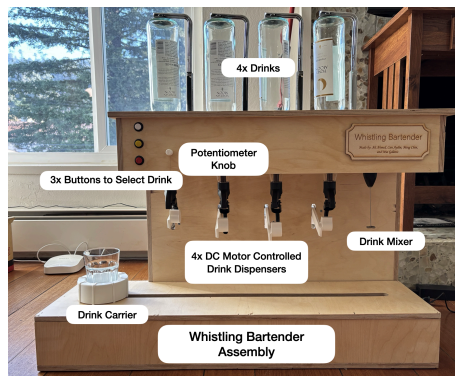
High-level Strategy: Our final project is composed of three main systems– dispensing, mixing, and a linear motion system. We designed and manufactured a 0.5-inch plywood housing that holds all our components together for a clean, professional look. The system is controlled with an ESP32 and a PCA9685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

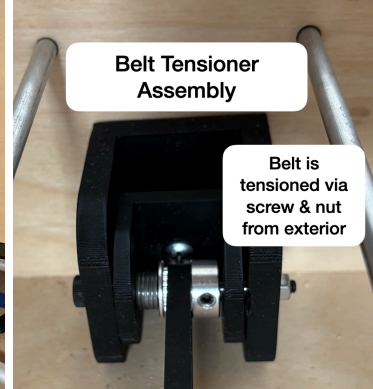
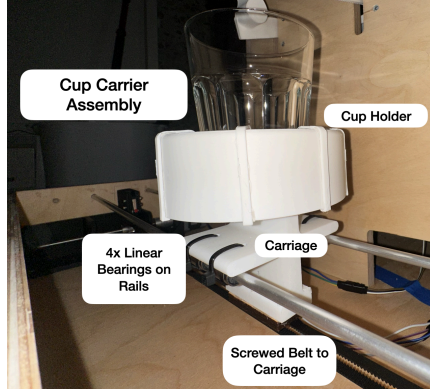
Dispensing: This system is composed of four DC brush motors attached to a 3D printed rack and pinion mechanism that would pull a lever of the drink dispensers to dispense the liquid.

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.

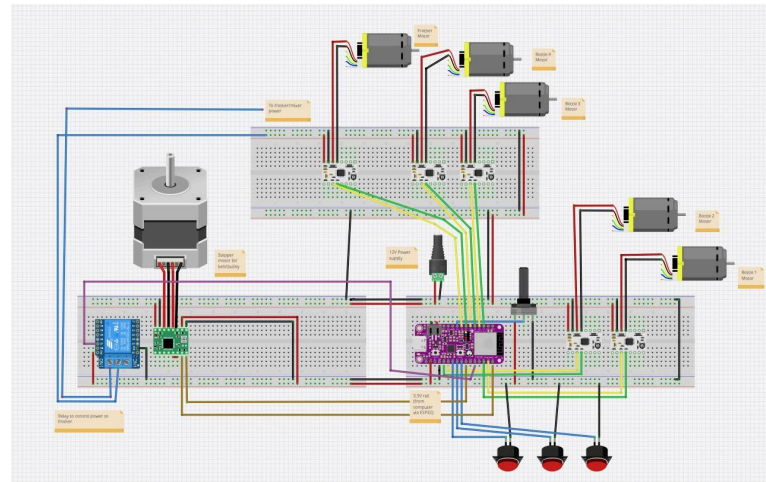
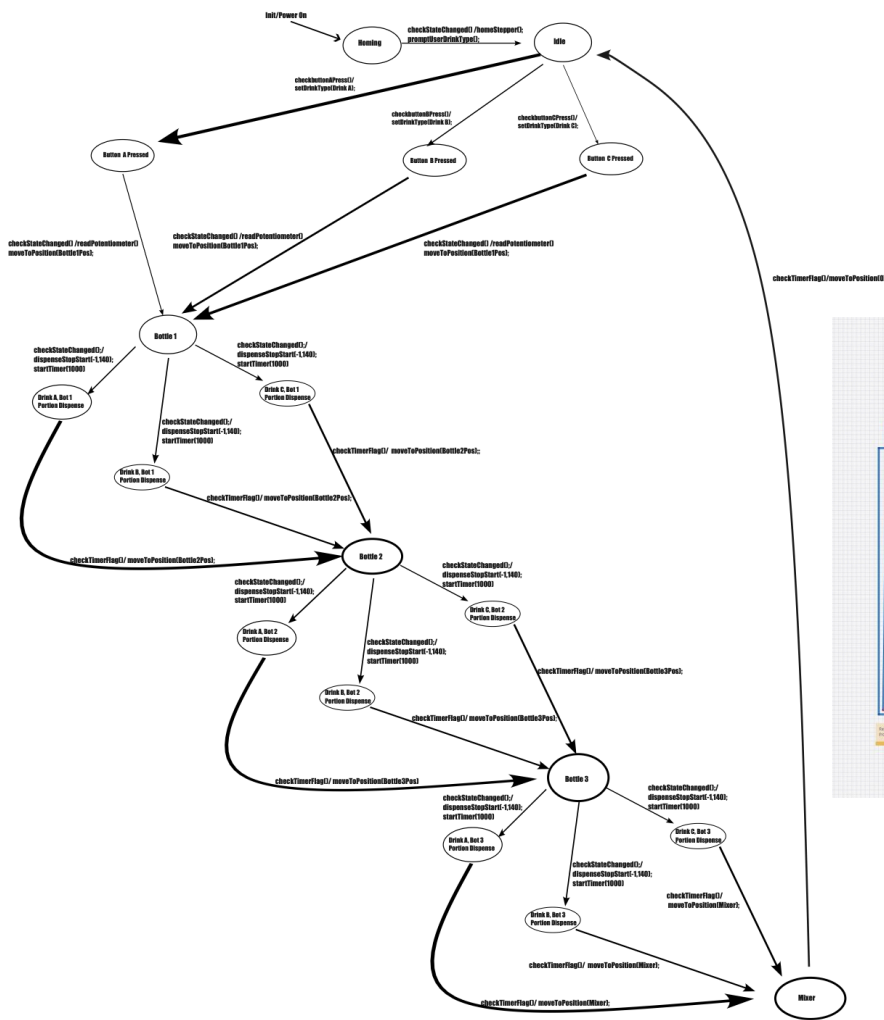
Linear motion system: The carriage assembly moves the cup from dispenser to dispenser. The carrier system is composed of two linear rods, a driving and driven pulley, shaft collars, a belt tensioning system, linear ball bearings, a cup holder, and a carriage.

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

$T = \text{Motor torque} \times \text{Gear ratio}$

$T = 0.63743 \text{ Nm} \times 6.3 = 4.015746 \text{ Nm}$

$n_{out} = \frac{490 \text{ rpm}}{6.3} = 77.778 \text{ rpm}$

Tangential force: $F_t = \frac{T_m}{\text{radius}} = \frac{0.63743 \text{ Nm}}{0.003 \text{ m}} = 212.47667 \text{ N}$

Bearing force: $F_{bearing} = 212.47667 \text{ N} \times \tan(20^\circ) = 77.3352 \text{ N}$

Load comparison

$F_{bearing} = 77.34 \text{ N} < 400.3399 \text{ N}$ (Dynamic load)

$F_{bearing} = 77.34 \text{ N} < 155.68776 \text{ N}$ (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: $77.34 \text{ N} \times 1.5 = 116.01 \text{ N}$

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.

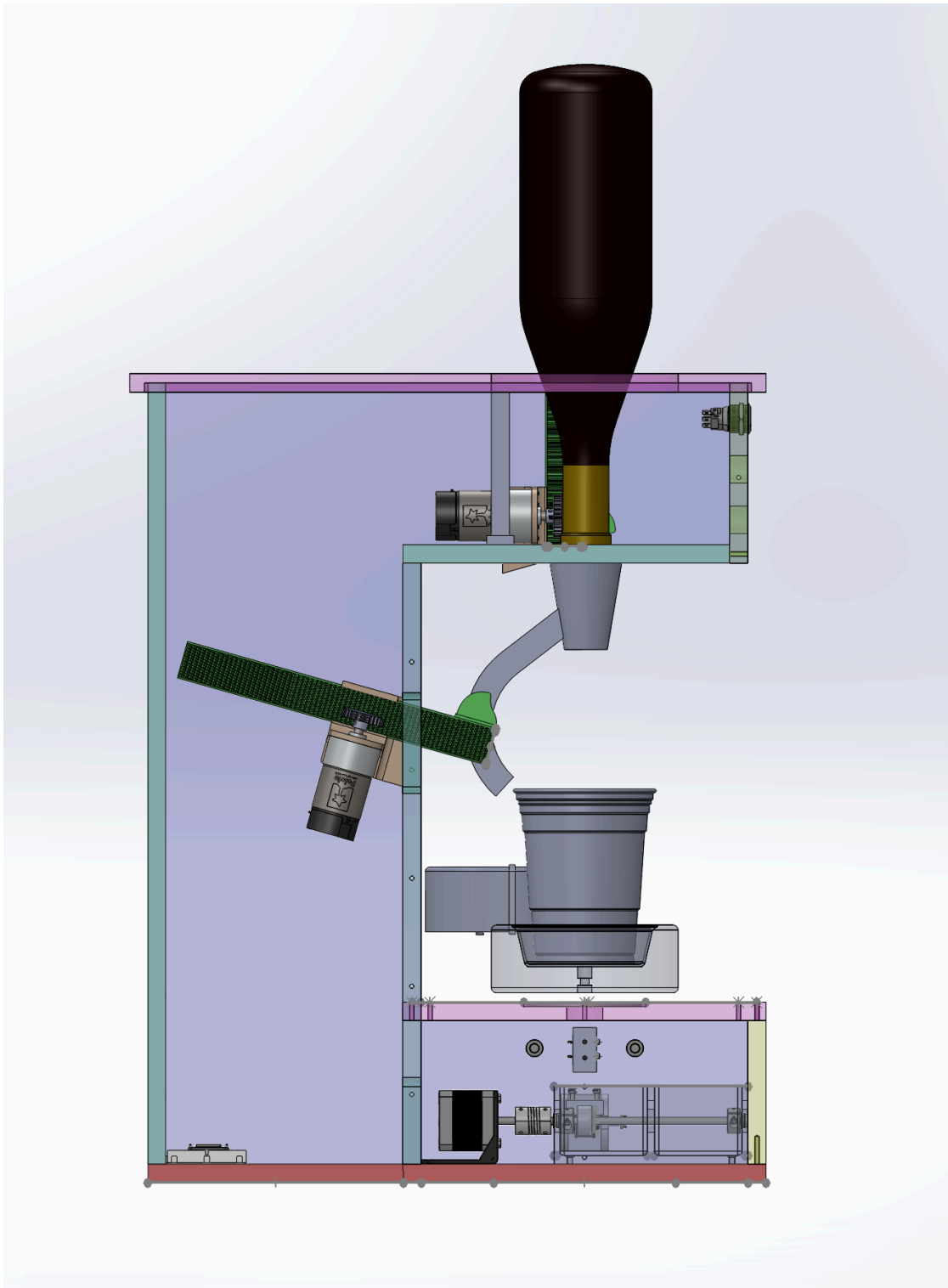
Appendix A: Bill of Material (BOM)

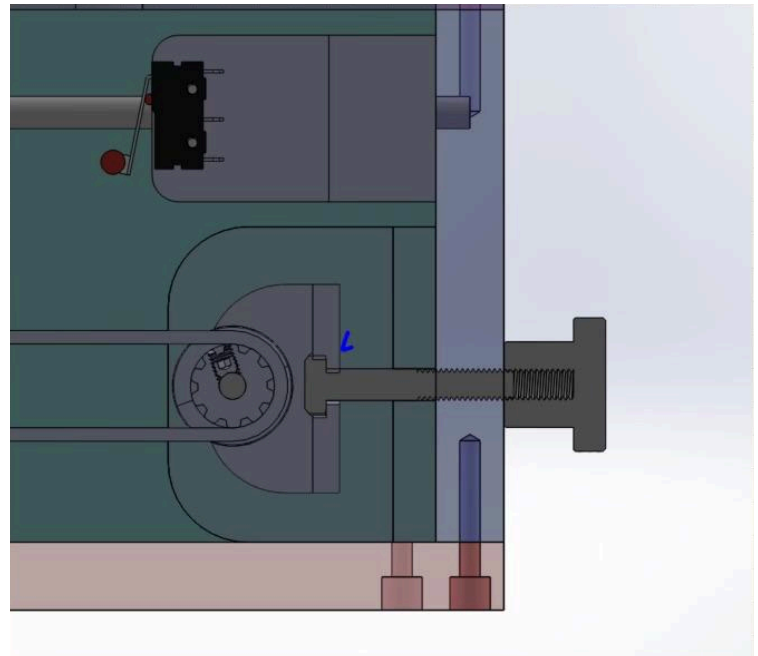
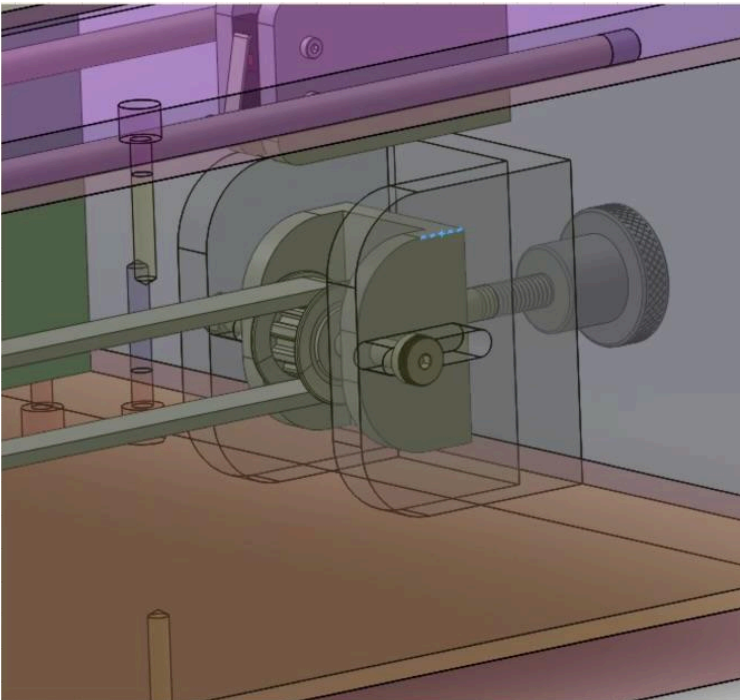
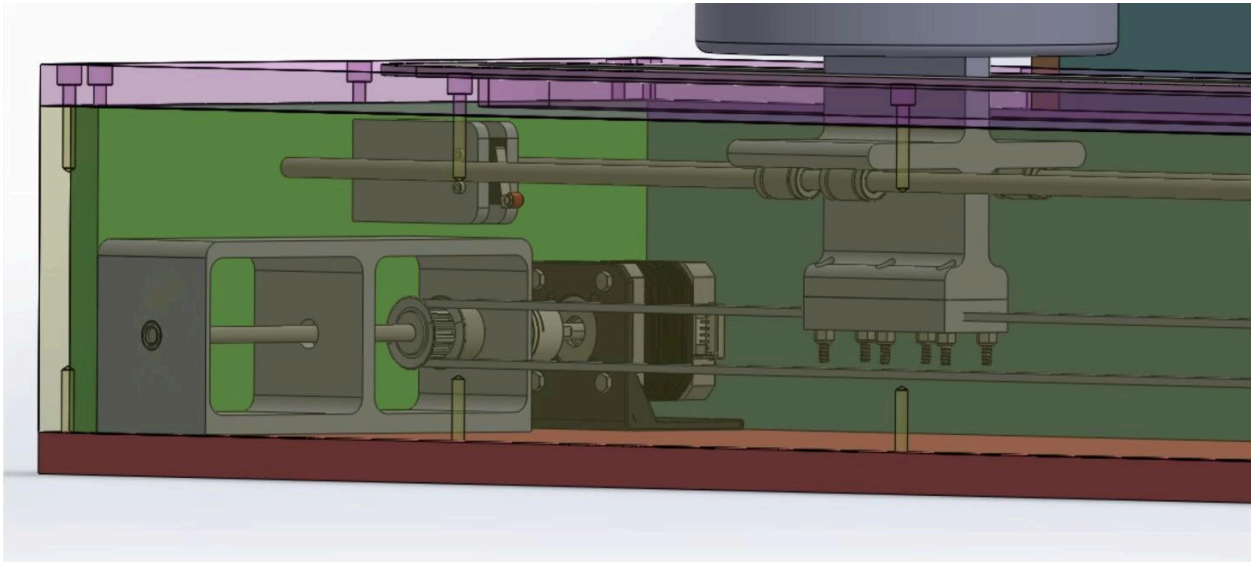
ME102B Drink Dispenser Bill of Material				
Part Number/Type	Quantity	Unit Cost	Total Cost	Source/Link
4'x8' Birch Plywood	1.5	\$87.63	\$131.45	https://store.jacobshall.org/products/1-2-plywood-shopbot
Linear Ball Bearings	1	\$8.59	\$8.59	https://www.amazon.com/dp/B0C3R38HJ7?ref=ppx_yo2ov_dt_b_fed_asin_title
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

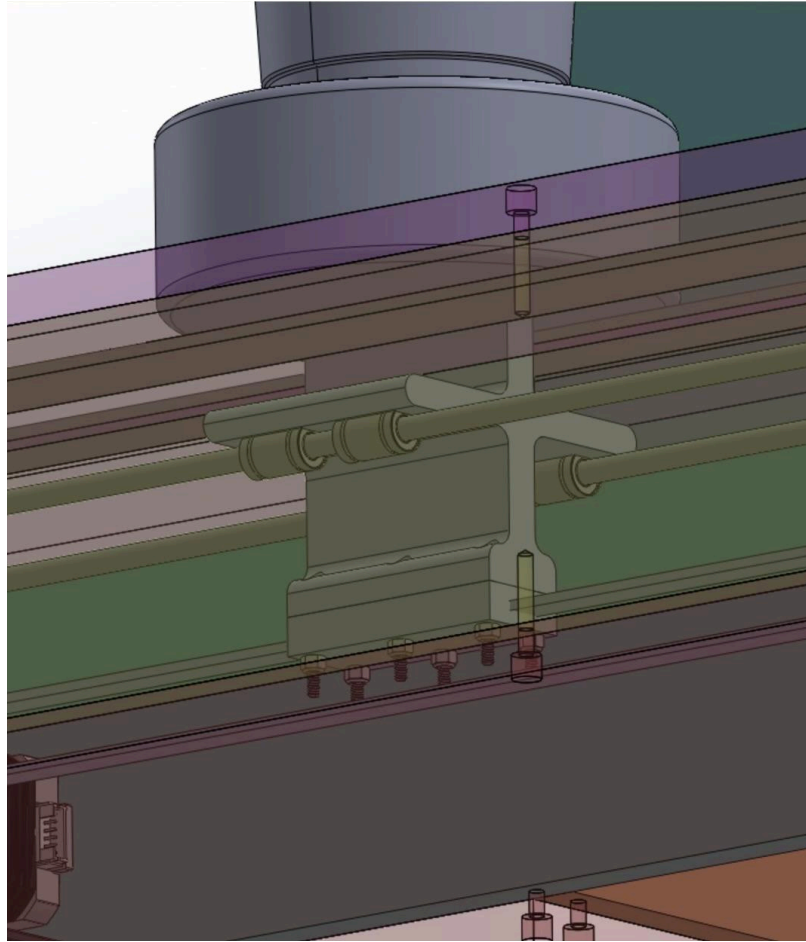
ME102B Drink Dispenser Bill of Material

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

```
showcase_WiPino
1  #include <Arduino.h>
2  #include <AccelStepper.h>
3  #include <ESP32Encoder.h>
4  #include <Wire.h>
5  #include <Adafruit_PWMServoDriver.h>
6
7  //define BTN_1 19
8  //define BTN_2 21
9  //define LED_PIN 32 //for dc motor control
10
11 //Motor 1 PWM channel 0 1
12 //Motor 2 PWM channel 2 3
13 //Motor 3 PWM channel 4 5
14 //Mixer PWM channel 6 7
15 #define LED_2 12
16 #define LED_3 13
17 #define MIXER_LED 19
18
19 ESP32Encoder encoder1;
20 ESP32Encoder encoder2;
21 ESP32Encoder encoder3;
22 ESP32Encoder encoder4;
23
24 Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver(0x49);
25
26 // Pin Definitions
27 #define BUTTON_A_PIN 4 // Drink A button
28 #define BUTTON_B_PIN // Drink B button
29 //define BUTTON_C_PIN 4 // Drink C button
30 #define POT_PIN 34 // Potentiometer pin
31 #define LIMIT_SWITCH_PIN 5 // Limit switch pin
32 //define LED_1_PIN 26 // LED for portionMultiplier = 1
33 //define LED_2_PIN 25 // LED for portionMultiplier = 2
34 //define LED_3_PIN 34 // LED for portionMultiplier = 3
35 #define DIR_PIN 32
36 #define STEP_PIN 15
37
38 // Define motor steps per revolution (e.g., 200 for typical NEMA 17)
39 #define STEPS_PER_REV 200
40 #define MICROSTEPPING 1 // Set your microstepping value (e.g., 16 for 1/16 microstep)
41
42 // Create AccelStepper object
43 AccelStepper stepper(AccelStepper::DRIVER, STEP_PIN, DIR_PIN);
44
45 int bottle1Pos = -1500;
46 int bottle2Pos = -2800;
```

```
showcase_WiPino
47 int bottle3Pos = -4000;
48 int bottle4Pos = -5100;
49
50 int frotherPos = -6000;
51
52
53
54 // Variables
55 //homing
56 bool homingCompleted = false;
57
58 //dc motor pi control
59 int theta = 0; // Current position from encoder
60 int thetaDes = 0; // Desired position (calculated from input)
61 int thetaMax = 512; // Maximum position value (adjust as needed)
62 int D = 0; // PWM duty cycle command
63 int deadZone = 5;
64 int duty = 0;
65
66 // PI Controller Gains
67 float Kp = 0.5;
68 float Ki = 0.2;
69 float errorSum = 0; // Accumulated error for integral term
70 const float IMax = 60.0; // Anti-windup limit for integral term
71
72 volatile int count1 = 0; //count for encoder, PI control
73 volatile int count2 = 0;
74 volatile int count3 = 0;
75 volatile int count4 = 0;
76 volatile int activeMotor = 0;
77
78 volatile bool deltaT = false;
79 hw_timer_t* timer1 = NULL;
80 portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;
81
82 // PWM properties
83 const int freq = 100000;
84 const int ledChannel_1 = 1;
85 const int ledChannel_2 = 2;
86 const int resolution = 8;
87 const int MAX_PWM_VOLTAGE = 255;
88
89 //debounce
90 #define DEBOUNCE_TIME 900 //debounce time in ms
91
92
```

```
showcase_WiPino
90 //debounce
91 #define DEBOUNCE_TIME 900 //debounce time in ms
92
93 int potValueChosen = 0;
94
95
96 unsigned long lastPressA = 0;
97 unsigned long lastPressB = 0;
98 unsigned long lastPressC = 0;
99
100 // State data structure
101 enum State {
102     INIT,
103     HOMING,
104     IDLE,
105     BUTTON_A_PRESSED,
106     //BUTTON_B_PRESSED,
107     //BUTTON_C_PRESSED,
108     BOTTLE_1,
109     DRINK_A_BOT_1,
110     DRINK_B_BOT_1,
111     DRINK_C_BOT_1,
112     BOTTLE_2,
113     DRINK_A_BOT_2,
114     DRINK_B_BOT_2,
115     DRINK_C_BOT_2,
116     BOTTLE_3,
117     DRINK_A_BOT_3,
118     DRINK_B_BOT_3,
119     DRINK_C_BOT_3,
120     MIXER
121 };
122 State currentState = INIT;
123
124 // Drink selection substates
125 enum DrinkState { DRINK_A,
126                  DRINK_B,
127                  DRINK_C };
128 DrinkState drinkState;
129
130 // timer variables
131 volatile bool TIMERflag = false;
132 hw_timer_t* timer0 = NULL;
133 portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
134
135 // Flags for Snsors
```

```
135 // Flags for Sensors
136 volatile bool BUTTON_A_flag = false;
137 volatile bool BUTTON_B_flag = false;
138 volatile bool BUTTON_C_flag = false;
139 volatile bool LIMIT_SWITCH_flag = false;
140 // global flag
141 bool stateChangedFlag = true; // Set to true when transitioning to a new state, used to make serial prints only print once
142
143
144 //button/switch ISR functions
145
146 void IRAM_ATTR isrButtonA() {
147     unsigned long currentTime = millis();
148     if (currentTime - lastPressA > DEBOUNCE_TIME) {
149         BUTTON_A_flag = true;
150         lastPressA = currentTime;
151         Serial.println("Button A flag set to true");
152     }
153 }
154
155 //void IRAM_ATTR isrButtonB() {
156 // unsigned long currentTime = millis();
157 // if (currentTime - lastPressB > DEBOUNCE_TIME) {
158 //     BUTTON_B_flag = true;
159 //     lastPressB = currentTime;
160 //     Serial.println("Button B flag set to true");
161 // }
162 //}
163
164 //void IRAM_ATTR isrButtonC() {
165 // unsigned long currentTime = millis();
166 // if (currentTime - lastPressC > DEBOUNCE_TIME) {
167 //     BUTTON_C_flag = true;
168 //     lastPressC = currentTime;
169 //     Serial.println("Button C flag set to true");
170 // }
171 //}
172
173 void IRAM_ATTR isrLimitSwitch() {
174     LIMIT_SWITCH_flag = true; // Set flag for limit switch
175 }
176
177 void IRAM_ATTR onTime() {
178     //Serial.println("");
179 }
```

Select Board

showcase_WIPino

```
176  
177 void IRAM_ATTR onTime() {  
178  
179     //Serial.println("");  
180     portENTER_CRITICAL_ISR(&timerMux0);  
181     TIMERflag = true;  
182     portEXIT_CRITICAL_ISR(&timerMux0);  
183     timerStop(timer0);  
184     //Serial.println("Timer Interrupt Triggered");  
185 }  
186  
187 void IRAM_ATTR onTime1() {  
188  
189     portENTER_CRITICAL_ISR(&timerMux1);  
190     switch (activeMotor) {  
191         case 1:  
192             count1 = encoder1.getCount();  
193             encoder1.clearCount();  
194             break;  
195         case 2:  
196             count2 = encoder2.getCount();  
197             encoder2.clearCount();  
198             break;  
199         case 3:  
200             count3 = encoder3.getCount();  
201             encoder3.clearCount();  
202             break;  
203         case 4:  
204             count4 = encoder4.getCount();  
205             encoder4.clearCount();  
206             break;  
207         default:  
208             // No active motor or invalid number, do nothing  
209             break;  
210     }  
211  
212     deltaT = true;  
213     portEXIT_CRITICAL_ISR(&timerMux1);  
214 }  
215  
216 // Portion Timings (in milliseconds)  
217 unsigned long drinkAPortionTime = 3000;  
218 unsigned long drinkBPortionTime = 800;  
219 unsigned long drinkCPortionTime = 1000;  
220  
221 // Output LEDpin Pin Value
```

Ln 1, Col 1

No board selected


```
showcase_WIPino

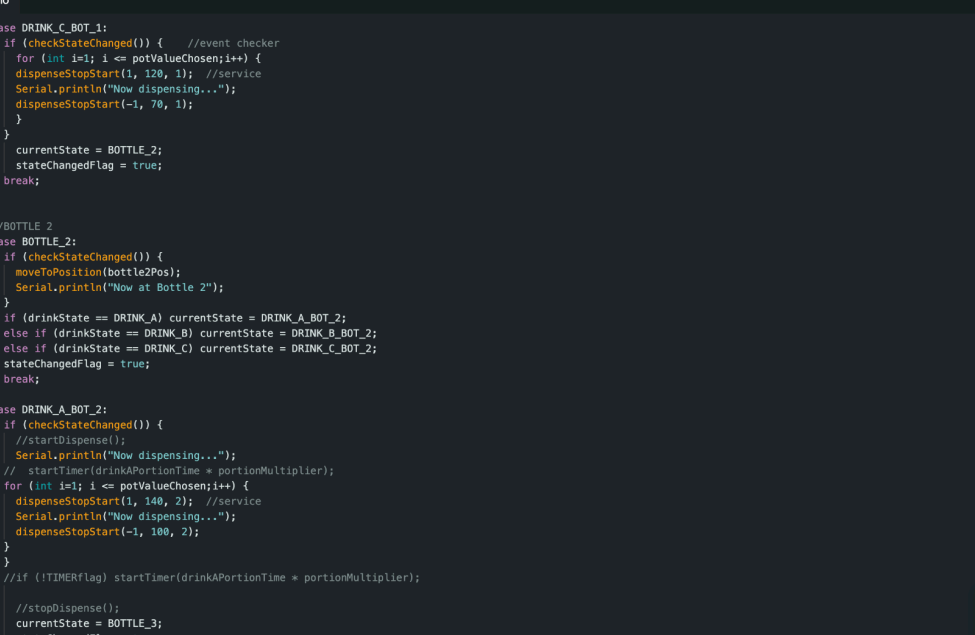
351 void loop() {
352 
353   unsigned long startupTime = millis();
354   while (millis() - startupTime < 1000) {
355     // Non-blocking delay for stabilization, peripherals
356   }
357   switch (currentState) {
358     case INIT:
359       if (checkStateChanged()) { //checkStateChanged() event checker and associated flag is used throughout to make sure print statements in loop() only get printed once per iteration
360         Serial.println("System Power On. Transitioning to Homing.");
361         //startTimer(1); //timer for stabilization, making sure system doesn't change states too quick in loop() and miss code
362       }
363       //event checker
364       currentState = HOMING;
365       stateChangedFlag = true;
366     break;
367 
368     case HOMING:
369       if (checkStateChanged()) { //event checker
370         homeStepper();
371         currentState = IDLE;
372         stateChangedFlag = true;
373       }
374       break;
375 
376     case IDLE:
377       if (checkStateChanged()) { //event checker
378         promptUserDrinkType(); //service (going from HOMING to IDLE)
379         BUTTON_A_flag = false;
380         //BUTTON_B_flag = false;
381         //BUTTON_C_flag = false;
382       }
383       if (checkbuttonAPressed()) { //event checker
384         setDrinkType(DRINK_A); //service (same logic for other 2 buttons)
385         currentState = BUTTON_A_PRESSED;
386         stateChangedFlag = true;
387       } else if (checkbuttonBPressed()) {
388         //setDrinkType(DRINK_B);
389         //currentState = BUTTON_B_PRESSED;
390         //stateChangedFlag = true;
```

```
showcase_WiPino

403
404
405 case BUTTON_A_PRESSED:
406   if (checkStateChanged()) { //event checker, going from IDLE to BUTTON pressed
407     //startPotTimer(); // service, start the 5-second timer for the user to adjust the potentiometer
408     //Serial.println("pot timer started");
409   }
410   //event checker
411   potValueChosen = readPotentiometer(); // Read the potentiometer value, service
412   currentState = BOTTLE_1;
413   stateChangedFlag = true;
414
415   break;
416
417
418 //case BUTTON_B_PRESSED:
419 //if (checkStateChanged()) { //event checker, going from IDLE to BUTTON pressed
420 //startPotTimer(); // service, start the 5-second timer for the user to adjust the potentiometer
421 //Serial.println("pot timer started");
422 //}
423 //event checker
424 //potValueChosen = readPotentiometer(); // Read the potentiometer value, service
425 //currentState = BOTTLE_1;
426 //moveToPosition(bottlePos); //service (going from BUTTON_PRESSED to BOTTLE_1)
427 //stateChangedFlag = true;
428
429 //break;
430
431
432 //case BUTTON_C_PRESSED:
433 //if (checkStateChanged()) { //event checker, going from IDLE to BUTTON pressed
434 //startPotTimer(); // service, start the 5-second timer for the user to adjust the potentiometer
435 //Serial.println("pot timer started");
436 //}
437 //event checker
438 //potValueChosen = readPotentiometer(); // Read the potentiometer value, service
439 //currentState = BOTTLE_1;
440 //moveToPosition(bottlePos); //service (going from BUTTON_PRESSED to BOTTLE_1)
441 //stateChangedFlag = true;
442
443 //break;
444
445 //BOTTLE 1
446 case BOTTLE_1:
447   if (checkStateChanged()) {
448     moveToPosition(bottlePos); //service (going from BUTTON_PRESSED to BOTTLE_1)
```



```
showcase_WiPino
445 //BOTTLE_1
446 case BOTTLE_1:
447     if (checkStateChanged()) {
448         moveToPosition(bottlePos); //service (going from BUTTON_PRESSED to BOTTLE_1)
449         Serial.println("Now at Bottle 1");
450     }
451     if (drinkState == DRINK_A) currentState = DRINK_A_BOT_1; //move to drink choice substate for this bottle, same logic for all bottle states
452     else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_1;
453     else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_1;
454     stateChangedFlag = true;
455     break;
456
457 case DRINK_A_BOT_1:
458     //same event checker/service logic here as all other DRINK_x_BOT_x cases, comments not repeated for brevity
459     if (checkStateChanged()) { //event checker
460         //dispenseStopStart(-1, 4500); //service
461         for (int i=1; i <= potValueChosen;i++) {
462             dispenseStopStart(1, 120, 1); //service
463         }
464         Serial.println("Now dispensing...");
465         //startTimer(5000); //service
466
467         //If (!TIMERflag) startTimer(drinkAPortionTime * portionMultiplier);
468         //else if (checkTimerFlag()) { //event checker
469             //dispenseStopStart(1, 200); //service
470             dispenseStopStart(-1, 70, 1);
471         }
472     }
473     currentState = BOTTLE_2;
474     stateChangedFlag = true;
475     break;
476
477 case DRINK_B_BOT_1:
478     if (checkStateChanged()) { //event checker
479         for (int i=1; i <= potValueChosen;i++) {
480             dispenseStopStart(1, 120, 1); //service
481             Serial.println("Now dispensing...");
482             dispenseStopStart(-1, 70, 1);
483         }
484     }
485     currentState = BOTTLE_2;
486     stateChangedFlag = true;
487     break;
488
489 case DRINK_C_BOT_1:
490     if (checkStateChanged()) { //event checker
```



```

showcase_WiPino
489
490
491 if (checkStateChanged()) { //event checker
492   for (int i=1; i <= potValueChosen;i++) {
493     dispenseStopStart(1, 120, 1); //service
494     Serial.println("Now dispensing...");
495     dispenseStopStart(-1, 70, 1);
496   }
497   currentState = BOTTLE_2;
498   stateChangedFlag = true;
499   break;
500
501 //BOTTLE 2
502 case BOTTLE_2:
503   if (checkStateChanged()) {
504     moveToPosition(bottle2Pos);
505     Serial.println("Now at Bottle 2");
506   }
507   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_2;
508   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_2;
509   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_2;
510   stateChangedFlag = true;
511   break;
512
513 case DRINK_A_BOT_2:
514   if (checkStateChanged()) {
515     //startDispense();
516     Serial.println("Now dispensing...");
517     // startTimer(drinkAPortionTime * portionMultiplier);
518     for (int i=1; i <= potValueChosen;i++) {
519       dispenseStopStart(1, 140, 2); //service
520       Serial.println("Now dispensing...");
521       dispenseStopStart(-1, 100, 2);
522     }
523   }
524   //If (!TIMERflag) startTimer(drinkAPortionTime * portionMultiplier);
525   //stopDispense();
526   currentState = BOTTLE_3;
527   stateChangedFlag = true;
528   break;
529
530 case DRINK_B_BOT_2:
531   //startDispense();
532   //startTimer(drinkBPortionTime * portionMultiplier);
533   for (int i=1; i <= potValueChosen;i++) {
534     dispenseStopStart(1, 140, 2); //service
535     Serial.println("Now dispensing...");
536     dispenseStopStart(-1, 100, 2);
537   }
538   //If (!TIMERflag) startTimer(drinkBPortionTime * portionMultiplier);
539   //stopDispense();
540   currentState = BOTTLE_4;
541   stateChangedFlag = true;
542   break;
543
544 case DRINK_C_BOT_2:
545   //startDispense();
546   //startTimer(drinkCPortionTime * portionMultiplier);
547   for (int i=1; i <= potValueChosen;i++) {
548     dispenseStopStart(1, 140, 2); //service
549     Serial.println("Now dispensing...");
550     dispenseStopStart(-1, 100, 2);
551   }
552   //If (!TIMERflag) startTimer(drinkCPortionTime * portionMultiplier);
553   //stopDispense();
554   currentState = BOTTLE_5;
555   stateChangedFlag = true;
556   break;
557
558 //BOTTLE 3
559 case BOTTLE_3:
560   if (checkStateChanged()) {
561     moveToPosition(bottle3Pos);
562     Serial.println("Now at Bottle 3");
563   }
564   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_3;
565   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_3;
566   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_3;
567   stateChangedFlag = true;
568   break;
569
570 //BOTTLE 4
571 case BOTTLE_4:
572   if (checkStateChanged()) {
573     moveToPosition(bottle4Pos);
574     Serial.println("Now at Bottle 4");
575   }
576   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_4;
577   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_4;
578   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_4;
579   stateChangedFlag = true;
580   break;
581
582 //BOTTLE 5
583 case BOTTLE_5:
584   if (checkStateChanged()) {
585     moveToPosition(bottle5Pos);
586     Serial.println("Now at Bottle 5");
587   }
588   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_5;
589   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_5;
590   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_5;
591   stateChangedFlag = true;
592   break;
593
594 //BOTTLE 6
595 case BOTTLE_6:
596   if (checkStateChanged()) {
597     moveToPosition(bottle6Pos);
598     Serial.println("Now at Bottle 6");
599   }
600   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_6;
601   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_6;
602   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_6;
603   stateChangedFlag = true;
604   break;
605
606 //BOTTLE 7
607 case BOTTLE_7:
608   if (checkStateChanged()) {
609     moveToPosition(bottle7Pos);
610     Serial.println("Now at Bottle 7");
611   }
612   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_7;
613   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_7;
614   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_7;
615   stateChangedFlag = true;
616   break;
617
618 //BOTTLE 8
619 case BOTTLE_8:
620   if (checkStateChanged()) {
621     moveToPosition(bottle8Pos);
622     Serial.println("Now at Bottle 8");
623   }
624   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_8;
625   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_8;
626   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_8;
627   stateChangedFlag = true;
628   break;
629
630 //BOTTLE 9
631 case BOTTLE_9:
632   if (checkStateChanged()) {
633     moveToPosition(bottle9Pos);
634     Serial.println("Now at Bottle 9");
635   }
636   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_9;
637   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_9;
638   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_9;
639   stateChangedFlag = true;
640   break;
641
642 //BOTTLE 10
643 case BOTTLE_10:
644   if (checkStateChanged()) {
645     moveToPosition(bottle10Pos);
646     Serial.println("Now at Bottle 10");
647   }
648   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_10;
649   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_10;
650   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_10;
651   stateChangedFlag = true;
652   break;
653
654 //BOTTLE 11
655 case BOTTLE_11:
656   if (checkStateChanged()) {
657     moveToPosition(bottle11Pos);
658     Serial.println("Now at Bottle 11");
659   }
660   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_11;
661   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_11;
662   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_11;
663   stateChangedFlag = true;
664   break;
665
666 //BOTTLE 12
667 case BOTTLE_12:
668   if (checkStateChanged()) {
669     moveToPosition(bottle12Pos);
670     Serial.println("Now at Bottle 12");
671   }
672   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_12;
673   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_12;
674   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_12;
675   stateChangedFlag = true;
676   break;
677
678 //BOTTLE 13
679 case BOTTLE_13:
680   if (checkStateChanged()) {
681     moveToPosition(bottle13Pos);
682     Serial.println("Now at Bottle 13");
683   }
684   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_13;
685   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_13;
686   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_13;
687   stateChangedFlag = true;
688   break;
689
690 //BOTTLE 14
691 case BOTTLE_14:
692   if (checkStateChanged()) {
693     moveToPosition(bottle14Pos);
694     Serial.println("Now at Bottle 14");
695   }
696   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_14;
697   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_14;
698   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_14;
699   stateChangedFlag = true;
700   break;
701
702 //BOTTLE 15
703 case BOTTLE_15:
704   if (checkStateChanged()) {
705     moveToPosition(bottle15Pos);
706     Serial.println("Now at Bottle 15");
707   }
708   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_15;
709   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_15;
710   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_15;
711   stateChangedFlag = true;
712   break;
713
714 //BOTTLE 16
715 case BOTTLE_16:
716   if (checkStateChanged()) {
717     moveToPosition(bottle16Pos);
718     Serial.println("Now at Bottle 16");
719   }
720   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_16;
721   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_16;
722   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_16;
723   stateChangedFlag = true;
724   break;
725
726 //BOTTLE 17
727 case BOTTLE_17:
728   if (checkStateChanged()) {
729     moveToPosition(bottle17Pos);
730     Serial.println("Now at Bottle 17");
731   }
732   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_17;
733   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_17;
734   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_17;
735   stateChangedFlag = true;
736   break;
737
738 //BOTTLE 18
739 case BOTTLE_18:
740   if (checkStateChanged()) {
741     moveToPosition(bottle18Pos);
742     Serial.println("Now at Bottle 18");
743   }
744   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_18;
745   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_18;
746   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_18;
747   stateChangedFlag = true;
748   break;
749
750 //BOTTLE 19
751 case BOTTLE_19:
752   if (checkStateChanged()) {
753     moveToPosition(bottle19Pos);
754     Serial.println("Now at Bottle 19");
755   }
756   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_19;
757   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_19;
758   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_19;
759   stateChangedFlag = true;
760   break;
761
762 //BOTTLE 20
763 case BOTTLE_20:
764   if (checkStateChanged()) {
765     moveToPosition(bottle20Pos);
766     Serial.println("Now at Bottle 20");
767   }
768   if (drinkState == DRINK_A) currentState = DRINK_A_BOT_20;
769   else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_20;
770   else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_20;
771   stateChangedFlag = true;
772   break;
773
774 //BOTTLE 21
775 case BOTTLE_21:
776   if (checkStateChanged()) {
777     moveToPosition(bottle21Pos);
778     Serial.println("Now at Bottle 21");
779   }
780   if
```

```
showcase_WIP.ino
533 case DRINK_B_BOT_2:
534   if (checkStateChanged()) {
535     Serial.println("Now dispensing...");
536     for (int i=1; i <= potValueChosen;i++) {
537       dispenseStopStart(1, 140, 2); //service
538       Serial.println("Now dispensing...");
539       dispenseStopStart(-1, 100, 2);
540     }
541     currentState = BOTTLE_3;
542     stateChangedFlag = true;
543     break;
544   }
545   case DRINK_C_BOT_2:
546     if (checkStateChanged()) {
547       Serial.println("Now dispensing...");
548       for (int i=1; i <= potValueChosen;i++) {
549         dispenseStopStart(1, 140, 2); //service
550         Serial.println("Now dispensing...");
551         dispenseStopStart(-1, 100, 2);
552       }
553       currentState = BOTTLE_3;
554       stateChangedFlag = true;
555       break;
556     }
557     //BOTTLE 3
558   case BOTTLE_3:
559     if (checkStateChanged()) {
560       moveToPosition(bottle3Pos);
561       Serial.println("Now at Bottle 3");
562     }
563     if (drinkState == DRINK_A) currentState = DRINK_A_BOT_3;
564     else if (drinkState == DRINK_B) currentState = DRINK_B_BOT_3;
565     else if (drinkState == DRINK_C) currentState = DRINK_C_BOT_3;
566     stateChangedFlag = true;
567     break;
568   case DRINK_A_BOT_3:
569     if (checkStateChanged()) {
570       Serial.println("Now dispensing...");
571     }
572   }
```

Ln 1, Col 1 X No board selected

```
showcase_WIP.ino
575 case DRINK_A_BOT_3:
576   if (checkStateChanged()) {
577     Serial.println("Now dispensing...");
578     for (int i=1; i <= potValueChosen;i++) {
579       dispenseStopStart(1, 140, 3); //service
580       Serial.println("Now dispensing...");
581       dispenseStopStart(-1,100, 3);
582     }
583     //stopDispense();
584     currentState = MIXER;
585     stateChangedFlag = true;
586     break;
587   }
588   case DRINK_B_BOT_3:
589     if (checkStateChanged()) {
590       Serial.println("Now dispensing...");
591       for (int i=1; i <= potValueChosen;i++) {
592         dispenseStopStart(1, 140, 3); //service
593         Serial.println("Now dispensing...");
594         dispenseStopStart(-1, 100, 3);
595       }
596       currentState = MIXER;
597       stateChangedFlag = true;
598       break;
599     }
600   case DRINK_C_BOT_3:
601     if (checkStateChanged()) {
602       Serial.println("Now dispensing...");
603       for (int i=1; i <= potValueChosen;i++) {
604         dispenseStopStart(1, 140, 3); //service
605         Serial.println("Now dispensing...");
606         dispenseStopStart(-1, 100, 3);
607       }
608       currentState = MIXER;
609       stateChangedFlag = true;
610       break;
611     }
612   case MIXER:
613     if (checkStateChanged()) {
614       Serial.println("Now dispensing...");
615     }
616   }
```

Ln 1, Col 1 X No board selected

```
showcase_WIP.ino
619
620 case MIXER:
621     if (checkStateChanged()) { //event checker           //service
622
623         moveToPosition(frotherPos);
624         Serial.println("Now at Mixer");
625
626         dispenseStopStart(1, 120, 4);
627         mixDrink();
628         dispenseStopStart(-1, 120, 4);
629
630         Serial.println("Resetting to Idle.");
631     }
632     moveToPosition(0);
633     currentState = IDLE;
634     stateChangedFlag = true;
635     break;
636 }
637 }
638
639 // Functions
640 void startTimer(unsigned long time) {
641     Serial.println("Starting Timer...");
642     timerWrite(timer0, 0);           // Reset the timer count
643     timerAlarmWrite(timer0, time * 1000, false); // Set alarm duration (ms to us, no auto-reload)
644     TIMERflag = false;               // Reset timer flag
645     timerAlarmEnable(timer0);
646     timerStart(timer0);
647
648     //Serial.println("Timer Started"); // Debugging print
649 }
650
651 bool checkTimerFlag() {
652     bool flagState = false; // Local variable to hold the flag state
653
654     portENTER_CRITICAL_ISR(&timerMux0); // Enter critical section
655     if (TIMERflag) {
656         TIMERflag = false; // Reset the flag
657         flagState = true;   // Mark that the flag was set
658     }
659     portEXIT_CRITICAL_ISR(&timerMux0); // Exit critical section
660
661     return flagState; // Return whether the flag was set
662 }
663
664 void stopTimer() {
```

```
showcase_WIP.ino
663
664 void stopTimer() {
665     timerStop(timer0);
666     TIMERflag = false;
667 }
668
669 // Service Functions
670
671 void promptUserDrinkType() {
672     Serial.println("Please select your drink type by pressing the corresponding button:");
673 }
674
675 void startPotTimer() {
676     Serial.println("Set your desired portion using the potentiometer. You have 5 seconds.");
677     startTimer(5000); //set 5 second timer
678
679     //timerAlarmWrite(timer0, 5000000, false); // 5 seconds
680     //TIMERflag = false;
681     //timerAlarmEnable(timer0);
682 }
683
684 int readPotentiometer() {
685     //Serial.println("in read pot");
686     int potValue = analogRead(POT_PIN);
687     portionMultiplier = map(potValue, 0, 4095, 1, 3); // Map potentiometer value
688     Serial.print("Portion Multiplier: ");
689     Serial.println(portionMultiplier);
690     //updateLEDS(portionMultiplier);
691     return portionMultiplier;
692 }
693
694 void setDrinkType(DrinkState chosenDrink) {
695     drinkState = chosenDrink;
696     Serial.print("Drink Type Selected: ");
697     if (drinkState == DRINK_A) Serial.println("A");
698     else if (drinkState == DRINK_B) Serial.println("B");
699     else if (drinkState == DRINK_C) Serial.println("C");
700 }
701
702 void moveToPosition(int position) {
703     Serial.print("Moving to position: ");
704     Serial.println(position);
705     stepper.moveTo(position);
706     stepper.runToPosition();
707 }
708 }
```

```
showcase_WiPino
710 void mixDrink() {
711   Serial.println("Mixing Drink...");
712   digitalWrite(MIXER_LED, HIGH);
713   delay(1000);
714   digitalWrite(MIXER_LED, LOW);
715 }
716
717
718
719 void updateLEDs(int multiplier) {
720   // turn all LEDs off first
721   //digitalWrite(LED_1_PIN, LOW);
722   //digitalWrite(LED_2_PIN, LOW);
723   //digitalWrite(LED_3_PIN, LOW);
724
725   //turn the corresponding LED on
726   if (multiplier == 1) {
727     digitalWrite(LED_1_PIN, HIGH);
728   } else if (multiplier == 2) {
729     digitalWrite(LED_2_PIN, HIGH);
730   } else if (multiplier == 3) {
731     digitalWrite(LED_3_PIN, HIGH);
732   }
733 }
734
735 //button/limit switch event checkers
736 bool checkbuttonAPressed() {
737   if (BUTTON_A_flag) {
738     BUTTON_A_flag = false;
739     return true;
740   }
741   return false;
742 }
743
744 //bool checkbuttonBPressed() {
745 //if (BUTTON_B_flag) {
746 //  BUTTON_B_flag = false;
747 //  return true;
748 //}
749 //return false;
750 //}
751
752 //bool checkbuttonCPressed() {
753 //if (BUTTON_C_flag) {
754 //  BUTTON_C_flag = false;
755 //  return true;
756 //}
757 //return false;
758 //}
759
760 bool checkLimitSwitchPressed() {
761   if (LIMIT_SWITCH_flag) {
762     LIMIT_SWITCH_flag = false;
763     return true;
764   }
765   return false;
766 }
767
768 bool checkStateChanged() {
769   if (stateChangedFlag) {
770     stateChangedFlag = false;
771     return true;
772   }
773   return false;
774 }
775
776 // Function to home the stepper motor using the limit switch
777 void homeStepper() {
778   Serial.println("Homming...");
779
780   // Move the motor towards the limit switch slowly
781   stepper.setMaxSpeed(1000);
782   stepper.setAcceleration(1000);
783
784   while (digitalRead(LIMIT_SWITCH_PIN) == LOW) {
785     Serial.println("Homming...");
786     Serial.println(stepper.currentPosition());
787     float target = stepper.currentPosition() + 30;
788     stepper.moveTo(target);
789     stepper.runToPosition(); // Blocking call to complete movement
790   }
791
792   // Stop when the limit switch is triggered
793   stepper.setCurrentPosition(0); // Set home position
794   homingCompleted = true;
795 }
```

```
showcase_WiPino
751
752 //bool checkbuttonCPressed() {
753 //if (BUTTON_C_flag) {
754 //  BUTTON_C_flag = false;
755 //  return true;
756 //}
757 //return false;
758 //}
759
760 bool checkLimitSwitchPressed() {
761   if (LIMIT_SWITCH_flag) {
762     LIMIT_SWITCH_flag = false;
763     return true;
764   }
765   return false;
766 }
767
768 bool checkStateChanged() {
769   if (stateChangedFlag) {
770     stateChangedFlag = false;
771     return true;
772   }
773   return false;
774 }
775
776 // Function to home the stepper motor using the limit switch
777 void homeStepper() {
778   Serial.println("Homming...");
779
780   // Move the motor towards the limit switch slowly
781   stepper.setMaxSpeed(1000);
782   stepper.setAcceleration(1000);
783
784   while (digitalRead(LIMIT_SWITCH_PIN) == LOW) {
785     Serial.println("Homming...");
786     Serial.println(stepper.currentPosition());
787     float target = stepper.currentPosition() + 30;
788     stepper.moveTo(target);
789     stepper.runToPosition(); // Blocking call to complete movement
790   }
791
792   // Stop when the limit switch is triggered
793   stepper.setCurrentPosition(0); // Set home position
794   homingCompleted = true;
795 }
```

The screenshot shows the Arduino IDE interface with a C++ sketch loaded. The sketch is titled "showcase_WiPino" and contains the following code:

```
791
792 // Stop when the limit switch is triggered
793 stepper.setCurrentPosition(0); // Set home position
794 homingCompleted = true;
795
796 Serial.println("Homing complete. Zero angle set.");
797 }
798
799 // Function to convert angle to steps
800 long angleToSteps(float angle) {
801     float stepsPerDegree = (STEPS_PER_REV * MICROSTEPPING) / 360.0;
802     return (long)(angle * stepsPerDegree);
803 }
804
805 void dispenseStopStart(int direction, int angle, int bottleNumber) {
806
807     activeMotor = bottleNumber;
808     int angleCounts = angle;
809     thetaDes = theta + direction * angleCounts;
810
811     int PwM_A = 0;
812     int PwM_B = 0;
813
814     if (bottleNumber == 1) {
815         PwM_A = 0;
816         PwM_B = 1;
817     } else if (bottleNumber == 2) {
818         PwM_A = 7;
819         PwM_B = 3;
820     }
821
822     else if (bottleNumber == 3) {
823         PwM_A = 4;
824         PwM_B = 5;
825     }
826
827     else if (bottleNumber == 4) {
828         PwM_A = 6;
829         PwM_B = 7;
830     }
831
832     startTimer(5000);
833     while (!checkTimerFlag()) {
834
835         //dc motor command/ pi control
836         // direction = -1 for forwards motor drive
837         // angle = # of rotations,
838         Serial.println("forward distance: " + angle);
```

The IDE interface includes a "Select Board" dropdown menu at the top, a toolbar on the left with icons for file operations, and a status bar at the bottom indicating "Ln 1, Col 1" and "No board selected". A small inset window in the bottom right corner shows a preview of the code.


```
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   // Accumulate the error for the integral term
862   errorSum += error;
863
864   // Anti-Windup: Clamp the integral sum to prevent wind-up
865   if (errorSum > IMax) {
866     errorSum = IMax;
867   } else if (errorSum < -IMax) {
868     errorSum = -IMax;
869   }
870
871   // PI control equation: D = Kp * error + Ki * errorSum
872   D = Kp * error + Ki * errorSum;
873
874   // Clamp the PWM command to the allowed range
875   if (D > MAX_PWM_VOLTAGE) {
876     D = MAX_PWM_VOLTAGE;
877   } else if (D < -MAX_PWM_VOLTAGE) {
878     D = -MAX_PWM_VOLTAGE;
879   }
880
881
```

```
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
912   pwm.setPWM(PWM_A, 0, 0);
913   pwm.setPWM(PWM_B, 0, 0);
914 }
```