Perfect Beer Pour Bot

Nana Porter-Honicky, Jun Yeob Lee, Alexandr Nechay, Hayden Ye

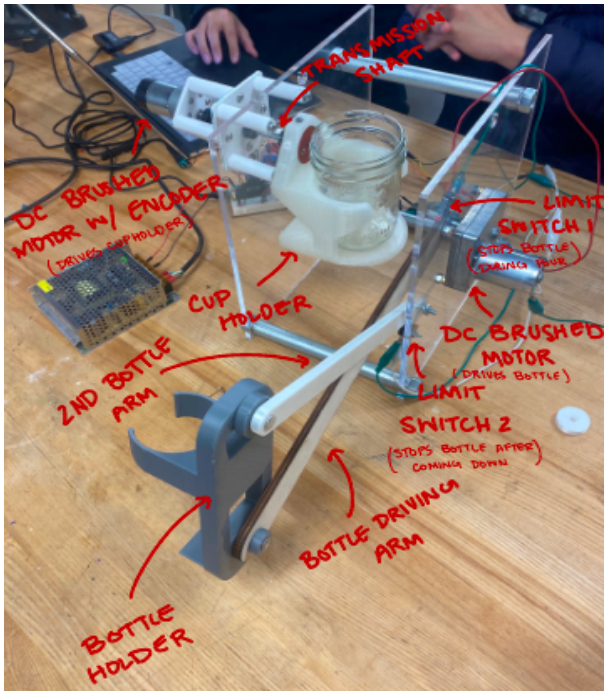
Opportunity

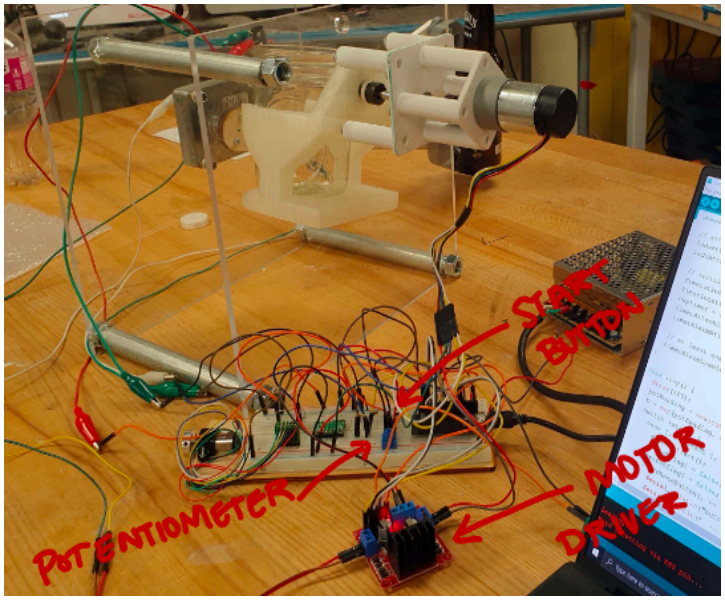
It is often hard for people to pour the perfect glass of beer. It requires an amount of dexterity and timing that many people do not have the patience for and even when someone does achieve the perfect pour it can be hard to replicate. With our Perfect Beer Pour machine, we aim to make this task easy and consistent.

High Level Strategy

The main functionality that we wanted to achieve with this machine was replicating the path a bartender would go through when pouring a glass of beer so we could have the perfect beer pour. Initially, we wanted to use a weighted sensor to tell the machine when to stop pouring the beer in addition to changing the angle of the glass to match the perfect pour angle. While our realized product does change the angle of the glass as well as imitates the human motion of pouring a beer, it does not use a weighted sensor, and instead uses limit switches to tell the machine when to stop moving the bottle. We also initially thought of using a linear actuator to pull the bottle up its path but switched to a linkage mechanism for simplicity. With our linkage mechanism driving the path of the bottle and our smaller motor doing a PI control of the cup, we were able to achieve our perfect beer pour.

Photos of Physical Device



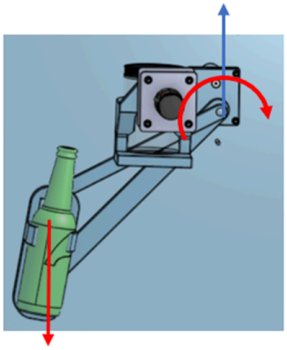


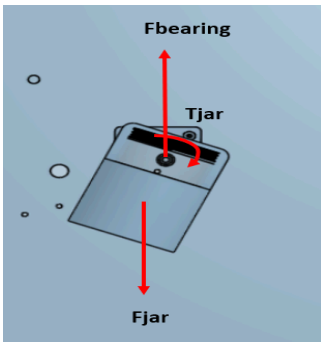
Function Critical Decisions

For the motor selections, the worst-case scenario is estimated, where the filled beer bottle is 556g. The length of the moment arm is measured to be 34.2cm. The calculated maximum torque is 0.556*34.2 = 19.0152 (kg*cm). Considering the friction which cannot be measured in the design phase, with the linkage power loss, fixture weight, and the extra load caused by the misalignment of the shaft, a conservative estimation is 50% higher. Therefore,

the conservative maximum torque required would be 28.52 kg*cm. The unit kg*cm is used instead of Nm because the motor datasheets usually use kg*cm and lbs*in. Based on this measurement, the compact square-face DC gear motor is selected (24V, 47rpm, 28 in-lbs or 32.26 kg*cm). This motor is sufficient to power the fully loaded linkage, in sacrifice of the rotational speed. By estimating the motion speed in the manual perfect pouring video, an ideal speed would be around 120 deg/sec, around 20 rpm, which is half of its no-load speed. Therefore, this motor (McMaster ID: 6409K27) has an ideal balance between torque output and flexibility in rotating speed.

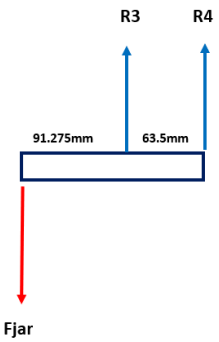
Also, the square compact motor has its own metal housing and bearing for the shaft, with an overhung load of 13lb, which is nearly 10 times larger than the possible radial load in operation.

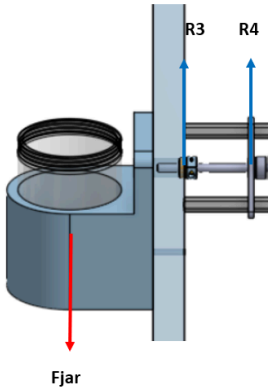




For the jar-motor system, we calculated the maximum torque on the bearing of the system using the average weight of the jar, Fjar = 0.443*9.8=4.341 N. The average weight center of the jar was estimated at a 2cm horizontal distance away from the bearing. Using a conservative estimation, the maximum torque was calculated as Tjar = 1.5*4.341*2 = 13.023 N-cm = 1.327 kg-cm. The torque is less than half of that of the bottle-motor system, allowing us to use a motor within a reasonable price range.

Based on the torque estimation, we selected Pololu #4757, with 3.0kg*cm stall torque and 1600 rpm no-load speed. In this case, a motor housing needs to be designed to protect the motor shaft. After inputting the housing dimensions, a free body diagram of the housing shaft is drawn as follows.





To calculate the bearing forces R3 and R4, the free body diagram is drawn as shown above. The force and moment balance equations are used to solve for the reaction forces. Summing moment of the system:

$$4.341 \times 154.775 - R3 \times 63.5 = 0$$

$$R3 = 10.58N$$

$$R4 = 4.341 - 10.58 = -6.24N$$

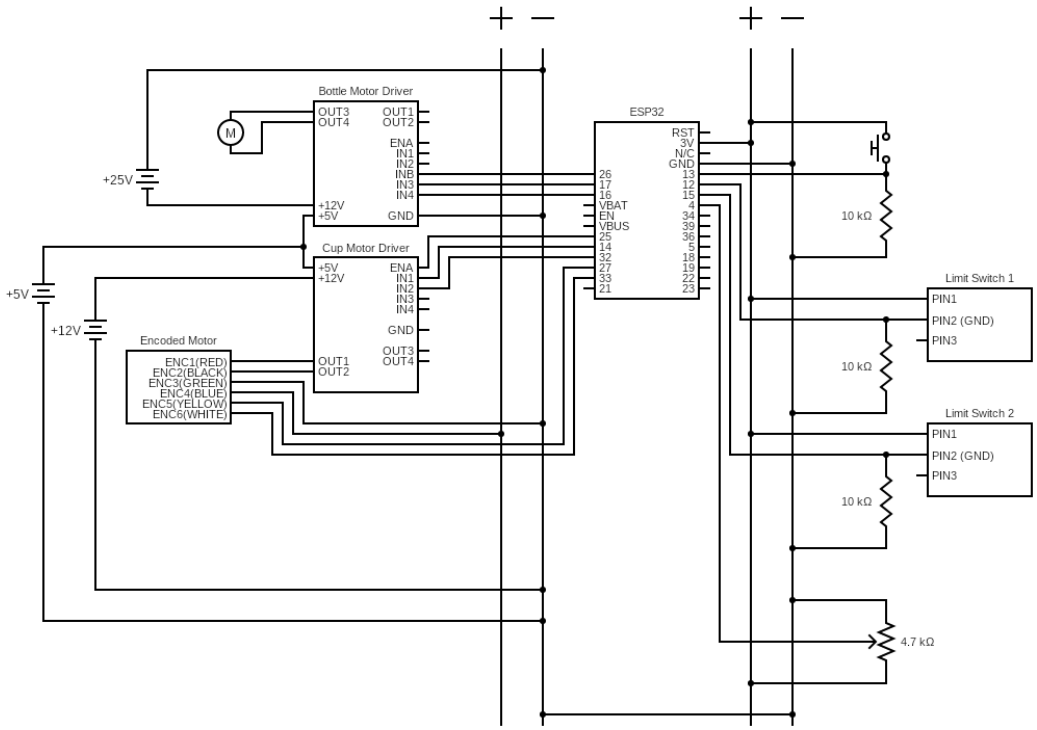
Both of these radial loads are far smaller than the load tolerance of bearing (57155k305), having 330lbs dynamic load capacity and 120lbs static load capacity.

Reflection

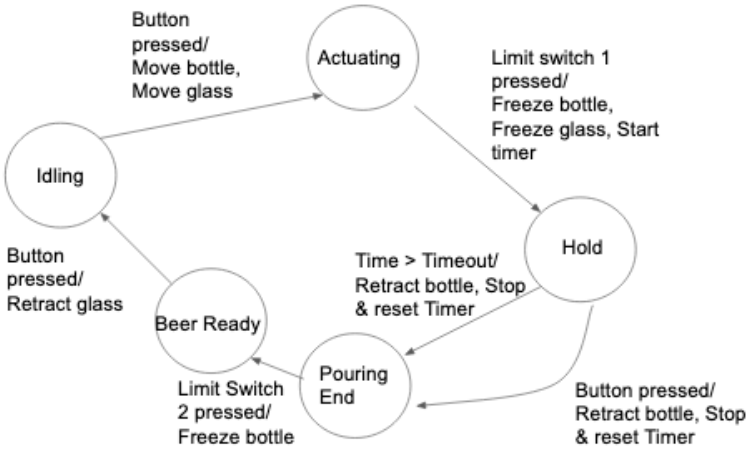
We should have considered fabricating more prototypes instead of assuming that our final fabricated part will be perfect and be exactly what we wanted. Doing more prototypes

would have allowed us to see the flaws in our design and would have saved us time that we instead spent trying to reconfigure or fix parts that we ended up replacing anyway. Additionally, we should have anticipated that when combining the code and the actual physical machine, there could be some flaws and errors. We should have allowed more time for us to debug and fix that instead of figuring out that there were problems at the last minute. We also should have used a motor with an encoder to drive the bottle linkage for a more precise motion. Overall, the project opened our eyes to how all these different aspects of engineering that we have learned about in separate classes come together to create a single physical thing. It was also interesting to learn how many aspects come into building something on a tight budget.

Circuit Diagram



State Transition Diagram



Appendix A: Bill of Materials

Name	Quantity	Cost per unit	Part No.	Link
Flanged Ball Bearing	2	6.42	57155K305	https://www.mcmaster.com/flanged-ball-bearings/flanged-ball-bearings-7/
Belleville Disc Spring for Ball Bearings	2	3.76 for 10	94065K26	https://www.mcmaster.com/catalog-128%2f1441/
316 Stainless Steel Round Shim	2	8.63 for 10	97022A372	https://www.mcmaster.com/catalog-128%2f3568/
One-Piece Clamp-On Shaft Collar	2	3.43	6435K12	https://www.mcmaster.com/catalog-128%2f1359/
Miniature Drive Shaft (¼', 4' long)	1	4.49	1327K114	https://www.mcmaster.com/catalog-128%2f1263/
Helical Flexible Shaft Coupling	1	59.97	2464K2	https://www.mcmaster.com/catalog-128%2f1374/
Set Screw Shaft Coupling	1	61.34	9861T722	9861T722 https://www.mcmaster.com/9861t722/
Metal Gearmotor 37Dx65L mm 12V with 64 CPR Encoder (Helical Pinion	1	51.95	4757	https://www.pololu.com/product/4757

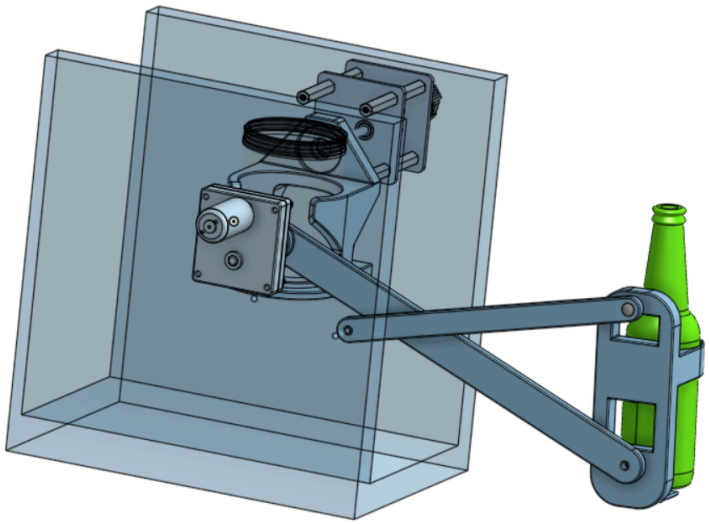
Compact Square-Face DC Gearmotor	1	62.74	6409K27	6409K27 https://www.mcmaster.com/6409K27/
Flanged screw (M3 6mm)	6	5.04 for 100	97654A674	https://www.mcmaster.com/catalog-128%2f3327/
Laser acrylic panels	2	22.00	N/A	Machined
Formlabs - Engineering - Tough 2000 Resin - Form3 Print Material (priced per ml)	About 1000ml	.21 per ml	N/A	https://store.jacobshall.org/products/formlabs-engineering-tough-2000-resin-form3-print-material-priced-per-ml
Washer (M5)	8	4.00 for 100	98689A114	https://www.mcmaster.com/catalog-128%2f3544/
Screw (M5*0.8 14mm)	8	10.01 for 25	92095a211	https://www.mcmaster.com/92095a211/
Hex nut (M5*0.8)	8	4.05 for 100	90592A095	https://www.mcmaster.com/90592A095/
Spacer: Male-female standoff (51mm, M5*0.8)	16	6.00	98952A430	https://www.mcmaster.com/catalog-128%2f3595/
6063 Aluminum Low-Profile Binding Barrels and Screws,	1	17.03 for 50	93121A337	https://www.mcmaster.com/93121A337/

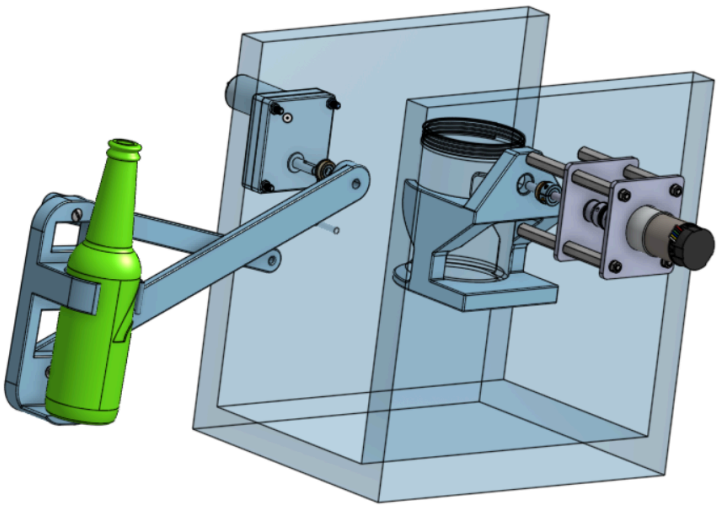
8-32 Thread Size, for 5/8"-7/8" Material Thickness				
063 Aluminum Low-Profile Binding Barrels and Screws, 8-32 Thread Size, for 1-1/4"-1-1/2" Material Thickness	3	12.51 for 25	93121A350	https://www.mcmaster.com/93121A350/
Limit Switches	2	5.99 for 10	N/A	https://www.amazon.com/HiLetgo-KW12-3-Roller-Switch-Normally/dp/B07X142VGC/ref=sr_1_1_sspa?crid=2OCDBJNFOX10I&keywords=limit+switch&qid=1668031870&sprefix=limit+switc%2Caps%2C225&sr=8-1-spons&psc=1
20 cm Steel ½-13 rod	2	8.14	99065A402	https://www.mcmaster.com/99065A402
17cm steel ¾” pipe	2	10.86	7750K112	https://www.mcmaster.com/7750K112
Heat sets				
½-13 hex nuts	6	12.60 for 10	94575A470	https://www.mcmaster.com/94575A470

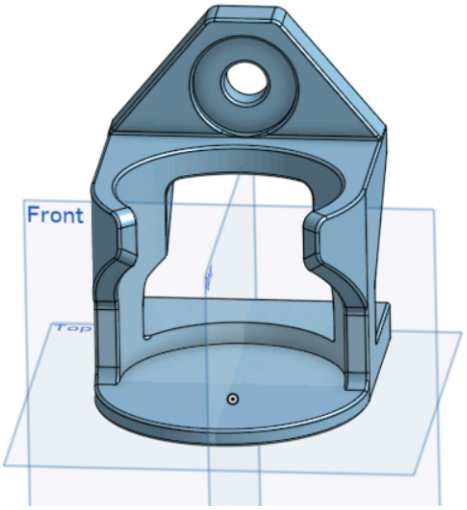
PLA cup holder	1	6.12	N/A	3D Printed
PLA bottle holder	1	3.20	N/A	3D Printed
PLA secondary linkage	1	5.59	N/A	3D Printed
PLA DC motor housing	1	7.34	N/A	3D Printed
8mm Flange couplers connector	2	9.99 for 2	N/A	https://www.amazon.com/dp/B0822425TT/ref=cm_sw_r_api_i_9MNQYJ44M8D3WWPG6K7P_0?_encoding=UTF8&psc=1

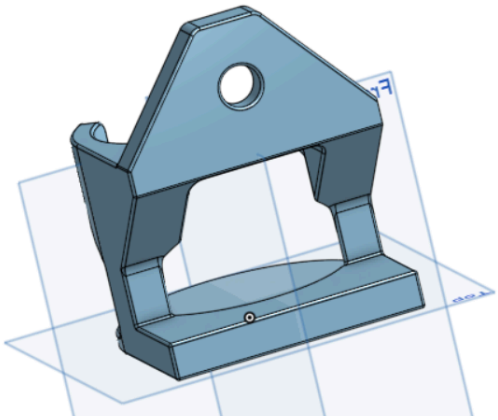
Total: $419.40

Appendix B: CAD






Front
Top



Appendix C: Code

```
#include <ESP32Encoder.h>
#include <Arduino.h>
#define BTN 13 //Button to be manually pressed by user
#define BIN_1 26 //Motor encoder pin  motor B (bottle) enable pin, direction controlled by dir3 dir4
#define BIN_2 25 //Motor encoder pin  motor A (cup) enable pin, dirrection controlled by dir1 dir2
#define SWITCH1 12 // Limit switch at final bottle position
#define SWITCH2 15 // Limit switch at initial bottle position
#define dir1 14 //motor driver input 1, motor A direction control
#define dir2 32 //motor driver input 2, motor A direction control
#define dir3 17
#define dir4 16 //motor driver input 4, motor B direction control
#define POT 4 //analog input: potentiometer to control motor B (bottle) velocity

ESP32Encoder encoder;

//Setup variables ------------------------------------
int state = 1; //State of the bottle-glass system
int sumerror = 0;
int theta = 0;
int thetaDes = 0; //the 10 times of desired theta
int thetaMax = 455; // 75.8 * 6 counts per revolution
int D_cup = 0; // PWM of the cup motor
int D = 0; //PWM for bottle motor
int potReading = 0;


int Kp = 50;   // TUNE THESE VALUES TO CHANGE CONTROLLER PERFORMANCE
float Ki = 2;
int KiMax = 50;

//Setup interrupt variables ----------------------------
volatile bool buttonflag = false;
volatile bool switchflag1 = false;
volatile bool switchflag2 = false;
volatile bool timeoutflag = false;
volatile bool interruptCounter = false;
hw_timer_t * TimeoutTimer = NULL;
portMUX_TYPE timeoutMux = portMUX_INITIALIZER_UNLOCKED;

volatile int count = 0; // encoder count
volatile bool cupflag = false;     // check timer interrupt
hw_timer_t * cuptimer = NULL;
portMUX_TYPE cuptimerMux = portMUX_INITIALIZER_UNLOCKED;

hw_timer_t * debounceTimer = NULL;
portMUX_TYPE debounceMux = portMUX_INITIALIZER_UNLOCKED;

// setting PWM properties ---------------------------
const int freq = 5000;
const int ledChannel_1 = 1; //Assumed that we are using a second esp32 board. Change the value if using one board
const int ledChannel_2 = 2; //Assumed that we are using a second esp32 board. Change the value if using one board
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;
```

```
//Initialization -------------------------------------
void IRAM_ATTR isr() {
  buttonflag = true;
}

void IRAM_ATTR isr2() {
  switchflag1 = true;
  interruptCounter = true;
  timerStart(debounceTimer);
}

void IRAM_ATTR isr3() {
  switchflag2 = true;
  interruptCounter = true;
  timerStart(debounceTimer);
}
void IRAM_ATTR isr4() {  //for debounce
  buttonflag = true;
  interruptCounter = true;
  timerStart(debounceTimer);
}


void IRAM_ATTR onTimeout() { // the function to be called when Timeout interrupt is triggered
  //portENTER_CRITICAL_ISR(&timeoutMux);
  timeoutflag = true;
  timerStop(TimeoutTimer);
  timerRestart(TimeoutTimer);
  //portEXIT_CRITICAL_ISR(&timeoutMux);
}
void TimeoutInterruptInit() {
  TimeoutTimer = timerBegin(0, 80, true);
  timerAttachInterrupt(TimeoutTimer, &onTimeout, true);
  timerAlarmWrite(TimeoutTimer, 5000000, true);
  timerAlarmEnable(TimeoutTimer);
  timerStop(TimeoutTimer); //Timer is stopped initially, until it's called to start
}

void IRAM_ATTR onCupTime() {
  portENTER_CRITICAL_ISR(&cuptimerMux);
  count = encoder.getCount( );
  encoder.clearCount ( );
  cupflag = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&cuptimerMux);
}
```

```
void IRAM_ATTR onDebounceTime() {
  portENTER_CRITICAL_ISR(&debounceMux);
  timerStop(debounceTimer);
  interruptCounter = false; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&debounceMux);
}
void TimerInterruptInit() {  //The timer simply counts the number of Tic generated by the quartz. With a quartz clocked at 80MHz, we will have 80
  debounceTimer = timerBegin(2, 80, true); // divides the frequency by the prescaler: 80,000,000 / 80 = 1,000,000 tics / sec
  timerAttachInterrupt(debounceTimer, &onDebounceTime, true);    // sets which function do you want to call when the interrupt is triggered
  timerAlarmWrite(debounceTimer, 400000, true);         // sets how many tics will you count to trigger the interrupt 50000
  timerAlarmEnable(debounceTimer); // Enables timer
}

void setup() {
  // put your setup code here, to run once:
  pinMode(BTN, INPUT);  // configures the specified pin to behave either as an input or an output
  attachInterrupt(BTN, isr4, RISING);
  pinMode(SWITCH1, INPUT);
  attachInterrupt(SWITCH1, isr2, RISING);
  pinMode(SWITCH2, INPUT);
  attachInterrupt(SWITCH2, isr3, RISING);
  pinMode(dir1, OUTPUT);
  pinMode(dir2, OUTPUT); // digital output, control motor A's direction, cup
  pinMode(dir3, OUTPUT);
  pinMode(dir4, OUTPUT); // digital output, control motor B's direction, bottle
  pinMode(POT, INPUT); //potentiometer analog input, control motor B's speed, bottle
  Serial.begin(115200);

  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_1, freq, resolution);
  ledcSetup(ledChannel_2, freq, resolution);

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

  // initilize timer
  TimeoutInterruptInit(); // Initiates timout timer interrupt
  TimerInterruptInit();
  cuptimer = 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, cou
  timerAttachInterrupt(cuptimer, &onCupTime, true); // edge (not level) triggered
  timerAlarmWrite(cuptimer, 10000, true); // 10000 * 1 us = 10 ms, autoreload true

  // at least enable the timer alarms
  timerAlarmEnable(cuptimer); // enable
}

void loop() {
  delay(100);
  potReading = analogRead(POT);
  D = map(potReading, 0, 4095, 0, 255);//absolute speed for bottle motor
  switch (state) {
    case 1: // STATE 1: IDLING
      glassretract();
      switchflag1 = false;
      switchflag2 = false;
      if (CheckButton() == true) { // If button is pressed, drive the bottle motor forward
        Serial.println("button triggered, state 1 to 2, current state 2: actuating");
        Serial.println("  -- service: glassretract(), bottlemove()");
        bottlemove(); //Turn on the motor until bottlefreeze() is called
        state = 2;
      }
      break;
    case 2: //STATE 2: ACTUATING
      glassmove();
      buttonflag = false;
      switchflag2 = false;
      //glassmove(); //Turn on the motor until bottlefreeze() is called
      if (CheckSwitch1() == true) { //If limit switch is pressed by the bottle, then freeze the bottle motion and start Timeout Timer
        Serial.println("switch triggered, state 2 to 3, current state 3: hold");
        Serial.println("  -- service: glassfreeze(), bottlefreeze(), timerStart()");
        //glassfreeze();
        bottlefreeze();
        timerStart(TimeoutTimer);
        state = 3;
      }
      break;
```

```
    case 3: //STATE 3: HOLD
      switchflag1 = false;
      switchflag2 = false;
      if (CheckTimeout() == true) { //If Timer reaches Timeout, then retract the bottle and stop Timeout Timer
        Serial.println("state 3 to 4, current state 4 pouring end: triggered by timeout");
        Serial.println("  -- service: bottleretract(), timerStop(), timerRestart()/i.e. reset");
        bottleretract();
        //timerRestart and timer stop included in isr
        state = 4;
      }
      else if (CheckButton() == true) { //If Button is pressed manually by the user, then retract the bottle and stop Timeout Timer
        Serial.println("state 3 to 4, current state 4 pouring end: triggered by button manually");
        Serial.println("  -- service: bottleretract(), timerStop(), timerRestart()/i.e. reset");
        bottleretract();
        timerRestart(TimeoutTimer);
        timerStop(TimeoutTimer);
        state = 4;
      }
      break;
    case 4: //STATE 4: POURING END
      glassfreeze();
      switchflag1 = false; //In case the switch1 is pressed in state 4 due to misoperation, turn off flag manually
      buttonflag = false;
      if (CheckSwitch2() == true) { //If the Limit switch is pressed by the bottle after retracting, then freeze the bottle motion
        Serial.println("state 4 to 5, current state 5 beer ready");
        Serial.println("  -- service: bottlefreeze()");
        bottlefreeze();
        state = 5;
      }
      break;
    case 5: //STATE 5: BEER READY
      switchflag1 = false;
      switchflag2 = false;
      if (CheckButton() == true) { //If Button is pressed by the user after the beer is taken out, then reset the bottle to initial position
        Serial.println("state 5 to 1, current state 1 idling");
        Serial.println("/--------------One cycle ended---------------/");
        Serial.println("Please place new beer bottle and empty cup");
        state = 1;
      }
      break;
  }
}

bool CheckButton() {  //Event Checker 1: Check if button is pressed by checking the button flag
  if (buttonflag == true && interruptCounter==false){
    buttonflag = false;
    return true;
  }
  else {
    return false;
  }
}

bool CheckSwitch1() {  //Event Checker 2: Check if the limit switch at final bottle position is pressed.
  if (switchflag1 == true && interruptCounter==false){
    switchflag1 = false;
    return true;
  }
  else {
    return false;
  }
}
```

```
bool CheckSwitch2() {  //Event Checker 3: Check if the limit switch at the initial bottle position is pressed.
  if (switchflag2 == true && interruptCounter==false){
    switchflag2 = false;
    return true;
  }
  else {
    return false;
  }
}

bool CheckTimeout() {  //Event Checker 3: Check if time has passed until Timeout by checking the timeout flag
  if (timeoutflag == true) {
    portENTER_CRITICAL_ISR(&timeoutMux);
    timeoutflag = false; // the function to be called when timer interrupt is triggered
    portEXIT_CRITICAL_ISR(&timeoutMux);
    return true;
  }
  else {
    return false;
  }
}

void glassmove() {
    if (cupflag) {
      portENTER_CRITICAL(&cuptimerMux);
      cupflag = false;
      portEXIT_CRITICAL(&cuptimerMux);

      theta += count;
      if (thetaDes > -40){
        Serial.print("Desired position of cup:");
        Serial.println(thetaDes/10);
        thetaDes -= 8; // cup's orientation varies from 45 degrees to 90 degrees, with respect to the horizontal (thetaDes = thetaMax/8 = 455/8)
      }
      int thetaD = thetaDes/10;
      sumerror += thetaD - theta;
      if (sumerror > KiMax){
        sumerror = KiMax;
      }else if (sumerror<-KiMax)
      {
        sumerror = -KiMax;
      }
      D_cup = Kp*(thetaD-theta)+Ki*sumerror; //PI Control of the cup
      //Ensure that you don't go past the maximum possible command
      if (D_cup > MAX_PWM_VOLTAGE) {
          D_cup = MAX_PWM_VOLTAGE;
      }
      else if (D_cup < -MAX_PWM_VOLTAGE) {
          D_cup = -MAX_PWM_VOLTAGE;
      }

      //speed magnitude control
      ledcWrite(ledChannel_2, abs(D_cup));

      //Map the motor directionality
      //control dir1 and dir2
      if (D_cup > 2) {
        digitalWrite(dir1, LOW);
        digitalWrite(dir2, HIGH);
      }
      else if (D_cup < -2) {
        digitalWrite(dir1, HIGH);
        digitalWrite(dir2, LOW);
      }
      else {
        digitalWrite(dir1, LOW);
        digitalWrite(dir2, LOW);
      }

  }
}
```

```
void glassretract() {
    if (cupflag) {
      portENTER_CRITICAL(&cuptimerMux);
      cupflag = false;
      portEXIT_CRITICAL(&cuptimerMux);

      theta += count; //Theta decreases from 90 degrees to 45 degrees
      thetaDes = 80; // cup's orientation varies from 90 degrees to 45 degrees, with respect to the horizontal (thetaDes = -thetaMax/8 = -455/8)
      int thetaD = thetaDes/10;
      sumerror += thetaD - theta;
      if (sumerror > KiMax){
        sumerror = KiMax;
      }else if (sumerror<-KiMax)
      {
        sumerror = -KiMax;
      }
      D_cup = Kp*(thetaD-theta)+Ki*sumerror; //PI Control of the cup

       //Ensure that you don't go past the maximum possible command
       if (D_cup > MAX_PWM_VOLTAGE) {
           D_cup = MAX_PWM_VOLTAGE;
       }
       else if (D_cup < -MAX_PWM_VOLTAGE) {
           D_cup = -MAX_PWM_VOLTAGE;
       }

       //speed magnitude control
       ledcWrite(ledChannel_2, abs(D_cup));

       //Map the motor directionality
       //control dir1 and dir2
       if (D_cup > 2) {
         digitalWrite(dir1, LOW);
         digitalWrite(dir2, HIGH);
       }
       else if (D_cup < -2) {
         digitalWrite(dir1, HIGH);
         digitalWrite(dir2, LOW);
       }
       else {
         digitalWrite(dir1, LOW);
         digitalWrite(dir2, LOW);
       }

  }
}
```

```
void glassfreeze() {
  D_cup = 0;
  digitalWrite(dir1, LOW);
  digitalWrite(dir2, LOW);
}

/*------------------bottle control-----------------------*/
void bottlemove() { //Service Function moving bottle from initial rest position (Open Loop Control)
  //D = 255; // CHOSEN BOTTLE MOTOR VOLTAGE VALUE (CHANGE SIGN & MAGNITUDE IF NECESSARY)
  if (D > MAX_PWM_VOLTAGE) { //Ensure that you don't go past the maximum possible command
      D = MAX_PWM_VOLTAGE;
  }
  else if (D < -MAX_PWM_VOLTAGE) {
      D = -MAX_PWM_VOLTAGE;
  }
  Serial.print("PWM setting (0~255) for bottle motor is: ");
  Serial.println(D);
  //speed magnitude control
  ledcWrite(ledChannel_1, abs(D));
```

```
  //Map the motor directionality
  //control dir3 and dir4
  if (D > 0) {
    digitalWrite(dir3, LOW);
    digitalWrite(dir4, HIGH);
  }
  else if (D < 0) {
    digitalWrite(dir3, HIGH);
    digitalWrite(dir4, LOW);
  }
  else {
    digitalWrite(dir3, LOW);
    digitalWrite(dir4, LOW);
  }
}


void bottlefreeze() {
  D = 0;
  ledcWrite(ledChannel_1, D);
  digitalWrite(dir3, LOW);
  digitalWrite(dir4, LOW);
}

void bottleretract() {
  //D = -255; // CHOSEN BOTTLE MOTOR VOLTAGE VALUE (CHANGE SIGN & MAGNITUDE IF NECESSARY)
  D = -100;
  Serial.print("PWM setting (0~255) for bottle motor is: ");
  Serial.println(D);
  /*if (D > MAX_PWM_VOLTAGE*2/3) { //Ensure that you don't go past the maximum possible command
    D = MAX_PWM_VOLTAGE*2/3;
  }
  else if (D < -MAX_PWM_VOLTAGE*2/3) {
    D = -MAX_PWM_VOLTAGE*2/3;
  }*/

  //speed magnitude control
  ledcWrite(ledChannel_1, abs(D));
 //speed magnitude control
 ledcWrite(ledChannel_1, abs(D));

 //Map the motor directionality
 //control dir3 and dir4
 if (D > 0) {
   digitalWrite(dir3, LOW);
   digitalWrite(dir4, HIGH);
 }
 else if (D < 0) {
   digitalWrite(dir3, HIGH);
   digitalWrite(dir4, LOW);
 }
 else {
   digitalWrite(dir3, LOW);
   digitalWrite(dir4, LOW);
 }
}
```