BUILDING A FULLY FUNCTIONING TENNIS BALL LAUNCHER WITH AN AUTOMATIC BALL FEEDER

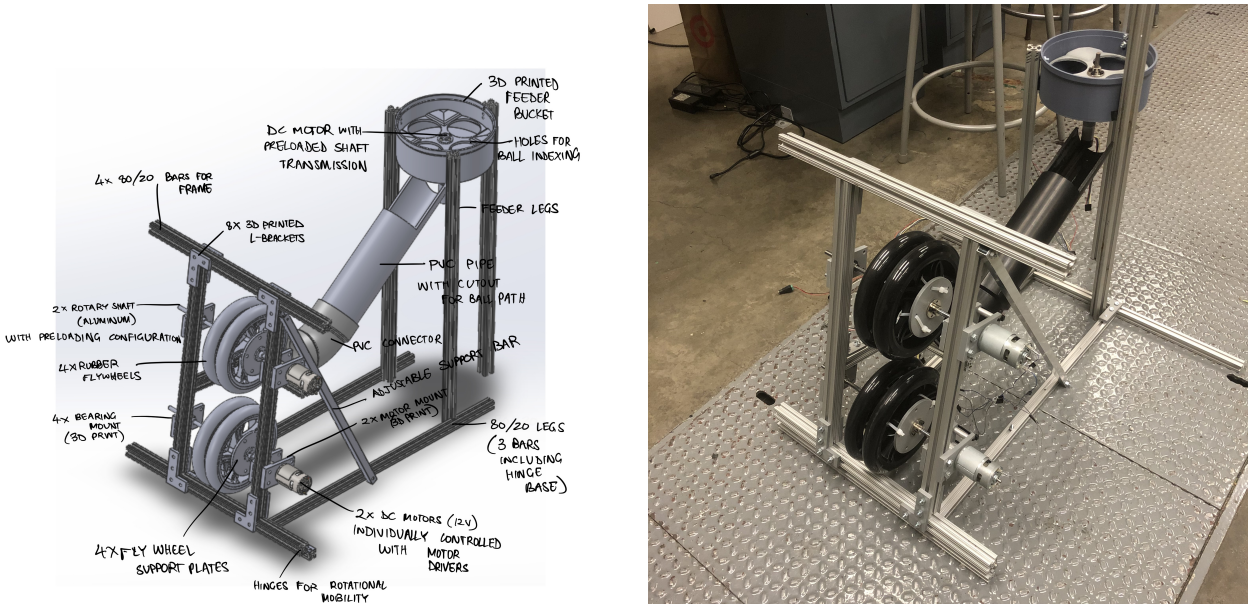
Aryan Sood, Fernando Medina, Masayuki Ohashi, Zeyad Alhomeida

Department of Mechanical Engineering, University of California at Berkeley
12/14/2023

1. Objectives and Strategies

1.1 Nomenclature

To be fully clear in our nomenclature and the names we have given to our parts, we included figure 1 which has both the fully labeled CAD as well as the physical system. For ease and clarity, we have labeled our CAD instead of the physical system.



(a) Labeled CAD

(b) Fully integrated physical system

Figure 1: Labeled diagram with both CAD and the physical system

1.2 Opportunity

The objective of this project is to design and engineer a tennis ball launcher with a feeder system. This device is intended to aid solo tennis players in practicing and enhancing their skills on the court. It's also an opportunity for us, as mechanical engineers, to demonstrate our engineering skills and techniques by creating a cost-efficient working prototype.

The tennis ball launcher offers several benefits:

- **Precision and Consistency**: The launcher is designed to deliver precise and consistent ball placement. It has adjustable settings for speed, spin, and shot placement, allowing players to fine-tune their strokes with each repetition.
- **Effective Solo Practice**: The device excels at enabling solo practice. Players no longer need to rely on finding a hitting partner or coordinating schedules.
- **Time Efficiency and Convenience**: With the tennis ball launcher, players can practice independently, honing their skills at their own pace and convenience.

The operation of the tennis ball launcher is based on a strategy that combines mechanical engineering principles with advanced control systems. It's a versatile and efficient training tool for tennis players.

1.3 High-Level Strategy

The system's behavior is characterized by two primary frames of operation:

- **Dual Flywheel System**: This system consists of two coaxial independently rotating flywheels powered by motors. The speed and direction of these motors can be controlled to adjust the speed, spin, and trajectory of the launched ball. This system allows for a wide range of shot types, from groundstrokes to lobs, mimicking the unpredictability of a real game.
- **Automatic Ball Feeder System**: This system ensures a steady supply of balls to the flywheel system. It can be programmed to deliver balls at regular intervals, allowing for continuous practice without interruption.

In conclusion, this project aims to leverage mechanical engineering principles and advanced control systems to create a tennis ball launcher that enhances the practice experience for solo tennis players. It's a testament to our engineering skills and a step towards making tennis practice more efficient and effective.

2. Methodology and Analysis

The assembly of a tennis ball launcher flywheel system is a meticulous process that involves two separate subassemblies: the Dual Flywheel System and the Automatic Ball Feeder System. The entire assembly can be seen in figure **??**

2.1 Calculations

$F_c = 2F$

$$F = \frac{F_c}{2} = \frac{15N}{2} = 7.5N$$

Since the flywheels are centered between the two bearings, the bearings will each experience a load of $F/2 = 3.75N$. Shear stress $\tau = T * r/J$ $J = \pi * (d^4)/32$

$d = 0.008m$

$$J = \frac{\pi * (d^4)}{32} = \frac{\pi * (0.008m)^4}{32} = 3.176e - 9m^4$$

Maximum Shear Stress on rotary rods:

$$\tau = \frac{T * r}{J} = \frac{0.05625Nm * 0.0075m}{3.176e - 9m^4} = 105MPa$$

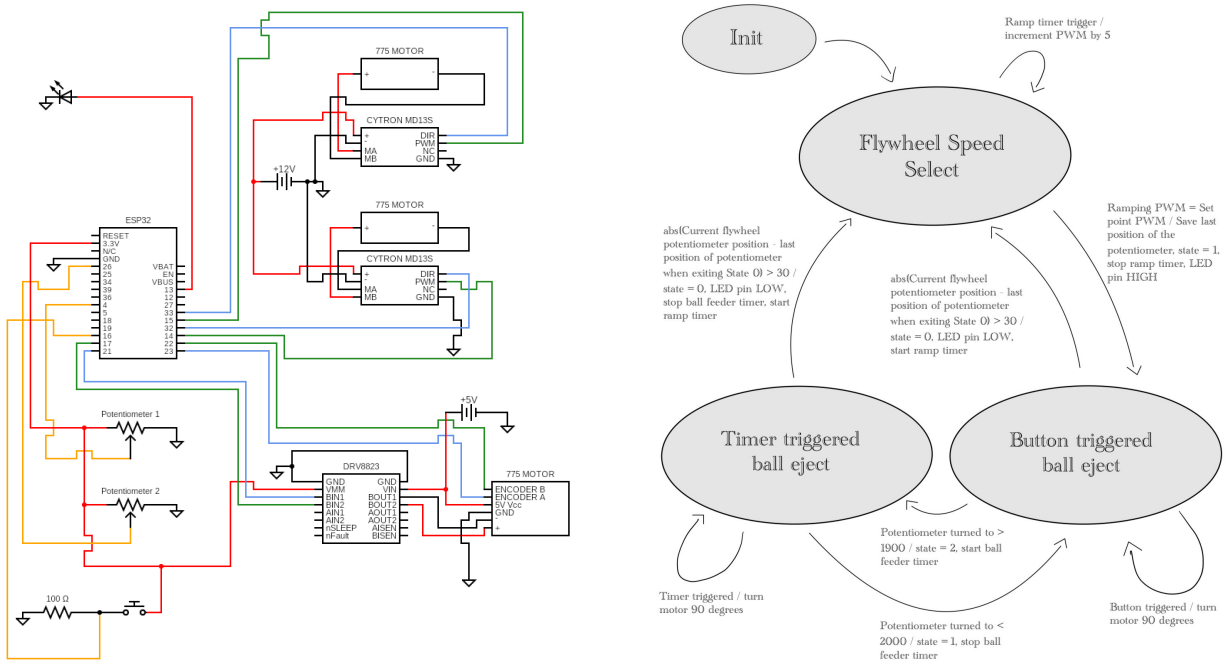
2.2 Function Critical Decisions

The maximum shear stress that the rotary shafts will experience is **105 MPa**, so aluminum with a shear strength of **210 MPa** is an ideal cost-efficient choice. The length is arbitrarily chosen to accommodate the preloading components and the dual-flywheel system. 80/20s are chosen for the main launcher frame as they allow modularity and ease in fastening and adjusting components. They are also durable and stress-resistant per load. We decided to add hinges to provide an option to change the serve angle for training. All 3D Printed components are custom-designed for efficiency and the geometries are selected for rigidity. The flywheels are off-the-shelf for cost-efficiency and

the rubber texture provides great grip on the tennis balls. They are attached to the rotary shafts using 3D-printed plates and securely positioned using shaft collars to prevent lateral sliding. Dual flywheels empirically provide better launch reliability than single flywheels due to increased surface area. 12V DC Motors are used with high-RPM and low-torque configurations to efficiently achieve the 70mph launch speed necessitated for the flywheels. The required torque for launch is low enough to not be a consideration in motor selection.

3. Software and Electronics

To illustrate the system's functionality and behavior, a detailed circuit diagram is provided in figure 2a as a visual representation of our electrical components. Orange wires represent user inputs, and blue and green wires represent signals from various electrical components. Figure 2b is our state transition diagram which depicts the dynamic states and transitions within our project.



(a) Circuit Diagram

(b) State Transition Diagram

Figure 2: Electronics and software inner workings

4. Reflection

One thing that worked well for our group was that we met frequently which allowed us to get things done quickly in a shorter period of time. However, we often underestimated the time to would take to do certain tasks; so, something we would do differently is to finish up all the tasks early no matter how menial they may seem.

Appendix

Appendix A: CAD

Figure 3: CAD of our full assembly

Figure 4: CAD of our shooter subsystem

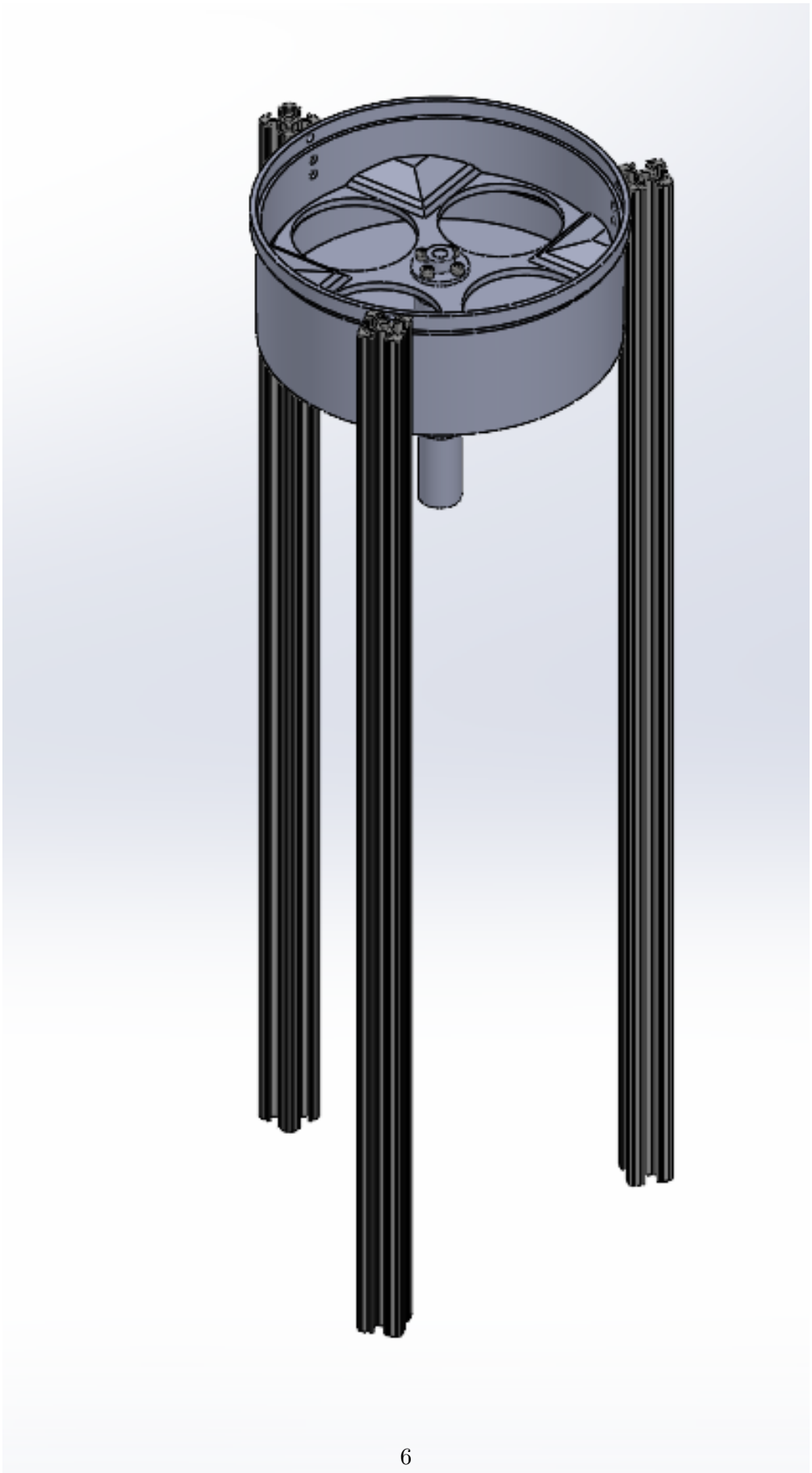


Figure 5: CAD of our feeder subsystem

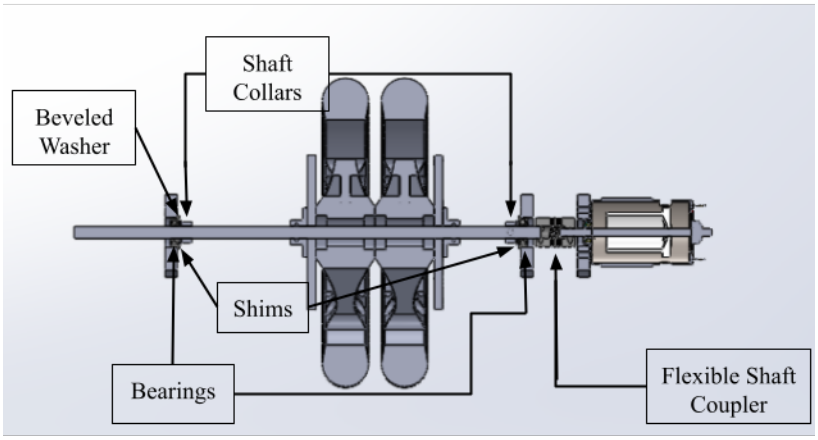


Figure 6: Labeled photo our shooter transmission

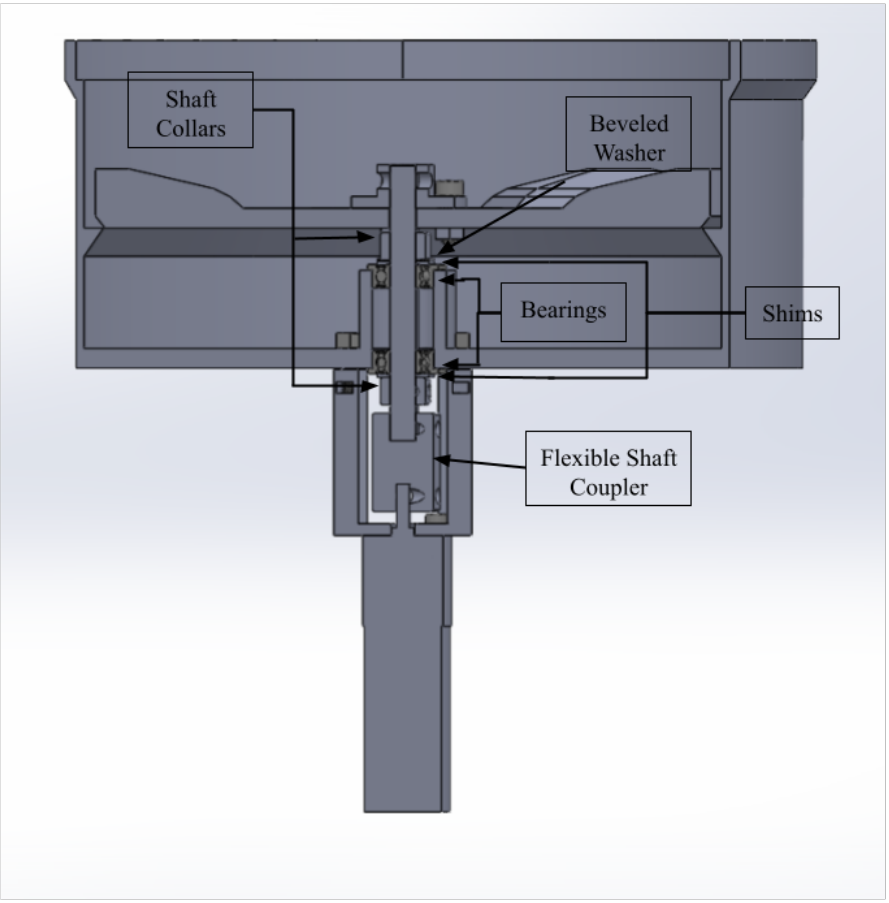


Figure 7: Labeled photo of feeder transmission

Appendix B: BOM

Table 1: List of Items and Prices

Item Name	Price (ea.)	Quantity	Vendor
uxcell Flanged Ball Bearing	$6.49	1	Amazon
M8x16mmx1mm Flat Washers	$4.49	1	Amazon
uxcell Round Steel Rod	$6.99	1	Amazon
5 Pcs 8mm Shaft Lock Collar	$5.99	1	Amazon
2 Pcs Aopin Flexible Beam Couplings	$11.69	1	Amazon
Flange Shaft Coupling	$7.83	1	Amazon
FREEDARE wheels	$19.99	1	Amazon
Linear Motion Rod	$9.99	1	Amazon
corner braces	$3.72	1	Home Depot
bulk fasteners	$5.11	1	ace
Narrow hinge	$12.99	1	ace
bolts	$0.33	20	ace
nuts	$0.16	20	ace
bolts	$0.59	15	ace
nuts	$0.16	15	ace
uxcell Flanged Ball Bearing	$6.49	1	Amazon
Scooter Wheels	$29.99	1	Amazon
Saiper Flexible Couplings	$8.99	1	Amazon
ELEGOO PLA+ Filament	$15.29	1	Amazon
Sydien Rigid Flange	$9.49	1	Amazon
YOTINO Flexible Couplings	$9.99	1	Amazon
DC 775 Motor	$17.99	2	Amazon
GeeekPi IIC LCD	$10.99	1	Amazon
w d Belleville Disc Spring	$4.11	1	McMaster
8020 1inch	-	5	second hand
8020 2cm	-	5	second hand
Encoder Metal Gearmotor 12V DC	-	1	second hand
Breadboard	-	1	micro-kit
Various wires	-	-	micro-kit
PVC pipe	$15.00	1	Home Depot
PVC 45 elbow	$30.00	1	Home Depot
Total	$304.64		

Appendix C: Step-by-step Assembly Process

Flywheel System

The assembly of the tennis ball launcher flywheel system begins with gathering components and attaching motor mounts to an 80/20. The motor is added, a flexible shaft cover is placed, and a ball bearing is inserted into the shaft coupling. The transmission system is preloaded, and a flywheel support plate and two flywheels are added and fastened. Shaft collars are placed for adjustment, and the ball bearing arrangement is repeated for the bottom assembly. The assembled 80/20s are mounted to create a rigid frame, and hinges are added for rotation. Finally, an adjusting rod is fastened between the flywheels to complete the assembly.

Feeder System

The feeder system is supplementary and an extension to the main flywheel system for launching, extending beyond the required scope. It contains mostly 3D printed parts: the bucket, the legs, and and the sorting plate. The motor-shaft assembly for this is also preloaded with bearings, shims, and disc springs. It stabds independent from the launcher using legs and the ball travels through the PVC pipe with a cutout that is attached using an 80/20 L shape, after falling through the feeder hole and is fed directly into the flywheels.

Appendix D: Code

```
#include <ESP32Encoder.h>
#define BIN_1 21 // BALL FEEDER PWM A
#define BIN_2 17 // BALL FEEDER PWM B
#define Encoder_1 23 // ENCODER A PIN
#define Encoder_2 22 // ENCODER B PIN
#define LED_PIN 13 // LED PIN
#define POT 4  // BALL EJECT MODE POTENTIOMETER PIN
#define BTN 16 // BUTTON PIN
#define PWM 14 // FLYWHEEL 1 PWM PIN
#define DIR 32 // FLYWHEEL 1 DIRECTION PIN
#define PWM2 15 // FLYWHEEL 2 PWM PIN
#define DIR2 33 // FLYWHEEL 2 DIRECTION PIN
#define POTFLY 26 // FLYWHEEL SPEED POTENTIOMETER PIN

ESP32Encoder encoder;

// Potentiometer Variables
-----------------------------------------------------------
int D = 0;
int D2 = 0;
int D3 = 0;
int potReading = 0;
int potReading2 = 0;

// Flywheel Ramp Variables
-----------------------------------------------------------------
int Pwm = 0; // PWM given to the motors
int setpoint = 0; // PWM the incrementing stops at
const int rampIncrement = 5; // Amount the PWM increments every time timer 3 is
called

// Cascaded Position Controller Variables
-----------------------------------------------
float err = 0;
float sumerr = 0; // Anti-windup variable
float Kp = 1.7;  // Portional Gain
float Ki = 0.02; // Integral Gain
int theta = 0; // Motor Postion
int thetaDes = 0; // Desired Motor Postion
volatile int count = 0; // encoder count
int counter = 0;

//Setup Timer variables -------------------------------
hw_timer_t* timer0 = NULL; // Timer 0
hw_timer_t* timer1 = NULL; // Timer 1
hw_timer_t* timer2 = NULL; // Timer 2
hw_timer_t* timer3 = NULL; // Timer 3
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux2 = portMUX_INITIALIZER_UNLOCKED;
```

```
portMUX_TYPE timerMux3 = portMUX_INITIALIZER_UNLOCKED;

// setting PWM properties ---------------------------
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int ledChannel_3 = 3;
const int ledChannel_4 = 4;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;
const int NOM_PWM_VOLTAGE = 150;

int state = 0; // Intialized at state zero

// Interrupt flags
-----------------------------------------------------------------------
volatile bool buttonIsPressed = false; // True when the button is pressed
volatile bool debounceT = false; // When timer 2 is triggered
volatile bool timedTurn = false; // When timer 0 is triggered
volatile bool deltaT = false; // When timer 1 is triggered
volatile bool Ramped = false;  // True when the Flywheel PWM is the same as the
setpoint

//Initialization
-----------------------------------------------------------------------

// Timer used for the ball feeder mode that ejects a ball every 5 seconds
void IRAM_ATTR onTime0() {
  portENTER_CRITICAL_ISR(&timerMux0);
  timedTurn = true;
  portEXIT_CRITICAL_ISR(&timerMux0);
}

// Timer used in the casaded position controller for the ball feeder motor
void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux1);
  count = encoder.getCount();
  encoder.clearCount();
  deltaT = true;
  portEXIT_CRITICAL_ISR(&timerMux1);
}

// Interrupt called when a button press is triggered
void IRAM_ATTR isr() {
  buttonIsPressed = true;
  timerStart(timer2);
}

// Timer used to debounce the button
void IRAM_ATTR onTime2() {
```

```
  portENTER_CRITICAL_ISR(&timerMux2);
  debounceT = true;
  portEXIT_CRITICAL_ISR(&timerMux2);
  timerStop(timer2);
}

// Timer that increments the pwm given to the flywheel motors
void IRAM_ATTR onTime3() {
  if (setpoint - Pwm > 0) {
    Pwm += rampIncrement;
  } else if (setpoint - Pwm < 0) {
    Pwm -= rampIncrement;
  } else if (Pwm = setpoint) {
    Ramped = true;
  }
}

void setup() {

  pinMode(POT, INPUT);
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);  // Sets the initial state of LED as turned-off
  pinMode(POTFLY, INPUT);
  pinMode(DIR, OUTPUT);
  digitalWrite(DIR, LOW); // Sets the direction of the Flywheel motor 1
  pinMode(DIR2, OUTPUT);
  digitalWrite(DIR2, LOW); // Sets the direction of the Flywheel motor 2

  pinMode(BTN, INPUT);
  attachInterrupt(BTN, isr, RISING);  // set "BTN" pin as the interrupt pin that
calls function named "isr" when the interrupt is triggered; "Rising" means
triggering interrupt when the pin goes from LOW to HIGH
  Serial.begin(115200);

  ESP32Encoder::useInternalWeakPullResistors = UP; // Enable the weak pull up
resistors
  encoder.attachHalfQuad(Encoder_1, Encoder_2); // Attach 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);
  ledcSetup(ledChannel_3, freq, resolution);
  ledcSetup(ledChannel_4, freq, resolution);

  // Attach the channel to the GPIO to be controlled
  ledcAttachPin(BIN_1, ledChannel_1);
  ledcAttachPin(BIN_2, ledChannel_2);
  ledcAttachPin(PWM, ledChannel_3);
```

```
  ledcAttachPin(PWM2, ledChannel_4);

  // Initilize timers
  timer0 = timerBegin(0, 80, true);              // timer 0, MWDT clock period =
12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer0, &onTime0, true);  // edge (not level) triggered
  timerAlarmWrite(timer0, 5000000, true);        // 5000000 * 1 us = 5 s, autoreload
true

  timer1 = timerBegin(1, 80, true);              // timer 1, MWDT clock period =
12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer1, &onTime1, true);  // edge (not level) triggered
  timerAlarmWrite(timer1, 10000, true);          // 10000 * 1 us = 10 ms, autoreload
true

  timer2 = timerBegin(2, 80, true);              // timer 2, MWDT clock period =
12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer2, &onTime2, true);  // edge (not level) triggered
  timerAlarmWrite(timer2, 300000, true);         // 300000 * 1 us = 300 ms,
autoreload true

  timer3 = timerBegin(3, 80, true);              // timer 3, MWDT clock period =
12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer3, &onTime3, true);  // edge (not level) triggered
  timerAlarmWrite(timer3, 1000000, true);        // 1000000 * 1 us = 1 s, autoreload
true


  // Enable the timer alarms
  timerAlarmEnable(timer0);  // enable
  timerAlarmEnable(timer1);  // enable
  timerAlarmEnable(timer2);  // enable
  timerAlarmEnable(timer3);  // enable

  // Stop the timers
  timerStop(timer0);
  timerStop(timer2);
  timerStop(timer3);
}

void loop() {

  TurnMotor(); // Control the ball feeder position
  potReading = analogRead(POT); // Reads ball eject mode potentiometer
  potReading2 = analogRead(POTFLY); // Reads flywheel PWM potentiometer
  D2 = map(potReading2, 0, 4095, 0, MAX_PWM_VOLTAGE); // Maps the Flywheel PWM to
possible PWMs

  // State machine
  switch (state) {
```

```
    // State that ramps the fly wheel motors to a desired PWM
    case 0:
      ledcWrite(ledChannel_3, Pwm);
      ledcWrite(ledChannel_4, Pwm);

      CheckForRamp();
      break;

    // State that ejects one ball per button press
    case 1:
      if (CheckForButtonPress()) {
        Turn90Deg();
      } else if (potReading <= 2100) {
        timerWrite(timer0, 0);
        timerStart(timer0);
        state = 2;
        //Serial.println("1 to 2");
      } else if (abs(D2 - D3) > 5) {
        state = 0;
        digitalWrite(LED_PIN, LOW);
        //Serial.println("1 to 0");
      }
      break;

    // State that ejects 1 ball every 5 seconds
    case 2:
      if (potReading >= 2000) {
        timerStop(timer0);
        state = 1;
        //Serial.println("2 to 1");
      } else if (timedTurn) {
        Turn90Deg();
        timedTurn = false;
      } else if (abs(D2 - D3) > 30) {
        timerStop(timer0);
        state = 0;
        digitalWrite(LED_PIN, LOW);
        //Serial.println("2 to 0");
      }
      break;
  }
  //plotControlData();
  //Serial.println(state);
}

// Event Checkers ---------------------------------

// Button Press Event Checker
bool CheckForButtonPress() {
```

```
  if (debounceT && buttonIsPressed) {
    portENTER_CRITICAL(&timerMux2);
    debounceT = false;
    portEXIT_CRITICAL(&timerMux2);
    timerStop(timer2);
    buttonIsPressed = false;
    return true;
  } else
    return false;
}

// Flywheel PWM Ramp Event Checker
void CheckForRamp() {
  if (D2 < 90) {
    Pwm = 0;
  } else if (D2 >= 110 && D2 <= 190) {
    setpoint = 50;
    if (Pwm != setpoint && !timerStarted(timer3)) {
      timerStart(timer3);
    } else if (Ramped) {
      Ramped = false;
      timerStop(timer3);
      state = 1;
      D3 = D2;
      digitalWrite(LED_PIN, HIGH);
      //Serial.println("0 to 1");
    }
  } else if (D2 > 210) {
    setpoint = 100;
    if (Pwm != setpoint && !timerStarted(timer3)) {
      timerStart(timer3);
    } else if (Ramped) {
      Ramped = false;
      timerStop(timer3);
      state = 1;
      D3 = D2;
      digitalWrite(LED_PIN, HIGH);
      //Serial.println("0 to 1");
    }
  }
}

// Event Service Fuctions -------------------------

// Sets the desired postion to + 90 degrees which ejects one ball
void Turn90Deg() {
  thetaDes += 122;
  counter += 1;
}
```

```
// Ball feeder motor cascaded postion controller
void TurnMotor() {
  if (deltaT) {
    portENTER_CRITICAL(&timerMux1);
    deltaT = false;
    portEXIT_CRITICAL(&timerMux1);

    theta += count;

    err = thetaDes - theta;
    sumerr = sumerr + err;
    D = Kp * (err + (Ki * sumerr));

    if (D > MAX_PWM_VOLTAGE) {
      D = MAX_PWM_VOLTAGE;
      sumerr -= err;
    } else if (D < -MAX_PWM_VOLTAGE) {
      D = -MAX_PWM_VOLTAGE;
      sumerr -= err;
    }
    if (D > 0) {
      ledcWrite(ledChannel_1, LOW);
      ledcWrite(ledChannel_2, D);
    } else if (D < 0) {
      ledcWrite(ledChannel_1, -D);
      ledcWrite(ledChannel_2, LOW);
    } else {
      ledcWrite(ledChannel_1, LOW);
      ledcWrite(ledChannel_2, LOW);
    }
  }
}


// Functions ---------------------------------

// Debug print controller data
void plotControlData() {
  Serial.print("Position:");
  Serial.print(theta);
  Serial.print(" ");
  Serial.print("Desired_Position:");
  Serial.print(thetaDes);
  Serial.print(" ");
  Serial.print("PWM_Duty:");
  Serial.println(D);
}
```