KinetiHoop

ME 102B Final Report

Team 3: Parth Behani, Aarav Goel, Aathavan Senthilkumar, Bryan Yim

Opportunity

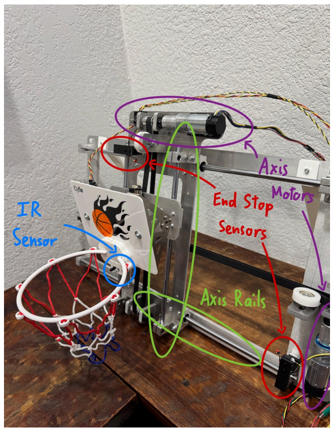
In an effort to fix the staleness of classic arcade games, we wanted to make a new, more interactive, and more engaging arcade game. We decided to focus on a variation of the age-old basketball shooting game, where a player is challenged to score as many shots as possible into a stationary hoop in a fixed amount of time. To make this more exciting, we aimed to make a 3-axis hoop which can translate in the x and y direction as well as tilt the backboard forwards and backwards. We wanted the game to be more dynamic where, as the player scores more points, the hoop could move in increasingly more erratic and difficult-to-make patterns. Additionally, we imagined a two player game where one player tries to land as many shots as they can while the other player can control the hoop's movement to try and make their opponent miss.

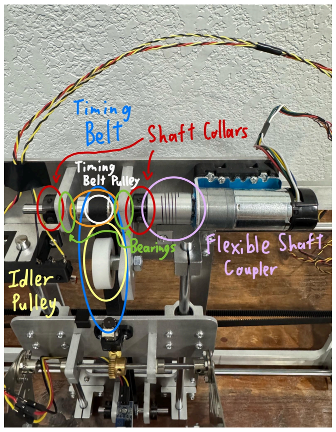
Strategy

Our high level strategy was to translate vertically and horizontally using timing belts and tilt forwards and backwards using a worm gear. For each translation axis, a motor spins a shaft with an attached pulley. This pulley drives a timing belt which is connected to a moving plate that rides along linear rails. Before each game, the device would calibrate each axis by driving each motor until a mechanical end stop switch is hit, identifying the zero or "home" coordinate. The player has access to a push button to advance through each state of the game and a screen showing which guides the player on what each state does and displays score.

Our achieved specifications differed slightly from what we initially imagined. We achieved x and y axis motion, but were unable to integrate the tilting mechanism due to a few different factors including ESP32 pin limitations, motor driver issues, and a lack of time for testing. We initially hoped to be able to span either axis in less than 1 second, but that ended up being closer to 1.75 seconds. We believe we can still reach 1 second using a higher voltage power supply, however, we found that this was more than sufficient for a human reaction time anyways. In addition to manual control, we would have liked to implement automatic trajectory planning, but were unable to do so within the semester.

Physical Device





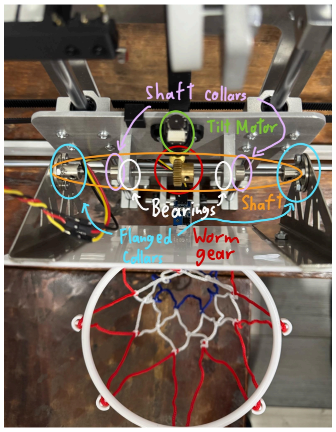
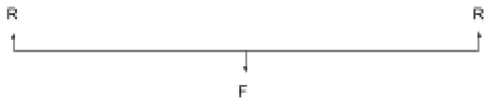


Figure 1, 2, & 3: From left to right: photos of fully assembled machine, pulley sub-assembly, and tilt sub-assembly with sensors and actuators highlighted

Decisions & Calculations

Linear guide rail thickness requirements:

The solid steel linear guide rails are our single heaviest moving components and therefore the lowest hanging fruit to save weight, and so with limited time, we first determined how light we could make them, starting with the smallest 8 mm diameter available in 0.5 m length.



Each x-axis guide rail, since there are two of them, must support half of the weight of the entire y-axis and tilt sub-assemblies. Though in reality the beam experiences a load at two points spaced ⅓ apart along the beam's length, we simplified the problem to be a single point load at the center as a worst case approximation. The standard formula for the maximum deflection at the center of a beam simply supported at both ends is:

$$\delta_{max} = \frac{FL^3}{48EI}$$

Where the applied force $F = mg/2 = 0.5 * 1.192kg * 9.81m/s^2/2 = 5.85N$, length $L = 0.5m$, young's modulus: $E = 200 * 10^9 Pa$, and second moment of area: $I = \frac{\pi D^4}{64} = \frac{\pi(0.008m)^4}{64} = 2.01 * 10^{-10} m^4$. This results in $\delta_{max} = 0.37mm$ which is more than tolerable without causing alignment issues, validating our choice of 8 mm diameter guide rail.

Torque and speed requirements:

For the x-axis motor, the aim was to determine a cost-efficient, high torque motor to carry the weight of the gantry system including the y-axis system. Accounting for the weight of the system, approximately 1192 grams, and the pulley diameter of 10 mm with a distance of 200 mm to be traversed in .5 seconds we can calculate desired torque.

First starting with acceleration:

$$a = \frac{2 \cdot 0.2}{0.5^2} = \frac{0.4}{0.25} = 1.6\,\mathrm{m/s^2}$$

We obtain static, dynamic, and total force:

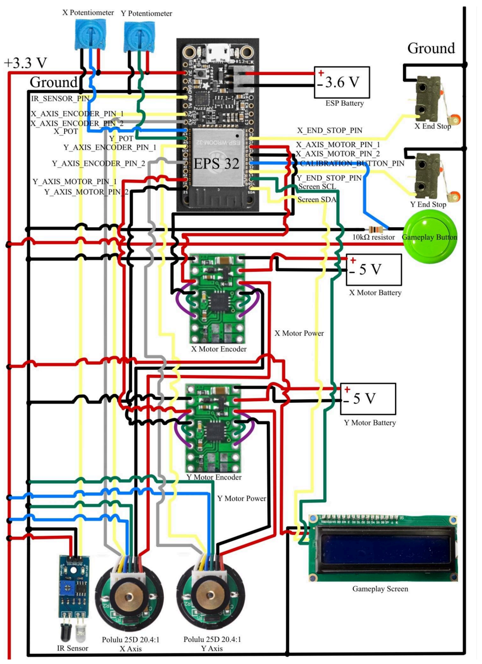
$$F_s = 1.192 \cdot 9.81 = 11.70\,\mathrm{N}$$
$$F_d = 1.192 \cdot 1.6 = 1.91\,\mathrm{N}$$
$$F_{total} = 11.70 + 1.91 = 13.61\,\mathrm{N}$$

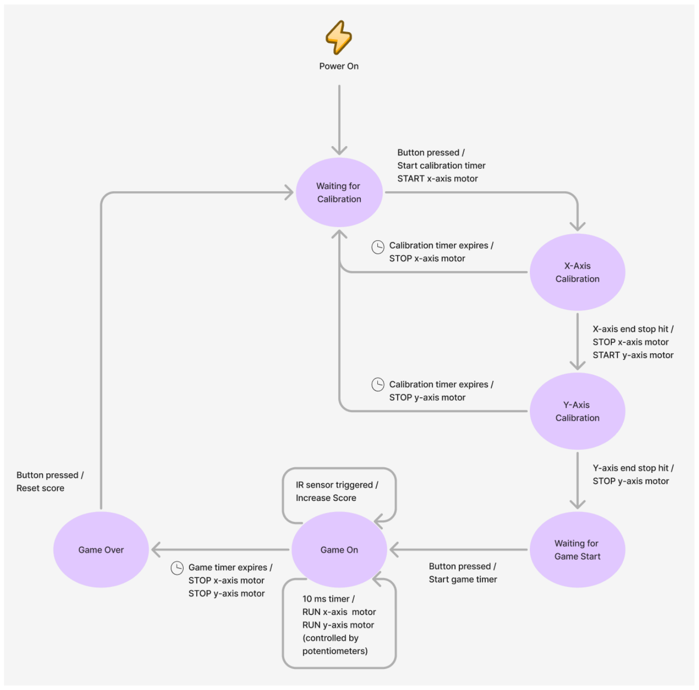
This allows us to get torque, which then, using tension ratio equations, helps us calculate the tension in the belts:

$$\tau = 13.61 \cdot 0.005 = 0.06805\,\text{N·m}$$
$$T_1 - T_2 = \frac{\tau}{r} = \frac{0.06805}{0.005} = 13.61\,\text{N}$$
$$\frac{T_1}{T_2} = e^{0.3\cdot\pi} = e^{0.942} = 2.565$$
$$T_2 = \frac{13.61}{2.565 - 1} = \frac{13.61}{1.565} = 8.69\,\text{N}$$
$$T_1 = 2.565 \cdot 8.69 = 22.29\,\text{N}$$

This led to the decision of purchasing the motors with the right torque as seen in our BOM in Appendix 1. The y-axis motors underwent the same calculations albeit with a traversal of 100 mm and a mass of 343 grams.

Diagrams





Power On
Waiting for Calibration
Button pressed / Start calibration timer START x-axis motor
X-Axis Calibration
Calibration timer expires / STOP x-axis motor
X-axis end stop hit / STOP x-axis motor START y-axis motor
Y-Axis Calibration
Calibration timer expires / STOP y-axis motor
Y-axis end stop hit / STOP y-axis motor
Waiting for Game Start
Button pressed / Start game timer
Game On
IR sensor triggered / Increase Score
10 ms timer / RUN x-axis motor RUN y-axis motor (controlled by potentiometers)
Game timer expires / STOP x-axis motor STOP y-axis motor
Game Over
Button pressed / Reset score

Figure 4: Circuit diagram with all implemented components

Figure 5: State diagram with all implemented components

Reflection

Looking back at the semester, we started off with a pretty ambitious set of goals. Though we did not finish all of them, we were still able to make a project we were proud of. Things that worked well for us include: finding a project idea that all team members were genuinely excited about working on, maintaining version history of our CAD and code throughout the process, and talking often to the machine shop staff about our manufacturing and implementation ideas for their feedback. Some things that we wish we realized earlier include: breaking up official deliverables into smaller parts so we can iterate on them before the due date, having a more defined break up of who is working on what for each deliverable, and realizing that some ESP pins have special behavior on boot up or during flashing.

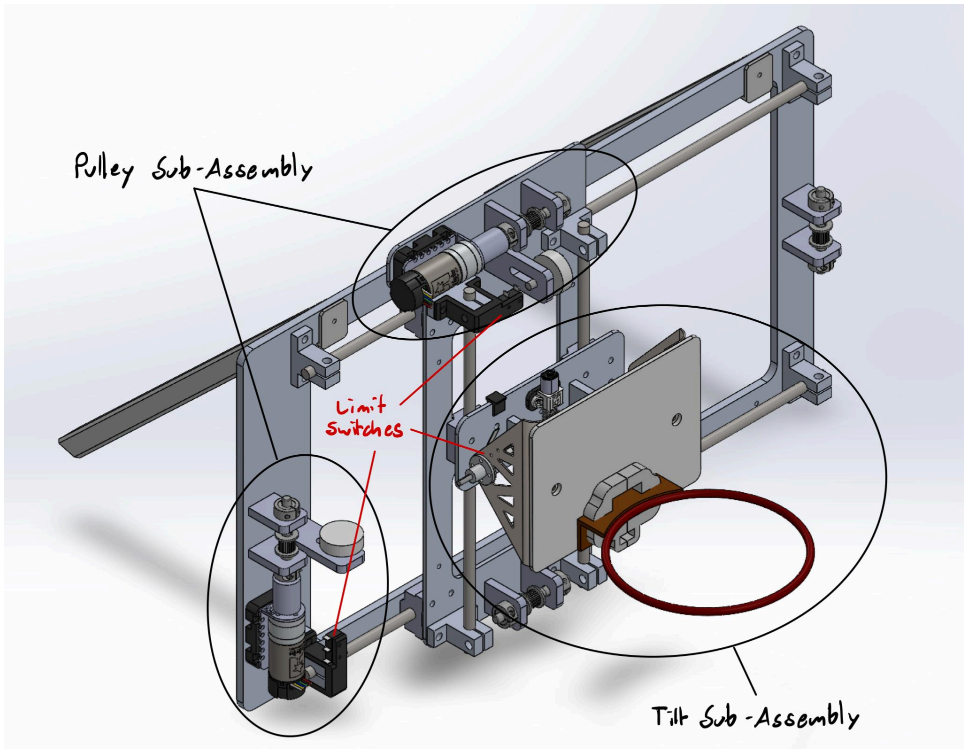
Appendix 1: BOM

Name	Quantity	Unit Cost	Total Cost	Vendor	Part Number	Link
Mechanical - Tilt						
Basketball Hoop	1	$9.89	$9.89	Amazon	B01KX246AE	https://www.amazon.com/dp/B01KX246AE
Shaft (4 mm Dia.)	1	$5.25	$5.25	McMaster	1327K507	https://www.mcmaster.com/1327K507
Shaft Collar (4 mm Bore Dia.)	10	$0.65	$6.49	Amazon	B0CQ2RDDHG	https://www.amazon.com/dp/B0CQ2RDDHG
Flanged Shaft Collar (4 mm Bore Dia.)	4	$2.50	$9.99	Amazon	B07PFVKJYT	https://www.amazon.com/dp/B07PFVKJYT
Sleeve Bearing (4 mm Bore Dia.)	2	$0.00	$0.00	Igus	LFM-0405-04	https://www.igus.com/product?artNr=LFM-0405-04
Worm + Worm Gear	1	$9.99	$9.99	Amazon	B0DFCVMC1M	https://www.amazon.com/dp/B0DFCVMC1M
Micro Motor + Encoder	1	$0.00	$0.00	Polulu	2215	https://www.pololu.com/product/2215
Micro Motor Driver	1	$0.00	$0.00	Polulu	2130	https://www.pololu.com/product/2130
Micro Motor Bracket	1	$0.00	$0.00	Polulu	989	https://www.pololu.com/product/989
Aluminum Stock (1/8" Thick)	1	$20.64	$20.64	Jacobs Material Store	N/A	https://store.jacobshall.org/products/24x24x-1-8-6061-aluminium-plate
Steel Stock (1/16" Thick)	1	$0.00	$0.00	Student Machine Shop	N/A	N/A
Mechanical - Gantry						
Linear Guide Rail Set (X-Axis)	1	$29.99	$29.99	Amazon	B0BKJJXCVN	https://www.amazon.com/dp/B0BKJJXCVN
Linear Guide Rail Set (Y-Axis)	1	$21.99	$21.99	Amazon	B0BKK68J2T	https://www.amazon.com/dp/B0BKK68J2T
Shaft (1/4" Dia.)	1	$8.37	$8.37	McMaster	1327K66	https://www.mcmaster.com/1327K66
Shaft Collar (1/4" Bore Dia.)	8	$2.50	$19.98	Amazon	B07GTC62MH	https://www.amazon.com/dp/B07GTC62MH
Sleeve Bearing (1/4" Bore Dia.)	4	$0.00	$0.00	Igus	JFI-0405-04	https://www.igus.com/product?artNr=JFI-0405-04
Timing Belt Pulley	5	$1.60	$7.99	Amazon	B07BT6MVXB	https://www.amazon.com/dp/B07BT6MVXB
Timing Belt	1	$16.99	$16.99	Amazon	B097T4DFM6	https://www.amazon.com/dp/B097T4DFM6
Timing Belt Clamp	10	$1.30	$12.99	Amazon	B0894BG3VK	https://www.amazon.com/dp/B0894BG3VK
Idler Pulley	2	$0.00	$0.00	Student Machine Shop	N/A	N/A
Flexible Shaft Coupler	5	$2.70	$13.49	Amazon	B09137356F	https://www.amazon.com/dp/B09137356F
Motor	2	$48.95	$97.90	Polulu	4843	https://www.pololu.com/product/4843
Motor Driver	2	$11.95	$23.90	Polulu	2999	https://www.pololu.com/product/2999
Motor Bracket	2	$3.98	$7.96	Polulu	2676	https://www.pololu.com/product/2676

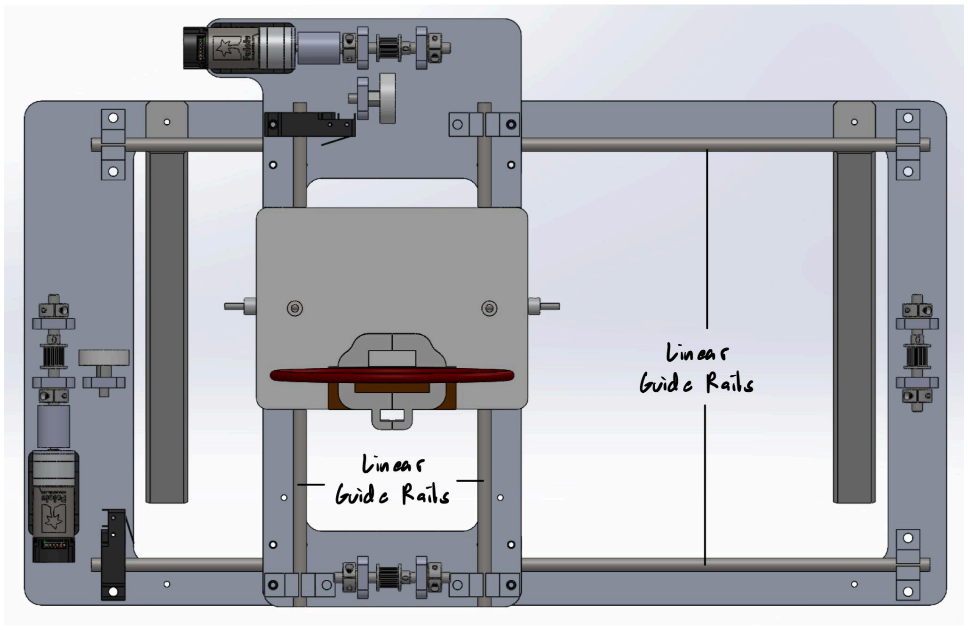
Aluminum Stock (1/4" Thick)	1	$0.00	$0.00	Student Machine Shop	N/A	N/A
Mechanical - Misc.						
Screws, Nuts, + Washers	1	$14.98	$14.98	Amazon	B0BN297MP5	https://www.amazon.com/dp/B0BN297MP5
Filament	1	$0.00	$0.00	Bambu	N/A	https://us.store.bambulab.com/products/pla-basic-filament
Electrical						
ESP32	1	$0.00	$0.00	Adafruit	3619	https://www.adafruit.com/product/3619
USB Cable	1	$0.00	$0.00	Adafruit	4474	https://www.adafruit.com/product/4474
Power Supply	1	$0.00	$0.00	Adafruit	276	https://www.adafruit.com/product/276
Breadboard	2	$0.00	$0.00	Polulu	352	https://www.pololu.com/product/352
Jumper Cable	1	$0.00	$0.00	Amazon	B07QXXMWRZ	https://www.amazon.com/dp/B07QXXMWRZ
Wire	1	$0.00	$0.00	Student Machine Shop	N/A	N/A
Resistor	1	$0.00	$0.00	Sparkfun	10969	https://www.sparkfun.com/products/10969
Button	1	$0.00	$0.00	Student Machine Shop	N/A	N/A
Potentiometer	2	$0.00	$0.00	Adafruit	356	https://www.adafruit.com/product/356
IR Sensor	1	$0.00	$0.00	Amazon	B07PFCC76N	https://www.amazon.com/dp/B07PFCC76N
Limit Switch	10	$0.60	$5.99	Amazon	B07X142VGC	https://www.amazon.com/dp/B07X142VGC
Screen	3	$4.00	$11.99	Amazon	B0BWTFN9WF	https://www.amazon.com/dp/B0BWTFN9WF
* All items with a cost of $0.00 were either provided for free by class resources or pre-owned by the team					Sum Total Cost	$356.76

Appendix 2: CAD Screenshots

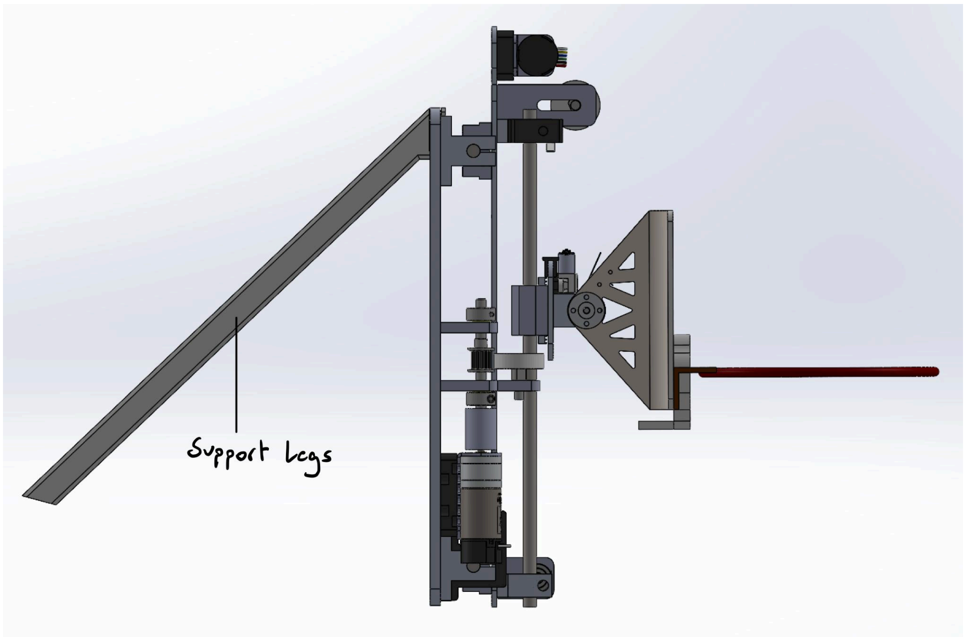
Isometric View:



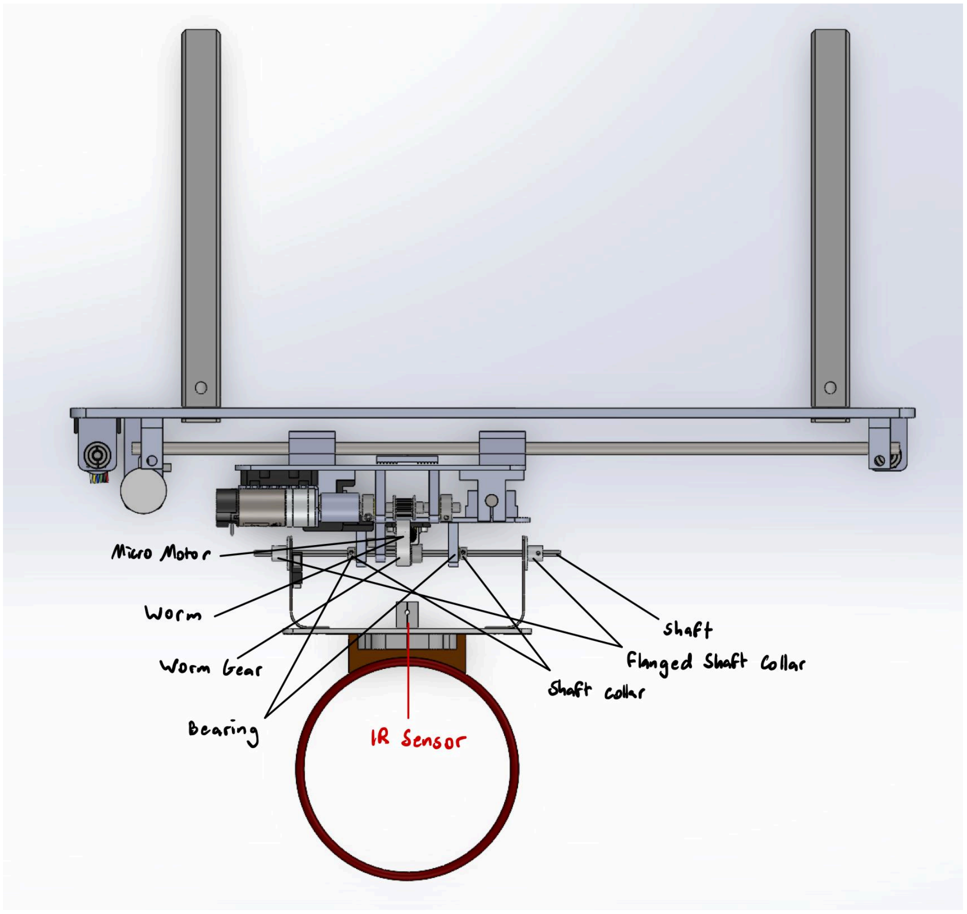
Front Orthographic View:



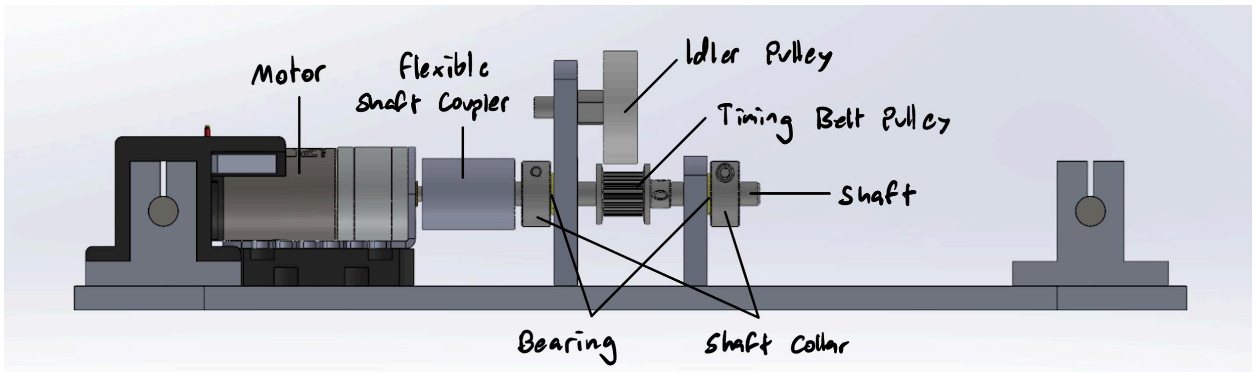
Left Orthographic View:



Top Orthographic View:



Pulley Sub-Assembly:



Appendix 3: Complete code

```
#include <Arduino.h>
#include <ESP32Encoder.h>
#include <LiquidCrystal_I2C.h>

// ESP32Encoder encoder;

int lcdColumns = 16;
int lcdRows = 2;

LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);
//IR, button, end-stop pins
#define IR_SENSOR_PIN 26      //Pin for IR Sensor input
#define CALIBRATION_BUTTON_PIN 32     // Pin for calibration button

#define X_END_STOP_PIN 27          // Pin for X-axis limit switch
#define Y_END_STOP_PIN 14          // Pin for Y-axis limit switch

//motor control pins
#define X_AXIS_MOTOR_PIN_1 33        // Pin 1 for X-axis motor
#define X_AXIS_MOTOR_PIN_2 15        // Pin 2 for X-axis motor
#define Y_AXIS_MOTOR_PIN_1 21       // Pin 1 for Y-axis motor 21
#define Y_AXIS_MOTOR_PIN_2 17        // Pin 2 for Y-axis motor 17

//motor encoder pins
#define X_AXIS_ENCODER_PIN_1 34      //Pin 1 for X-axis encoder
#define X_AXIS_ENCODER_PIN_2 39      //Pin 2 for X-axis encoder
#define Y_AXIS_ENCODER_PIN_1 5      //Pin 1 for Y-axis encoder 5
#define Y_AXIS_ENCODER_PIN_2 19      //Pin 2 for Y-axis encoder 19

#define X_POT 36
#define Y_POT 4

//timer definitions
#define TIMER_FREQUENCY 1000000
#define TIMEOUT_DURATION 5 * 1000     // Timeout duration in milliseconds

//setting PWM properties
#define PWM_FREQUENCY 5000
#define PWM_RESOLUTION 8
#define ledChannel_1 1
#define ledChannel_2 2
#define MAX_PWM_VOLTAGE 255
#define NOM_PWM_VOLTAGE 255

//Create encoders
ESP32Encoder X_ENCODER;
ESP32Encoder Y_ENCODER;

//Motor controller for x
int Kp = 5;
int Ki = 1;
int IMax = 0;
float error = 0;
float sum = 0;

volatile int count = 0;                   // encoder count
volatile bool deltaT = false;             // check timer interrupt 2
volatile int game_over_counter = 0;

const int revs = 13;
const int cpr = 48;
int theta = 0;
int thetaDes = 0;
int thetaMax = revs * cpr;
int D = 0;
int X_POTReading = 0;

//Motor controller for y
int Kp_y = 5;
int Ki_y = 3;
int IMax_y = 0;
float error_y = 0;
float sum_y = 0;

volatile int count_y = 0;                  // encoder count

const int revs_y = 5;
const int cpr_y = 48;
int theta_y = 0;
int thetaDes_y = 0;
int thetaMax_y = revs_y * cpr_y;
int D_y = 0;
int Y_POTReading = 0;

// Define states
enum States {
    IDLE,
    X_AXIS_CALIBRATING,
    Y_AXIS_CALIBRATING,
    READY_TO_PLAY,
    GAME_ON,
    GAME_OVER
};

States currentState = IDLE;
bool buttonPressed = false;
bool timeoutOccurred = false; // Flag to indicate timeout in main loop

// Flags to indicate if limit switches were hit
volatile bool xAxisHit = false;
volatile bool yAxisHit = false;

hw_timer_t *timer2 = NULL;  // Hardware timer
hw_timer_t* timer0 = NULL;  // Hardware timer
hw_timer_t* timer1 = NULL;  // Hardware timer
// hw_timer_t *timer3 = NULL;                // for PI controller

// portMUX_TYPE timerMux2 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux3 = portMUX_INITIALIZER_UNLOCKED;

volatile bool debouncing = false;
volatile bool madeHoop = false;
volatile bool gameOver = false;
int score = 0;
int level = 1;
```

```
volatile bool start = false;

void IRAM_ATTR calibrationButtonISR() {
  buttonPressed = true; //press flag
}

void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux3);
  count = X_ENCODER.getCount();
  X_ENCODER.clearCount();
  count_y = Y_ENCODER.getCount();
  Y_ENCODER.clearCount();
  deltaT = true;  // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux3);
  game_over_counter = game_over_counter + 1;
}

void IRAM_ATTR timeoutISR() {
  portENTER_CRITICAL_ISR(&timerMux0);
  timeoutOccurred = true;  // Set timeout flag instead of printing directly
  portEXIT_CRITICAL_ISR(&timerMux0);
}

//ISR for preventing debounce
void IRAM_ATTR debounceTimerCallback() {
  portENTER_CRITICAL_ISR(&timerMux1);
  madeHoop = false;
  buttonPressed = false;
  debouncing = false;  // Reset the debounce flag
  timerStop(timer0);   // Stop the debounce timer
  portEXIT_CRITICAL_ISR(&timerMux1);
}

//ISR for ending the game after timer ends
// void IRAM_ATTR onTime0() {
//   portENTER_CRITICAL_ISR(&timerMux0);
//   gameOver = true;
//   portEXIT_CRITICAL_ISR(&timerMux0);
// }

// ISR for X-axis limit switch
void IRAM_ATTR xAxisStopISR() {
  if (currentState == X_AXIS_CALIBRATING) {
      xAxisHit = true;  // Set flag indicating X-axis end stop was hit
  }
}

// ISR for Y-axis limit switch
void IRAM_ATTR yAxisStopISR() {
  if (currentState == Y_AXIS_CALIBRATING) {
      yAxisHit = true;  // Set flag indicating X-axis end stop was hit
  }
}

// ISR for scoring a point
void IRAM_ATTR pointTrigger() {  // the function to be called when interrupt is triggered
  madeHoop = true;
}

void setup() {
  Serial.begin(9600);
  Serial.println("Setup");

  // Calibration button interrupt
  pinMode(CALIBRATION_BUTTON_PIN, INPUT);
  attachInterrupt(CALIBRATION_BUTTON_PIN, calibrationButtonISR, RISING);

  //initialize LCD
  lcd.init();
  //turn on LCD Backlight
  lcd.backlight();

  // Configure limit switches with internal pull-up resistors
  pinMode(X_END_STOP_PIN, INPUT_PULLUP);
  pinMode(Y_END_STOP_PIN, INPUT_PULLUP);

  // Attach external interrupts for each limit switch
  attachInterrupt(X_END_STOP_PIN, xAxisStopISR, RISING);
  attachInterrupt(Y_END_STOP_PIN, yAxisStopISR, RISING);

  // Configure motor pins as outputs
  ledcAttach(X_AXIS_MOTOR_PIN_1, PWM_FREQUENCY, PWM_RESOLUTION);
  ledcAttach(X_AXIS_MOTOR_PIN_2, PWM_FREQUENCY, PWM_RESOLUTION);
  ledcAttach(Y_AXIS_MOTOR_PIN_1, PWM_FREQUENCY, PWM_RESOLUTION);
  ledcAttach(Y_AXIS_MOTOR_PIN_2, PWM_FREQUENCY, PWM_RESOLUTION);

  ledcWrite(X_AXIS_MOTOR_PIN_1, LOW);
  ledcWrite(X_AXIS_MOTOR_PIN_2, LOW);
  ledcWrite(Y_AXIS_MOTOR_PIN_1, LOW);
  ledcWrite(Y_AXIS_MOTOR_PIN_2, LOW);

  // Configure motor encoders
  ESP32Encoder::useInternalWeakPullResistors = puType::up;  // Enable the weak pull up resistors
  X_ENCODER.attachHalfQuad(X_AXIS_ENCODER_PIN_1, X_AXIS_ENCODER_PIN_2);                            // Attache pins
for use as encoder pins
  X_ENCODER.setCount(0);                                      // set starting count value after attaching
  Y_ENCODER.attachHalfQuad(Y_AXIS_ENCODER_PIN_1, Y_AXIS_ENCODER_PIN_2);                            // Attache pins
for use as encoder pins
  Y_ENCODER.setCount(0);                                      // set starting count value after attaching

  //Set up IR sensor
  pinMode(IR_SENSOR_PIN, INPUT);
  attachInterrupt(IR_SENSOR_PIN, pointTrigger, RISING);

  //Calibration Timer
  // SetupCalibrationTimer();

  //Debounce Timer
  SetupDebounceTimer();

  SetupCalibrationTimer();

  // SetupPITimer();
  // timer1 = timerBegin(1000000);            // Set timer frequency to 1Mhz
  // timerAttachInterrupt(timer1, &onTime1);  // Attach onTimer1 function to our timer.
  // timerAlarm(timer1, 10000, true, 0);      // 10000 * 1 us = 10 ms, autoreload true

  //Pos potentiometer
  pinMode(X_POT, INPUT);
  pinMode(Y_POT, INPUT);
  Serial.println("Started in Idle state. Press button to start calibration.");
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Welcome");
  lcd.setCursor(0,1);
  lcd.print("Idle state");
}

void SetupDebounceTimer(){
  timer0 = timerBegin(1000000);
  timerAttachInterrupt(timer0, &debounceTimerCallback);
  timerAlarm(timer0, 1500000, true, 0); //1.5 second timer
  timerStop(timer0);
}

void SetupCalibrationTimer(){
  // Initialize hardware timer for calibration
  timer2 = timerBegin(TIMER_FREQUENCY);  // Set timer frequency
  timerAttachInterrupt(timer2, &timeoutISR);
  timerAlarm(timer2, TIMEOUT_DURATION * 1000, false, 0); // Alarm in microseconds, no auto-reload
  timerStop(timer2);
}

// void SetupGamePlayTimer(){
//   //Gameplay Timer
//   timer1 = timerBegin(TIMER_FREQUENCY);
//   timerAttachInterrupt(timer1, &onTime0);
//   timerAlarm(timer1, 10 * 1000 * 1000, false, 0); // 1 Minute timer
//   timerStop(timer1);
// }

void SetupPITimer(){
  timer1 = timerBegin(1000000);            // Set timer frequency to 1Mhz
  timerAttachInterrupt(timer1, &onTime1);  // Attach onTimer1 function to our timer.
  timerAlarm(timer1, 10000, true, 0);      // 10000 * 1 us = 10 ms, autoreload true
  //timerStop(timer3);
}

void loop() {
  // delay(100);
  // Main state machine
  // Serial.println(currentState);
  // Serial.println(buttonPressed);
  lcd.setCursor(0,0);
  switch (currentState) {
    case IDLE:
        // Serial.println("In idle state");
        // Serial.println(buttonPressedChecker());
        if (buttonPressedChecker() == true) {
          startXAxisCalibrationService();
          currentState = X_AXIS_CALIBRATING;
        }
        break;

    case X_AXIS_CALIBRATING:
        // Check if the X-axis limit switch was hit and print message
        if (xAxisHitChecker() == true) {
          startYAxisCalibrationService();
          currentState = Y_AXIS_CALIBRATING;
        }
        if (CheckTimeout()) {
          calibration_timeoutService();
          currentState = IDLE;
        }
        break;

    case Y_AXIS_CALIBRATING:
        // Check if the X-axis limit switch was hit and print message
        if (yAxisHitChecker() == true) {
          endCalibrationService();
          currentState = READY_TO_PLAY;
        }
        if (CheckTimeout()) {
          calibration_timeoutService();
          currentState = IDLE;
        }
        break;

    case READY_TO_PLAY:
        if (buttonPressedChecker() == true) {
          StartGame(); //write function to start game
          currentState = GAME_ON;
        }
        break;
    case GAME_ON:
        //stationary
        if (CheckForMadeHoop()) {
          IncreaseScore();
          currentState = GAME_ON;
        }
        if (CheckEncoder()){
          RunPositionControl_X();
          RunPositionControl_Y();
          // currentState = GAME_ON;
        }
        if (CheckGameTimer()){
          GameOver();
          currentState = GAME_OVER;
        }

        break;
        //within each level case also check for timer to run out

    case GAME_OVER:
        if (buttonPressedChecker() == true) {
            ReStart();
            currentState = IDLE;
            // ReStart();
            // currentState = READY_TO_PLAY;
          }
        break;
  }
}

void RunPositionControl_X(){
  // count = encoder.getCount();
  // encoder.clearCount();

  theta += count;
  X_POTReading = analogRead(X_POT);
  thetaDes = map(X_POTReading, 0, 4095, 0, thetaMax);

  //CONTROL SECTION
  error = thetaDes - theta;
  sum += (error/10);

  // anti-windup section
  if (sum > 15){
    sum = 15;
  }
  else if (sum < -15){
    sum = -15;
  }

  D = Kp * error + Ki * sum;

  // Serial.print("D: ");
  // Serial.print(D);
  // Serial.print(" ");

  //Ensure that you don't go past the maximum possible command
  if (D > MAX_PWM_VOLTAGE) {
    D = MAX_PWM_VOLTAGE;
  } else if (D < -MAX_PWM_VOLTAGE) {
    D = -MAX_PWM_VOLTAGE;
  }
  //Map the D value to motor directionality

  if (D > 0) {
    ledcWrite(X_AXIS_MOTOR_PIN_1, 0);
    ledcWrite(X_AXIS_MOTOR_PIN_2, D);
  } else if (D < 0) {
    ledcWrite(X_AXIS_MOTOR_PIN_2, 0);
    ledcWrite(X_AXIS_MOTOR_PIN_1, -D);
  } else {
    ledcWrite(X_AXIS_MOTOR_PIN_2, 0);
    ledcWrite(X_AXIS_MOTOR_PIN_1, 0);
  }

  //plotControlData_X();

}

void plotControlData_X() {
  Serial.print("Position:");
  Serial.print(theta);
  Serial.print(" ");
  Serial.print("Desired_Position:");
  Serial.print(thetaDes);
  Serial.print(" ");
  Serial.print("PWM_Duty:");
  Serial.println(D);
}

void RunPositionControl_Y(){
  // count = encoder.getCount();
  // encoder.clearCount();

  theta_y += count_y;
  Y_POTReading = analogRead(Y_POT);
  thetaDes_y = map(Y_POTReading, 0, 4095, 0, thetaMax_y);

  //CONTROL SECTION
  error_y = thetaDes_y - theta_y;
  sum_y += (error_y/10);

  // anti-windup section
  if (sum_y > 50){
    sum_y = 50;
  }
  else if (sum_y < -50){
    sum_y = -50;
  }

  D_y = Kp_y * error_y + Ki_y * sum_y;

  // Serial.print("D: ");
  // Serial.print(D);
  // Serial.print(" ");

  //Ensure that you don't go past the maximum possible command
  if (D_y > MAX_PWM_VOLTAGE) {
    D_y = MAX_PWM_VOLTAGE;
  } else if (D_y < -MAX_PWM_VOLTAGE) {
    D_y = -MAX_PWM_VOLTAGE;
  }
  //Map the D value to motor directionality

  if (D_y > 0) {
    ledcWrite(Y_AXIS_MOTOR_PIN_1, 0);
    ledcWrite(Y_AXIS_MOTOR_PIN_2, D_y);
  } else if (D_y < 0) {
    ledcWrite(Y_AXIS_MOTOR_PIN_2, 0);
    ledcWrite(Y_AXIS_MOTOR_PIN_1, -D_y);
  } else {
    ledcWrite(Y_AXIS_MOTOR_PIN_2, 0);
    ledcWrite(Y_AXIS_MOTOR_PIN_1, 0);
  }

  plotControlData_Y();

}

void plotControlData_Y() {
  Serial.print("Position y:");
  Serial.print(theta_y);
  Serial.print(" ");
  Serial.print("Desired_Position y:");
  Serial.print(thetaDes_y);
  Serial.print(" ");
  Serial.print("PWM_Duty y:");
  Serial.println(D_y);
}

bool CheckTimeout(){
    if (timeoutOccurred){
        timeoutOccurred = false;
        return true;
    }
    return false;
}

void driveMotor(int M1, int M2, int direction = 0, int pos = -1){
  /*
  M1, M2 - motor pins
  direction - 0, -1, 1
  pos - default will keep running, else int for encoder count
  */
  int pwm_value = 0;

  if (pos==-1) {
    // no control, just run
    pwm_value = MAX_PWM_VOLTAGE;
  }
  else {
    // need to implement position control
    pwm_value = 0;
  }

  if (direction == 0) {
    ledcWrite(M1, LOW);
    ledcWrite(M2, LOW);
  }
  else if (direction == -1) {
    ledcWrite(M1, LOW);
    ledcWrite(M2, pwm_value);
  }
  else if (direction == 1) {
    ledcWrite(M1, pwm_value);
    ledcWrite(M2, LOW);
  }
}

void driveXAxisMotor(int direction = 0, int pos = -1){
  driveMotor(X_AXIS_MOTOR_PIN_1, X_AXIS_MOTOR_PIN_2, direction, pos);
  if (direction == 0){
    Serial.println("X-axis motor stopped");
  }
  else{
    Serial.println("X-axis motor running");
  }
}

void driveYAxisMotor(int direction = 0, int pos = -1){
  driveMotor(Y_AXIS_MOTOR_PIN_1, Y_AXIS_MOTOR_PIN_2, direction, pos);
  if (direction == 0){
    Serial.println("Y-axis motor stopped");
  }
  else{
    Serial.println("Y-axis motor running");
  }
}

bool buttonPressedChecker() {
  if (buttonPressed && !debouncing) {
    buttonPressed = false;
    debouncing = true;
    timerStart(timer0);
    return true;
  }
  return false;
}

bool xAxisHitChecker() {
  if (xAxisHit==true) {
    xAxisHit = false;
    return true;
  }
  return false;
}

bool yAxisHitChecker() {
  if (yAxisHit==true) {
    yAxisHit = false;
    return true;
  }
  return false;
}


//Event Service
void calibration_timeoutService() {
  driveXAxisMotor(0); // Stop X-axis motor if running
  driveYAxisMotor(0); // Stop Y-axis motor if running

  Serial.println("Calibration Error: Timeout occurred");
  Serial.println("Going back to Idle. Press button to try calibration again.");
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Calibration");
  lcd.setCursor(0,1);
  lcd.print("timeout");
  timerStop(timer2);
  timerRestart(timer2);
  timeoutOccurred = false;
}

//Event Service
void startXAxisCalibrationService() {
  Serial.println("X-axis calibration started");
  lcd.clear();
  lcd.print("Calibration");
  lcd.setCursor(0,1);
  lcd.print("started");
  driveXAxisMotor(1); //go forward with no position control
  timerStart(timer2);       // Start timeout timer
}

void startYAxisCalibrationService() {
  Serial.println("Y-axis calibration started");
  lcd.clear();
  lcd.print("Y-axis");
  lcd.setCursor(0,1);
  lcd.print("Calibration");
  driveYAxisMotor(1); //go forward with no position control
  timerRestart(timer2);       // Start timeout timer
}
```

```
//Event Service
void endCalibrationService(){
  driveXAxisMotor(0); //stop x
  driveYAxisMotor(0); //stop y

  //set 0 encoder positions
  X_ENCODER.setCount(0);
  Y_ENCODER.setCount(0);

  timerStop(timer2);  // Stop timer as calibration is complete
  timerEnd(timer2);

  //driveXAxisMotor(-1);
 // driveYAxisMotor(-1);
  //delay(300);
  //driveXAxisMotor(0); //stop x
  //driveYAxisMotor(0); //stop y

  Serial.println("Y end stop hit, Calibration complete!");
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Calibration");
  lcd.setCursor(0,1);
  lcd.print("complete!");
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Press play");
  buttonPressed = false;
}

//Event Checker
bool CheckForMadeHoop() {
  if (madeHoop && !debouncing){
    debouncing = true;
    madeHoop = false;
    timerStart(timer0);
    return true;
  }
  return false;
}

bool CheckGameTimer(){
  // Serial.println(game_over_counter);
  if (game_over_counter > 3000){
    portENTER_CRITICAL(&timerMux1);
    game_over_counter = 0;
    portEXIT_CRITICAL(&timerMux1);
    return true;
  }
  return false;
}

bool CheckEncoder(){
  if (deltaT){
    portENTER_CRITICAL(&timerMux1);
    deltaT = false;
    portEXIT_CRITICAL(&timerMux1);
    // Serial.println("PI timer");
    return true;
  }
  return false;
}

void StartGame(){
  // madeHoop = false;
  Serial.println("Game started");
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Game started");
  SetupPITimer();
  // SetupGamePlayTimer();
  // timerStart(timer1);
  // timerStart(timer3);
}

//Event Service
void GameOver() {
  driveXAxisMotor(0);
  driveYAxisMotor(0);

  Serial.print("Game Over! Your score is: ");
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Game Over!");
  lcd.setCursor(0,1);
  lcd.print("Your Score:");
  lcd.print(score);
  Serial.println(score);

  timerStop(timer1);
  timerEnd(timer1);
  buttonPressed = false;
}


void IncreaseScore(){
  score += 1;
  Serial.println("");
  Serial.print("Score: ");
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Score:");
  lcd.setCursor(0,1);
  lcd.print(score);
  Serial.println(score);
}

void ReStart(){
  Serial.println("Back to idle");
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Idle");
  score = 0;
  SetupCalibrationTimer();
  // count = 0;
  theta = 0;
  deltaT = 0;
  game_over_counter =0;
  // game_over_timer = 0;
}
```