Billiard Robot

ME 102B Fall 2021
Group 12: Yuting(Elizabeth) Chen, Haoxiang Huang, WenHao Li

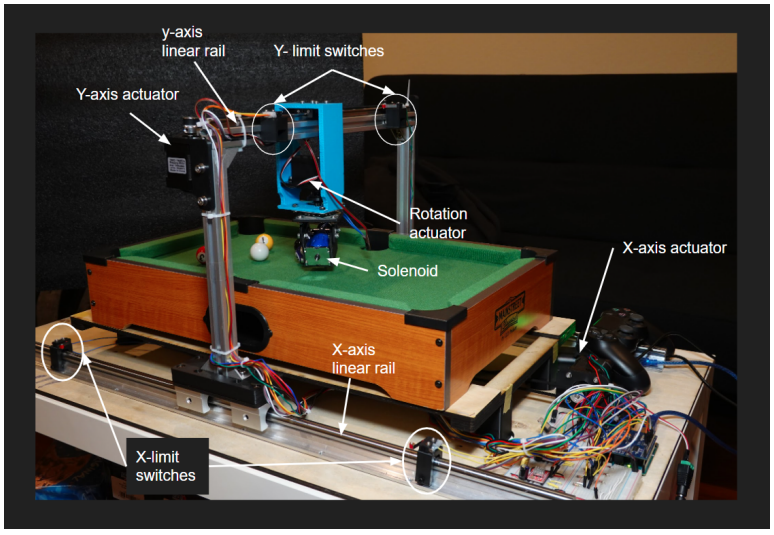
Opportunity

We initially wanted to build a robot that can play billiards autonomously with people who need an opponent. The final design of our billiard robot simplifies to a billiard-playing robot controlled by a player through a PS4 controller. With the hardware and mechanism completed, it is possible to implement computer vision to the robot in the future for it to be an autonomous robot opponent.

Strategy

The main functionality we wanted to achieve was to move the tip of the cue stick accurately above the pool table. Our initial idea was to have the robot travel on a track circulating the pool table. However, we faced two main challenges with this idea: turning around on the corners and stabilizing when hitting a ball. Inspired by 3D printers, we designed the device based on linear rails and stepper motors. We also attached a servo motor to a solenoid to adjust the hitting direction. To reduce the work due to time constraints, we removed the height adjustment, which could be added back as a future improvement. Since the robot movement would be controlled by a player, we used the PS4 controller as the control interface. The buttons and joysticks on the controller were the sensors for user inputs. We also took advantage of the led and vibration motors on the controller for visual and tactile feedback on different states.

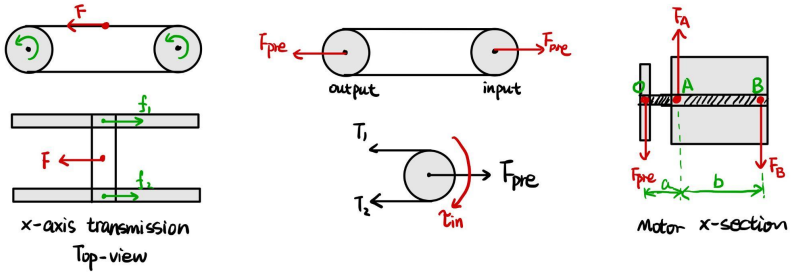
Photo of the Physical Device



Function-Critical Decisions

X and Y axis actuators: The calculation shown below indicates that to move the mechanism, the actuator is required at least 0.102 Nm holding torque. The two Twotrees Nema 17 stepper motors on both axes have holding torque that meets about a safety factor of 6 (based on the assumption). The loading force on the y-axis (approx: 600g) is smaller than the

loading force on the x-axis (approx:2kg). Based on the calculations, under 2 times of our designed operation torque, the built-in bearing is applied about 35.84N, which is within the safety range of allowable loading from the spec of the motor. For future improvements, we could add a support to reduce the loading force on the motor built-in bearing to increase the motor life.



Motor torque calculation:

$$assume: no\ slip,\ max\ coefficient\ of\ friction\ (steel\ \&\ steel,\ clean/dry)\ =\ 0.8,$$
$$loading\ force\ =\ 2kg,\ pulley\ diameter\ =\ 13mm,\ a\ =\ 5mm,\ b\ =\ 35mm$$
$$f_1 + f_2 = \mu_{max} mg = 0.8 \times 2 \times 9.8 = 15.68\ N$$
$$\tau = F \cdot R = 15.68N \times 0.0065m = 0.102Nm$$
$$motor\ holding\ Torque\ (\tau_{max}) = 0.59Nm \qquad Safety\ Factor\ = \frac{0.59}{0.102} = 5.78$$

Motor shaft preload / bearing load calculation:

$$T_{2-min} = T_i - \frac{\tau_{in}}{d_{in}} = 0 \qquad T_i = \frac{\tau_{in}}{d_{in}} = \frac{0.102Nm}{0.013m} = 15.69N\ (under\ 2\tau_{operation})$$
$$\phi = \pi \qquad T_{pre} = 2T_i = 31.38N$$
$$solve\ equations:\ F_A - F_L - F_B = 0 \quad and \quad M_{F_L/A} + M_{F_B/A} = 0 \qquad F_A = 35.84N$$

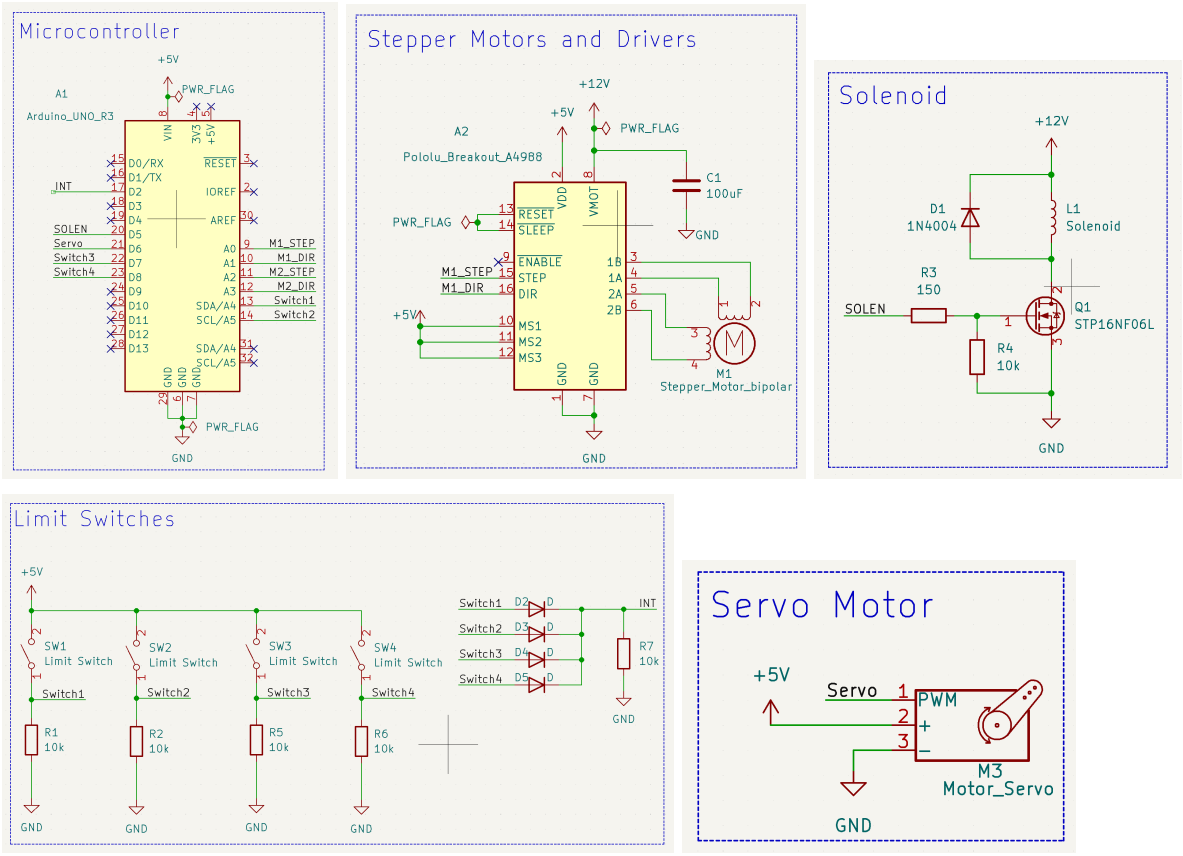
Rotational motion actuator-servo motor: Based on the assumption and calculation shown below, the servo motor can handle more inertia torque than we possibly need.

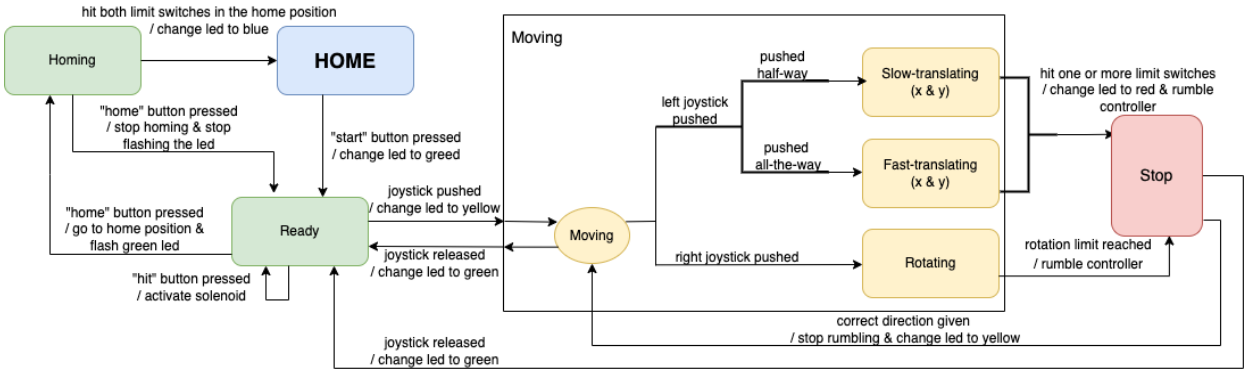
$$assume:\ extreme\ case\ of\ moment\ of\ inertia:\ "Hoop\ about\ cylinder\ axis"\ I = MR^2,\ M = 76.13N$$
$$I = MR^2 = 76.13g \times R_{max}^2 = 76.13g \times (25\sqrt{2})^2 mm = 9.52 \times 10^{-5} kgm^2$$
$$\tau_{max} = \frac{\tau_{max}}{I_{max}} = \frac{0.54Nm}{9.52\times10^{-5} kgm^2} = 5.67 \times 10^3 rad/s^2$$

Solenoid selection: Based on the calculation below, the force generated by the solenoid is able to give a certain speed to the cue ball. However, there is always a loss of energy from friction and other imperfections. Therefore, we pick a larger solenoid instead of a smaller one which also fits our case in theory.

$$assume: Ball - table\ coefficient\ of\ friction\ = 0.5;\ M_{ball} = 15g;\ Solenoid\ Max\ suction\ force\ = 45N$$
$$Max\ \#of\ ball\ = 16,\ the\ tip\ of\ the\ solenoid\ is\ placed\ at\ 3mm\ away\ from\ balls;\ hitting\ time\ = 0.1s$$
$$f_{16balls} = \mu mg = 0.0735 \times 16 = 1.176N;$$
$$based\ on\ the\ graph:\ Typical\ Force\ vs.\ Strock,\ force\ provide\ = 15.27N \gg f_{16balls}$$
$$\Delta E = \frac{1}{2} m(v_f^2 - v_i^2) = F\Delta t \qquad v_f = 3.567m/s$$

Circuit Diagram and State Transition Diagram





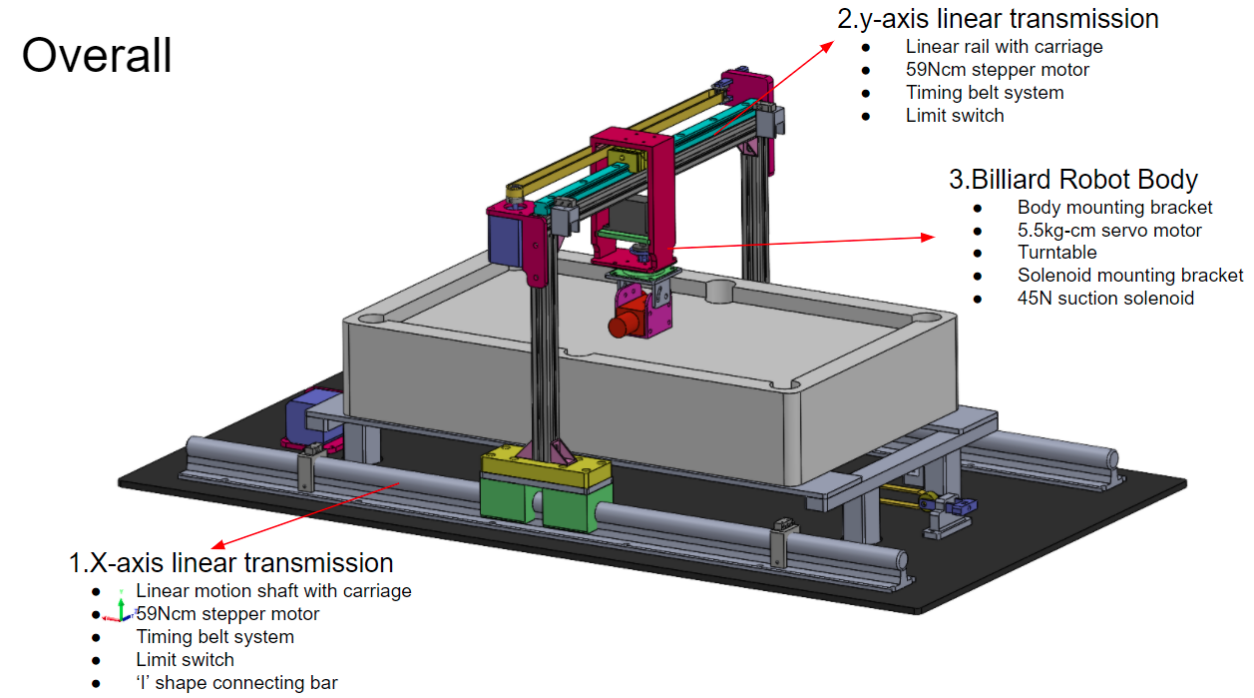
Reflection

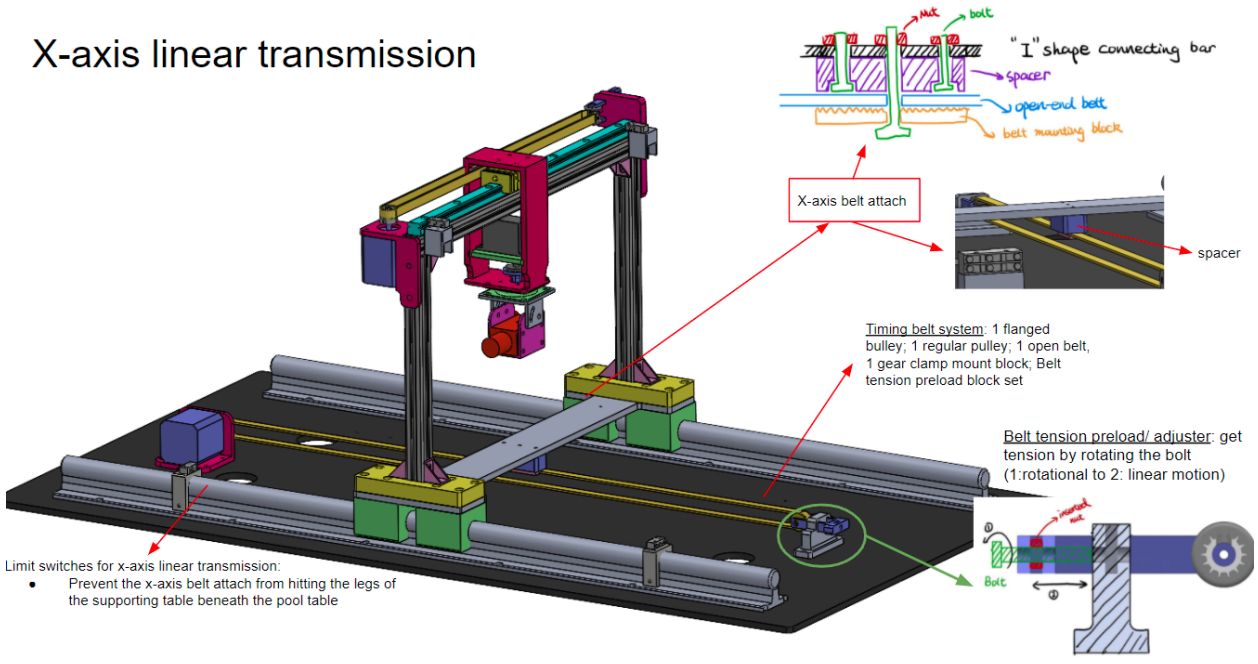
In order to be successful in ME 102B, students as a group should plan and follow a rigorous timeline for the project. Start early and do not procrastinate, and expect to encounter a lot of new design problems occurring in the process. Group members should monitor each other and give feedback on problems in time. Do not be afraid to communicate with anyone. Do not hesitate to ask for suggestions from the instructor and the GSIs if necessary.

Our billiard robot works in a similar way to a 3D printer, but without the height adjustment feature because of the time constraints. The reason for this is that our initial design is too complicated to finish in the time of only one semester. Therefore, for future class projects, we should start with the simplest aspect, and then develop the design based on the timeline.

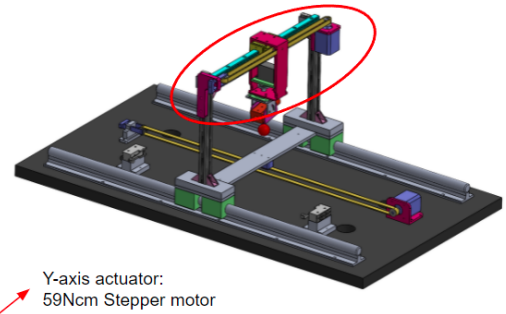
Complete Bill of Materials
https://docs.google.com/spreadsheets/d/1Sfi7-Sd-vbXvyyZtsYCpaGJIsU2DrbG6bK7b_SALeCk/edit?usp=sharing

CAD Design

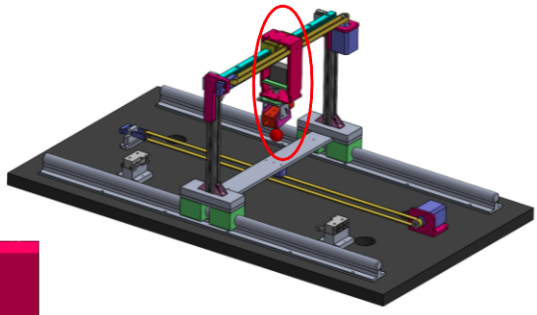


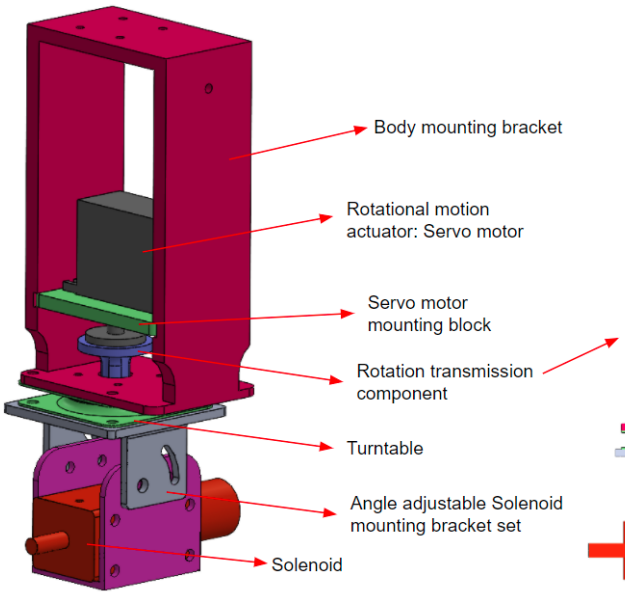


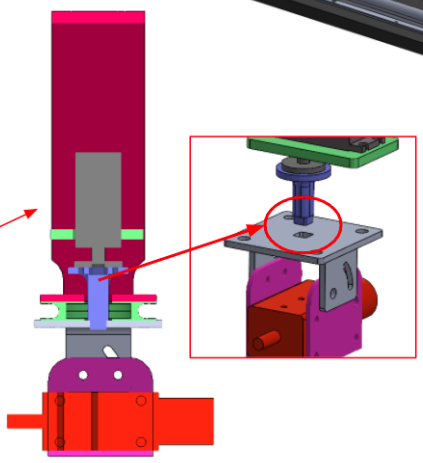
Y-axis linear transmission



Billiard Robot Body







Notes:
In order to eliminate/reduce the down force acting on the servo motor shaft, we add a rotation transmission component which is connected on the bottom part of the turntable. It transfer the load on the turntable.
*turntable can handle about 44 lb force from product information sheet. (reference link could be found in BOM)

Code

```
#include <PS4BT.h>
#include <PS4USB.h>
#include <usbhub.h>
#include <AccelStepper.h>
#include <Servo.h>

#ifdef dobogusinclude
#include <spi4teensy3.h>
#endif
#include <SPI.h>


USB Usb;

//// Uncomment to enable bluetooth connection
//BTD Btd(&Usb);
//PS4BT PS4(&Btd, PAIR);
//PS4BT PS4(&Btd);

// Comment this line for bluetooth connection
PS4USB PS4(&Usb);

/*********************************/
/*  Pin definitions              */
/*********************************/
#define solenPin 5 //solenoid
#define servoPin 6  //servo
#define stepPinX A0 //stepper
#define dirPinX A1
#define stepPinY A2
#define dirPinY A3
#define LSW_X1 A4 //limit switch
#define LSW_X2 A5
#define LSW_Y1 7
#define LSW_Y2 8
#define INT_PIN 2 //interrupt

/*********************************/
/*   State variables             */
/*********************************/
int sysState = 0;
bool isHome = true;

/*********************************/
/*   Variables for the user input   */
/*********************************/
#define up_ths 157
#define low_ths 97
int joyLeftX;
int joyLeftY;
int joyRightX;
int joyRightY;
int stepX = 0;
int dirX = 0;
int stepY = 0;
int dirY = 0;

/*****************************************/
/*  Motor instances and variables  */
/*****************************************/
#define low_speed 2000
#define high_speed 6400
#define max_servo_dist 3600
AccelStepper stepperX = AccelStepper(1, stepPinX, dirPinX);
AccelStepper stepperY = AccelStepper(1, stepPinY, dirPinY);
int speedX = 0;
int speedY = 0;
Servo servo;
int speedServo = 0;
int servoDist = 0;

```

```
/*********************************/
/*  Limit switches and interrupt  */
/*********************************/
volatile bool anySwitch = false;

void setup() {
  initActuators();
  initLimitSWs();

  Serial.begin(115200);
#if !defined(__MIPSEL__)
  while (!Serial); // Wait for serial port to connect
#endif
  if (Usb.Init() == -1) {
    Serial.print(F("\r\nOSC did not start"));
    while (1); // Halt
  }
  Serial.println(F("\r\nPS4 pairing..."));
}

void initActuators() {
  pinMode(solenPin, OUTPUT);
  pinMode(servoPin, OUTPUT);
  stepperX.setMaxSpeed(high_speed);
  stepperX.setAcceleration(50);
  stepperY.setMaxSpeed(high_speed);
  stepperY.setAcceleration(50);

  stepperX.setSpeed(0);
  stepperX.runSpeed();
  stepperY.setSpeed(0);
  stepperY.runSpeed();
}

void initLimitSWs() {
  pinMode(LSW_X1, INPUT);
  pinMode(LSW_X2, INPUT);
  pinMode(LSW_Y1, INPUT);
  pinMode(LSW_Y2, INPUT);

  pinMode(INT_PIN, INPUT);
  attachInterrupt(digitalPinToInterrupt(INT_PIN), switchPressed, RISING);
}

void switchPressed() {
  //Serial.println("switch reached");
  anySwitch = true;
}

void loop() {
  Usb.Task();

  if (PS4.connected()) {

    /***********************************
      System state definitions:
        0: Home state
        1: Stationary
        2: Moving (linear & rotational)
          2.0: StopX
          2.1: StopY
        3: Homing
    ***********************************/
    switch (sysState) {
      case 0: //Home state
        PS4.setLed(Blue);
        if (checkStart()) {
          sysState = 1; //If started, change to stationary state
          start_sys_res();
          //Serial.println("started");
```

```
    }
    break;

  case 1: //Stationary state
    if (checkLeftJoyX() || checkLeftJoyY() || checkRightJoyX()) {
      sysState = 2;
      PS4.setLed(Yellow);
      //Serial.println("Moving");
    }
    if (checkHit()) {
      hit_res();
      //Serial.println("hit!");
    }
    if (checkHome()) {
      servoDist = 0;
      sysState = 3;
      //Serial.println("Homing...");
    }
    break;

  case 2: //Moving

    //move xy axis
    if (checkLeftJoyX() && checkEdgeClearX()) {
      drive_stepperX_res(joyLeftX);
      //Serial.print("Left X: ");
      //Serial.println(stepperX.speed());
    }

    if (checkLeftJoyY() && checkEdgeClearY()) {
      drive_stepperY_res(joyLeftY);
      //Serial.print("Left Y: ");
      //Serial.println(stepperY.speed());
    }

    if (digitalRead(INT_PIN) == LOW) {
      anySwitch = false;
      PS4.setLed(Yellow);
      PS4.setRumbleOff();
    }
    if (anySwitch) {
      PS4.setLed(Red);
      PS4.setRumbleOn(100, 100);
    }

    //rotate cue stick
    if (checkRightJoyX()) {
      servo.attach(servoPin);
      rotate_servo_res();
    } else {
      servo.detach();
    }

    if (!checkLeftJoyX() && !checkLeftJoyY() && !checkRightJoyX()) {
      sysState = 1;
      PS4.setLed(Green);
      PS4.setRumbleOff();
    }

    break;

  case 3: //Homing

    if (checkHomeCompleteX() && checkHomeCompleteY()) {
      //Serial.println("Homed!");
      PS4.setLedFlash(0, 0);
      sysState = 0;
    } else {
      PS4.setLedFlash(50, 50);
      if (!checkHomeCompleteX())
        homeX_res();
      if (!checkHomeCompleteY())
```

```
            homeY_res();
        }
        if (PS4.getButtonClick(TOUCHPAD)) {
          PS4.setLedFlash(0, 0);
          sysState = 2;
        }
        break;
    }
  }
}



/***********************************/
/*   Event Checker Functions       */
/***********************************/

//If the system is in home state,  chech if a "start" command is sent
bool checkStart() {
  return PS4.getButtonClick(CIRCLE);
}

// Check if the hit command is sent to turn on the solenoid
bool checkHit() {
  return PS4.getButtonClick(CROSS);
}

bool checkHome() {
  return PS4.getButtonClick(TOUCHPAD);
}

bool checkLeftJoyX() {
  joyLeftX = PS4.getAnalogHat(LeftHatX);
  return (joyLeftX > up_ths) || (joyLeftX < low_ths);
}


bool checkLeftJoyY() {
  joyLeftY = PS4.getAnalogHat(LeftHatY);
  return (joyLeftY > up_ths) || (joyLeftY < low_ths);
}

bool checkEdgeClearX() {
  if (anySwitch) {
    if (digitalRead(LSW_X1) == HIGH) //negative direction of x
      return joyLeftX < low_ths; //the command must go to the positve direction
    else if (digitalRead(LSW_X2) == HIGH)
      return joyLeftX > up_ths;
  }
  return true;
}

bool checkEdgeClearY() {
  if (anySwitch) {
    if (digitalRead(LSW_Y1) == HIGH) //negative direction of Y
      return joyLeftY < low_ths; //the command must go to the positve direction
    else if (digitalRead(LSW_Y2) == HIGH)
      return joyLeftY > up_ths;
  }
  return true;
}

bool checkRightJoyX() {
  joyRightX = PS4.getAnalogHat(RightHatX);
  return (joyRightX > up_ths) || (joyRightX < low_ths);
}

bool checkHomeCompleteX() {
  return digitalRead(LSW_X1) == HIGH;
}

```

```
bool checkHomeCompleteY() {
  return digitalRead(LSW_Y1) == HIGH;
}


/***********************************/
/*   Service Response Functions     */
/***********************************/
void start_sys_res() {
  PS4.setLed(Green);
}

void hit_res() {
  digitalWrite(solenPin, HIGH);
  delay(100);
  digitalWrite(solenPin, LOW);
}

void drive_stepperX_res(int joyLeftX) {
  if (joyLeftX > up_ths && joyLeftX < 201) {
    speedX = low_speed;
  } else if (joyLeftX >= 201) {
    speedX = high_speed;
  }else if (joyLeftX < low_ths && joyLeftX > 54) {
    speedX = -low_speed;
  }else if (joyLeftX <= 54) {
    speedX = -high_speed;
  }
  if (speedX != int(stepperX.speed())) {
    //Serial.println("new speed x set");
    stepperX.setSpeed(speedX);
  }

  stepperX.runSpeed();
}

void drive_stepperY_res(int joyLeftY) {
  if (joyLeftY > up_ths && joyLeftY < 201) {
    speedY = -low_speed;
  } else if (joyLeftY >= 201) {
    speedY = -high_speed;
  }else if (joyLeftY < low_ths && joyLeftY > 54) {
    speedY = low_speed;
  }else if (joyLeftY <= 54) {
    speedY = high_speed;
  }
  if (speedY != int(stepperY.speed())) {
    //Serial.println("new speed y set");
    stepperY.setSpeed(speedY);
  }

  stepperY.runSpeed();
}


void rotate_servo_res() {
  if (joyRightX > up_ths) {
    speedServo = 100;
    servoDist += 10;
  } else if(joyRightX < low_ths) {
    speedServo = 82;
    servoDist -= 10;
  }
  if (servoDist < max_servo_dist && servoDist > -max_servo_dist) {
    servo.attach(servoPin);
    PS4.setRumbleOff();
    servo.write(speedServo);
  } else {
    servo.detach();
    PS4.setRumbleOn(100, 100);
    if (servoDist >= max_servo_dist) {
      servoDist = max_servo_dist;
    } else if (servoDist <= -max_servo_dist) {
      servoDist = -max_servo_dist;
    }
  }
  delay(15);
}

void homeX_res() {
  stepperX.setSpeed(high_speed);
  stepperX.runSpeed();
}

void homeY_res() {
  stepperY.setSpeed(-high_speed);
  stepperY.runSpeed();
}
```