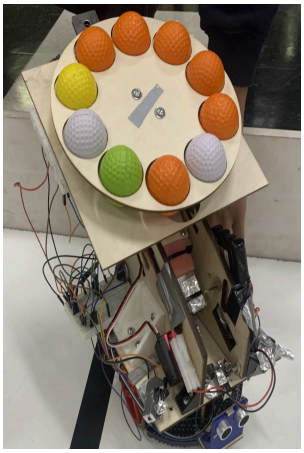
Advanced Mechatronics Final Report

Ruler-Catapult Basketball Robot

Yohannes Bekele
Juan Chavarria
Ali Mohamed
Russell Phillips

4/28/2023

ME 348E Advanced Mechatronics
Mechanical Engineering Department, Cockrell School of Engineering
The University of Texas at Austin



Introduction

We started off this project by settling our goals and objectives for the competition. We initially analyzed the rules of the game as well as the game field, and we discussed what aspects of the game we should focus on to perform well in the competition. We further broke down the game into four sections: shooting, loading, localization, and motion/movement. IR beacon detection is an integral part of the game, specifically because it requires the robot to have accurate positioning to correctly detect the correct basket to shoot at. Other indicators, such as wall distances and tape intersections can enhance localization precision with the use of line-following and ultrasonic sensors. With this in mind, we made a morphological chart to draft potential solutions for localization and motion, and also generated early shooting and reloading concepts.

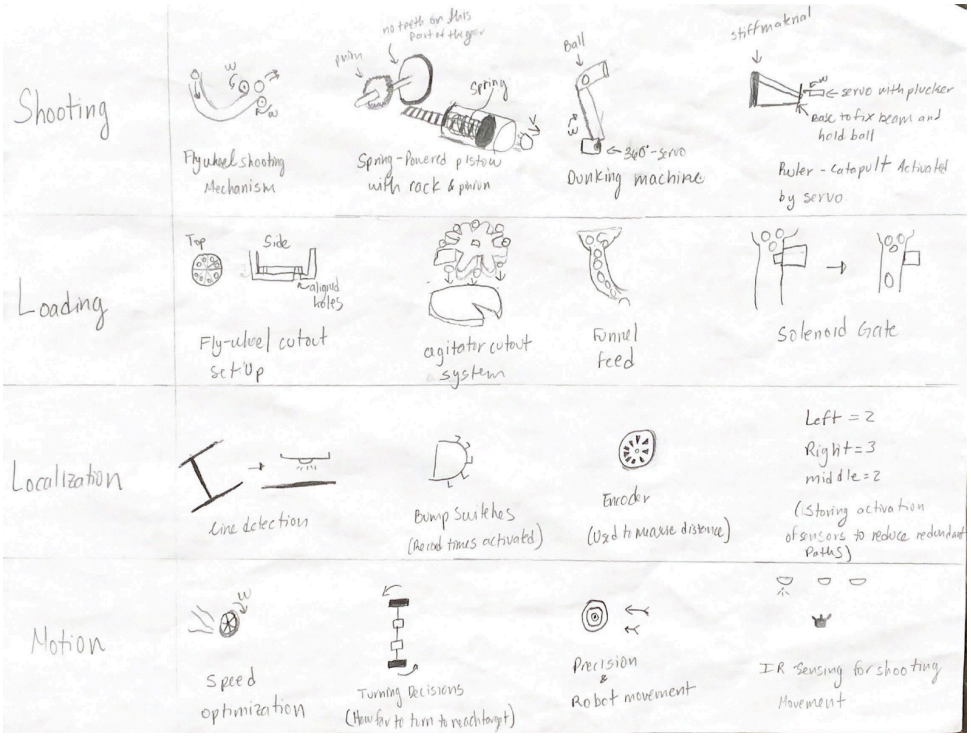


Figure 1. Morphological Chart for the Robot

We decided to have a fixed shooting position during the game to minimize movement and mainly focus on shooting. This, as previously stated, would require precise position with the use of line-sensors, encoders, and an ultrasonic sensor, which would be universal for any designs. In regards to shooting, we drafted three ideas: a linear spring shooting mechanism, a three arm catapult shooting system, and a classical flywheel shooting mechanism. These systems would be designed to shoot from the top-most intersection, so they'd have to be powered and designed to meet this distance. We further evaluated each design by ranking them based on the criteria shown in Table 1 below. More information on how each criteria was evaluated is shown in Appendix A.

	Linear Spring	3 Arm	Flywheel
Ranking			
Ease of Assembly	10	8	6
Practicality	7	12	5
Reliability	8	11	5
Time-Efficiency	6	11	7
Total	**31**	**42**	**23**

Table 1. Ranking of Designs

The clear winner was the three arm catapult idea, mainly because it was the most practical. One of the drastic changes was that having three arms pushed our dimensions to the limits, and we weren't sure if complicating a reloading mechanism to efficiently load balls to three shooters would be worth the trouble. We moved on with a one shooter design with a simple flywheel reloader, which will be referenced in further sections. The early design sketches and early shooter prototype can also be found in Appendix A, and the final prototype in Appendix B.

Description of Operation

After establishing the game plan, we decided to move forward to the prototyping and testing of our system. In this section, we will touch on the reloading and shooting mechanisms of

the robot, the control logic of the robot, and the full electronic and mechanical assembly of the robot.

The mechanical design of the robot consisted of the RSLK chassis, standoffs and a flat surface to mount the shooting and reloading, and the electronics. As far as the design of the robot, we implemented the linear spring shooter with a reloading flywheel on top to minimize design complexity. The linear spring, which is a ruler, is flicked by a high-torque servo and essentially catapults the ball into the desired location. Figure 2 below shows the final CAD assembly of the robot. More images of the robot can be found in Appendix B, which shows the front view of the real system, where the ruler is bent by the high torque servo.

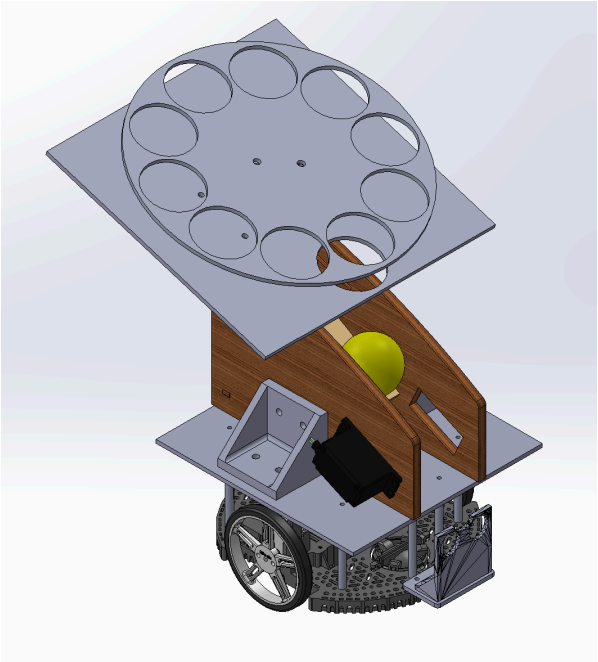


Figure 2. Final Robot Assembly

The software system architecture employs the following operations perception, data analysis and decision-making algorithms, and actuation. The system first gathers information from the ultrasonic sensor and motor encoders. These data are then analyzed using a perception and localization algorithm to extract useful information, such as what side the robot is on, how

far away the robot is from the back wall, and where the centerline is. The system then uses the onboard infrared array sensors for line detection and following. The system uses a feedback loop, specifically PD control, where infrared array sensor data feeds into the line following controller, which in turn drives the motors. This loop continues until the system has the desired shooting position. Based on the information gathered from the IR receiving sensors, the system then uses a decision-making algorithm to determine which basket to shoot at. For example, if the middle IR receiver receives the most signals, the decision-making algorithm will instruct the system to align itself with the middle basket, reload, and shoot. Finally, the system acts based on the decisions made by the decision-making algorithms. This may involve activating wheel motors, actuating servo and stepper motors, or sending messages to other systems.

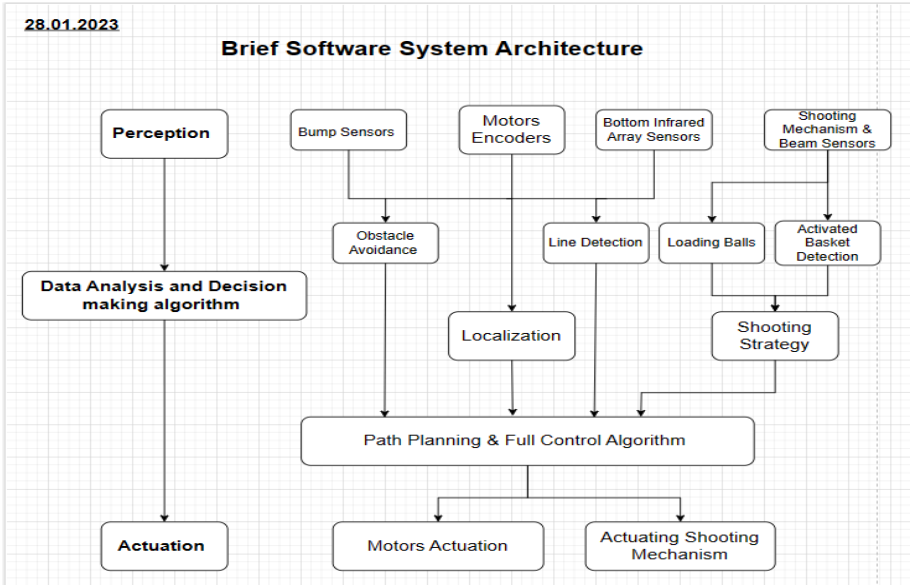


Figure 3. Block Diagram of Robot Logic

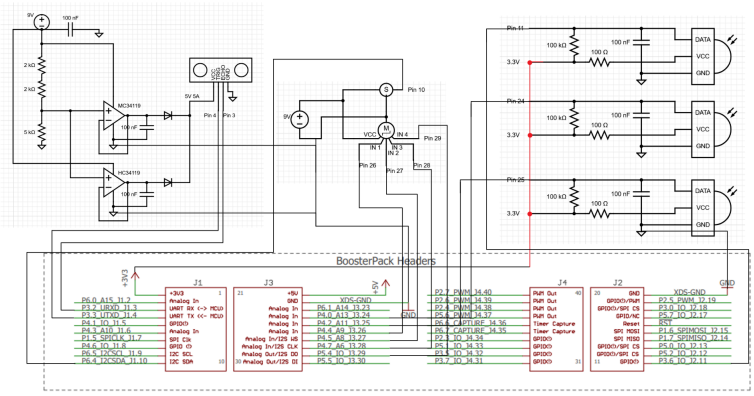


Figure 4. Wiring Diagram

Bill of Materials

The bill of materials contains all of the components used for the final prototype as well as their prices, vendor information, and quantities. Table X below shows the final BOM. The bolts and standoffs were free from TIW, and everything labeled "provided" was from the lab.

Quantity	Item	Cost	Total Cost	Vendor
1	12"x12", 1/4" Wood Plate (Shooting Mechanism)	$9.50	$9.50	TIW
3	10"x12", 1/8" Wood Plate (Loading Mechanism)	$4.74	$14.22	TIW
2	Ruler (Metal-Stainless Steel)	$3.00	$6.00	Walmart
1	DS3235 **Servo Motor (Torque 35kgcm)**	$25.99	$25.99	Amazon - VRNZAU
3	Vishay TSOP34156 **IR Receiver Module**, 56kHz	-	-	Provided
1	28BYJ-48 5V **Stepper Motor**	-	-	Provided
1	HY-SRF05 **Ultrasonic Sensor**	$2.00	$2.00	TIW
1	ULN2003 Driver (**Darlington Array**)	-	-	Provided
50	Breadboard Wires	-	-	Provided
2	Breadboard	$1.00	$2.00	TIW
6	M5 Bolts & Nuts	-	-	TIW
13	M4 Bolts & Nuts	-	-	TIW
21	M3 Standoffs	-	-	TIW
9	M2.5 Bolts	-	-	TIW
	Total	$45.23	$57.71	

Table 2. Final BOM

Conclusions

The timeline of the project overall went well. The shooting mechanism testing was the most test-intensive part of the project. Making sure the shooting mechanism is consistent and reliable is a very important aspect to the success of this project. In order to ensure the shooting

mechanism is consistent, several variables had to be tuned. Some of the variables include the stiffness of the ruler, stiffness of the ball, the cantilevered length of the ruler, as well as the speed of the servo motor deflecting the tip of the ruler. Through various tests, these variables were optimized to get a consistent and reliable shooting. During the demonstration day, there were issues with two of the IR sensors not picking the proper signal they're supposed to pick and caused an issue with finding the proper basket with its respective sensor. Despite efforts made to shield the IR sensors from picking up the wrong signal, making sure the IR sensors detect their intended signal was unsuccessful aside from the middle sensor. A solution was implemented to only use the middle sensor to detect the signals by rotating the robot but not enough time was allotted to test the new code. This robot can be improved by finding a better way to shield the IR sensors and by also optimizing the program code if one IR sensor is used.

Reflection

One of the major concerns throughout the project was the consistency of the shooting mechanism. Over a month of testing, we had to suppress multiple variables that affected the shooting such as the servo horn wearing down, keeping two rulers together through extensive testing, preventing loss of energy through wood deformation, tuning servo speed and ruler length, and finding the right parameters for ruler and servo positioning. The ruler and servo positioning within the system was estimated using physics principles, but the other issues were fixed through extensive testing. The extensive testing phase took a significant portion of the semester, and it was not organized because we became aware of these issues through the course of testing. The large amount of variables made it difficult to tune each one, so having a better documented testing approach would have made this much more time efficient. Besides this, the rest of the design phase went at the desired pace.

Appendix

Appendix A. *Ranking Information and Early Sketches*

	Linear Spring	3 Arm	Flywheel
Ease of Assembly			
Juan	1	2	3
Ali	3	2	1
Russell	3	2	1
Yohannes	3	2	1
Practicality			
Juan	2	3	1
Ali	2	3	1
Russell	2	3	1
Yohannes	1	3	2
Reliability			
Juan	2	3	1
Ali	3	2	1
Russell	1	3	2
Yohannes	2	3	1
Time-Efficiency			
Juan	1	3	2
Ali	1	2	3
Russell	2	3	1
Yohannes	2	3	1

Figure A1. Ranking of Each Category

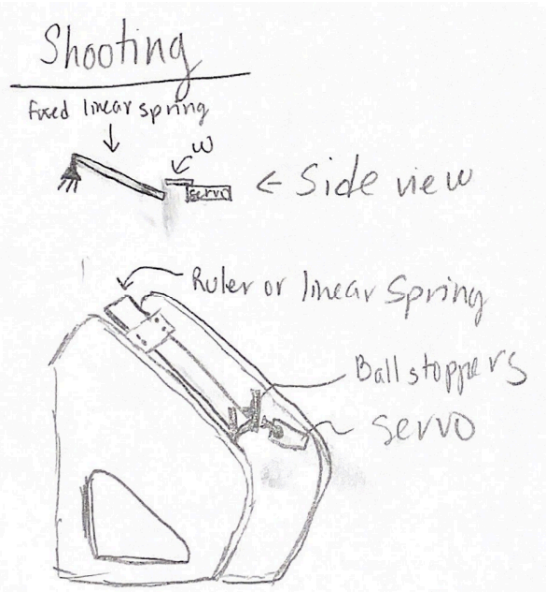


Figure A2. Linear Spring Early Concept

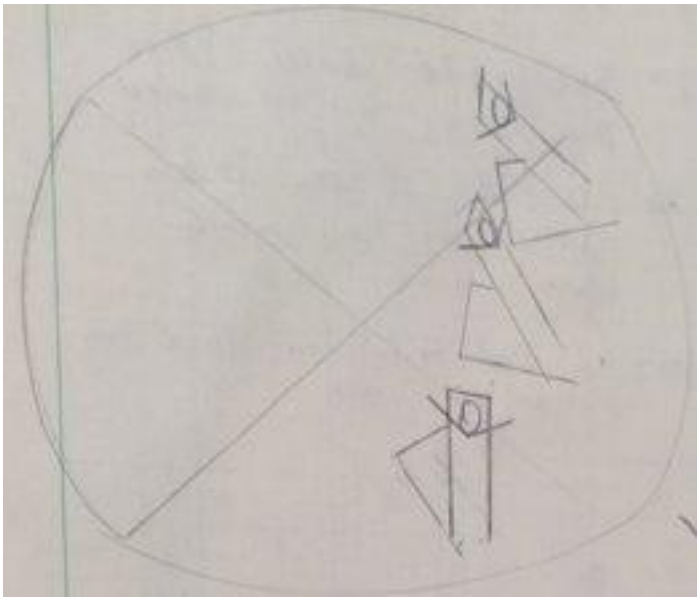


Figure A3. Three Arm Early Concept

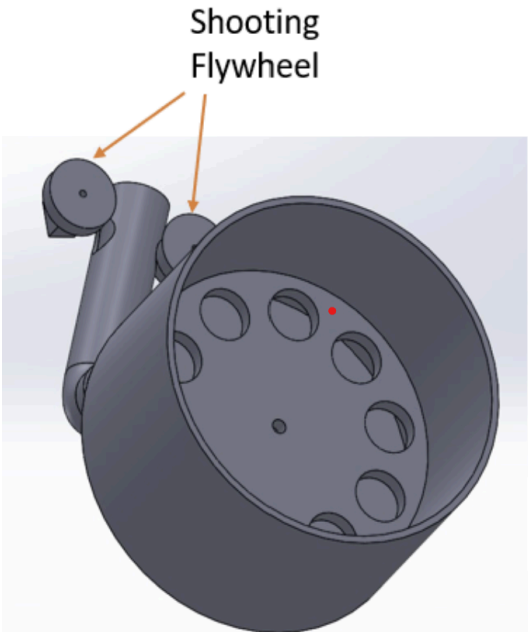


Figure A4. Flywheel Early Concept

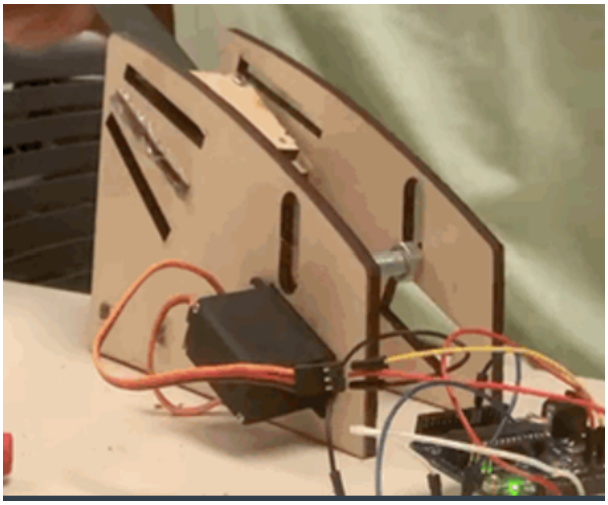


Figure A5. Early Shooting Prototype

Appendix B. M*ultiple Views of Fully Assembled Robot*

Figure B1. Robot Side View

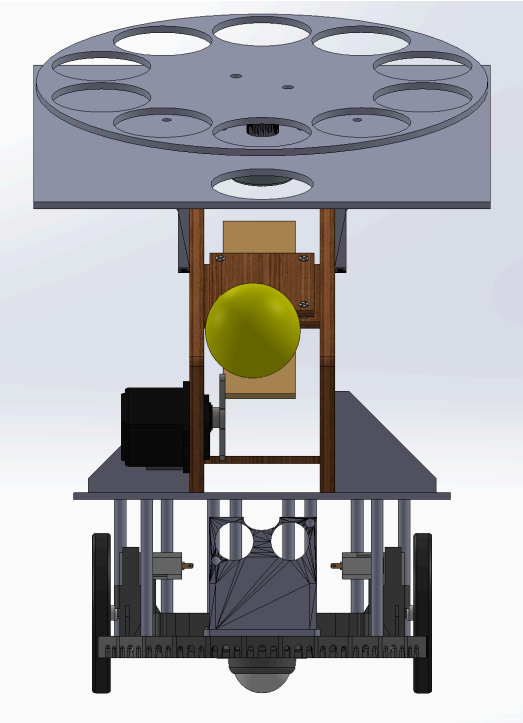


Figure B2. Robot Front View

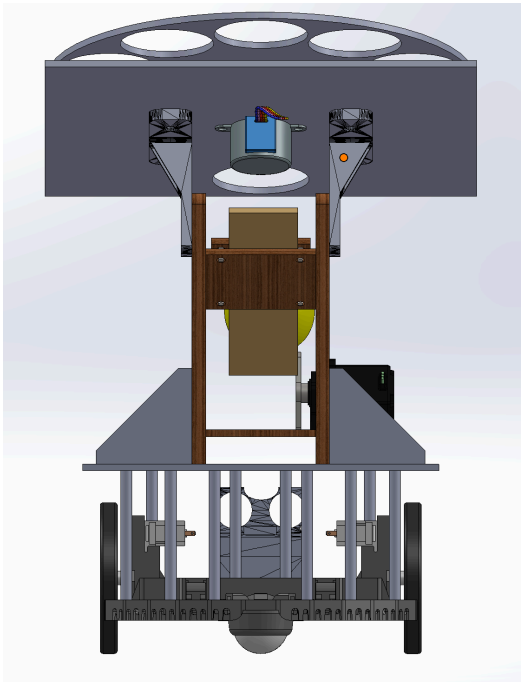


Figure B3. Robot Rear View

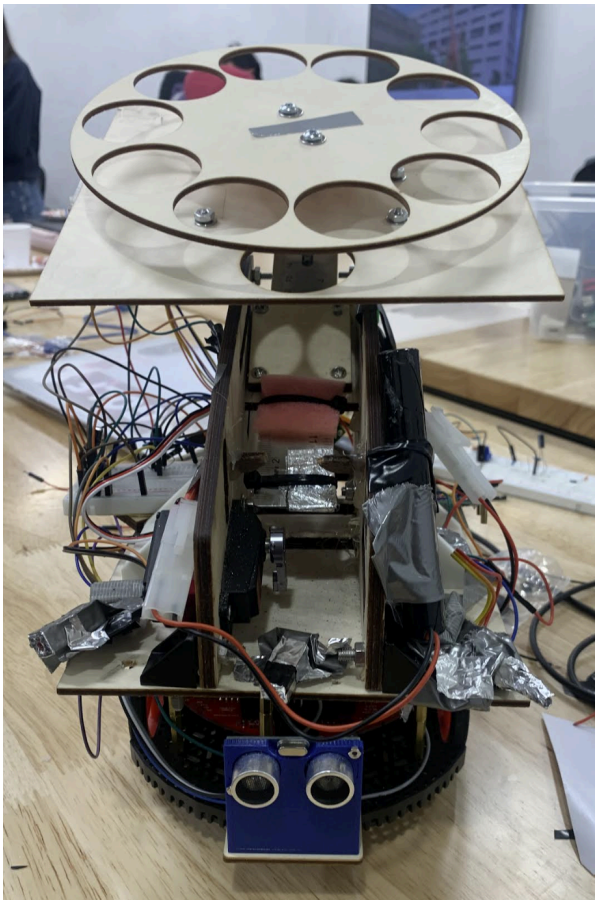


Figure B4. Final Robot

Appendix C. *Final Code*

```
/* Example code for HC-SR04 ultrasonic distance sensor with Arduino. No
library required. More info: https://www.makerguides.com */


#include <Servo.h>
#include "SimpleRSLK.h"
#include "Encoder.h"
#include <cstring>
#include <math.h>

/* Diameter of Romi wheels in inches */
#define wheelDiameter 2.7559055
#define Rd 5.60708661 // robot diamter in inches.
#define cm_inch 0.39370079

/* Number of encoder (rising) pulses every time the wheel turns
completely */
#define cntPerRevolution 360

/* How far in inches for the robot to travel */
int inchesToTravel = (Rd * PI);

int wheelSpeed = 18; // Default raw pwm speed for motor.



uint16_t sensorVal[LS_NUM_SENSORS];
uint16_t sensorCalVal[LS_NUM_SENSORS];
uint16_t sensorMaxVal[LS_NUM_SENSORS];
uint16_t sensorMinVal[LS_NUM_SENSORS];

// Servo shit
unsigned long MOVING_TIME = 1000; // moving time is 3 seconds
unsigned long moveStartTime = 0;
int startAngle = 150; // 30°
int stopAngle  = 20; // 90°
int long progress = 0;
int long angle = 0;


//IR
// constants
#define IRbeacon1 11
#define IRbeacon2 24
#define IRbeacon3 25
// counter
int IR1CNTL = 0;
int IR2CNTC = 0;
int IR3CNTR = 0;
int LC = 0; //loop count
// time
long previousMillis = 0;
int avgInterval = 200;
long interval = 0;
#define SAMPLES 100 // IR samples

//stepper pins
#define IN1 26
#define IN2 27
#define IN3 28
```

```
#define IN4 29


//servo
Servo myServo;
// twelve servo objects can be created on most boards

int pos = 180;    // variable to store the servo position
int Steps = 0;
boolean Direction = true;

// Define Trig and Echo pin:
#define LED_R 5
#define LED_G 6
#define echoPin 3
#define trigPin 4
#define SAMPLE_SIZE 100
#define CENTER 60
#define recorrection_speed 8
#define WALL_DIST 85
int SEARCH = 1;
int SEARCH_R = 1;
int SEARCH_W = 1;

// Define variables:
long duration;
float distance;
uint16_t I_S = 0;
unsigned long last_time = 0;
double total_error = 0, last_error = 0;
int max_control = 50;
int min_control = -50;
int Integral_max = 30;
int Integral_min = -30;
unsigned long current_time = 0;
int delta_error = 0;
int delta_time = 0;

int motorSpeed = 40;

int u=0,v_l,v_r, v = 15, vmax = 40;
float error = 0, Kp = 2.2/2.1, Ki = 0,  Kd = 3.4/2;
// float error = 0, Kp = 0.3, Ki = 0.12,  Kd = 0.4;
// Kp = 2.2/2, Ki = 0, Kd = 4/2;

#define SAMPLE_SIZE 100
#define SIZE 1100
#define Start_Btn 5
#define T 20
int EN_Last = 0;
int motorSpeed_slow = 10;
int T_C = 1;
int F_Index = 0;


typedef enum LINE {
  RIGHT,
  LEFT,
}LINE;
```

```
typedef struct Measurements{
  int EN_RD_L[SIZE];
  int EN_RD_R[SIZE];
  int DIST[SIZE];
  Measurements() {
    memset(EN_RD_L, 0, SIZE * sizeof(int));
    memset(EN_RD_R, 0, SIZE * sizeof(int));
    memset(DIST, 0, SIZE * sizeof(int));
  }
} Measurements;

class Sensor_Byte
  {
  public:

  Sensor_Byte() {a = 0b00000000;}
  unsigned char BitArray(uint16_t* Array) // Must be an 8 bit array
  {
    copy(Array);
    for(int i = 0; i < LS_NUM_SENSORS ; i++) //0b1110010
    {

      (values[i] < 1500) ? values[i] = 0 : values[i] = 1;
       Serial.print(values[i]);
      (values[i]) ? a = (a << 1) + 1 : a = a << 1;
    }
    Serial.println("");
    Serial.println(a);

    return a;
  }

  int FindAngle(void)
  {
  //  Serial.println("Accessing Switch case");
      switch (a)
      {
        case 0b00011000:
        theta = 0; // Angle deviation from centre
        break;
        case 0b00111100:
            theta = 0;
            break;
        case 0b10000000:
            theta = 30/3.3;
            break;
        case 0b11000000:
            theta = 27/3.3;
            break;
        case 0b11100000:
            theta = 19/3.3;
            break;
        case 0b01110000:
            theta = 15/3.3;
        case 0b00110000:
            theta = 10/1.8;
            break;
        case 0b00111000:
            theta = 5;
```

```
            break;
        case 0b00011110:
            theta = -3;
            break;
        case 0b00011100:
            theta = -4;
            break;
        case 0b00001110:
            theta = -10/1.8;
            break;
        case 0b00000110:
            theta = -19/2.5;
            break;
        case 0b00000111:
            theta = -21/2.5;
            break;
        case 0b00000011:
            theta = -27/3.3;
            break;
        case 0b00000001:
            theta = -30/3.3;
            break;
        case 0b00000000:
            theta = 0;
            break;
        case 0b11111111:
            theta = 0;
            break;
          default:
            (a > 0x08) ? theta = (int)(a >> 4) : theta = (int)((a-
0x20))/3;
            break;
      }

       return theta;
  }

  private:
  unsigned char a;
  int theta = 0;
  uint16_t values[LS_NUM_SENSORS];

  void copy(uint16_t* Array)
  {
    for(int i = 0; i < LS_NUM_SENSORS; i++ ) values[i] = Array[i];
  }

  };


float Get_Distance();
void LineFollowing(int theta);
void spin_left();
void spin_right();
void spin_revert(int EN_LEFT, int EN_RIGHT);
void check_intersection();
void find_wall();
void Calibration_Sequence();
void GO_TO_DEST();
void Go_forward();
```

```
LINE Distance_Verification();
void Go_dist(float distance);
float distanceTraveled(float wheel_diam, uint16_t cnt_per_rev, uint8_t
current_cnt);
uint32_t countForDistance(float wheel_diam, uint16_t cnt_per_rev,
uint32_t distance);
float Get_mean();
void Test_Sequence();
void rotateLeft();
void reload();
void shoot();
void rotateRight();
void stepperHALF(int xw);
void IR_Scan();
void Recorrection();
void RESET_MATRIX();
void  Tune_PID();
 void recorrection(); // russell


/* The distance the wheel turns per revolution is equal to the diameter *
PI.
 * The distance the wheel turns per encoder pulse is equal to the above
divided
 * by the number of pulses per revolution.
 */
Sensor_Byte _Byte;
Measurements Wall;

int counter = 0;

void setup() {
  // Define inputs and outputs:
  // Ultrasonic
   moveStartTime = millis(); // start moving
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  //IR
  pinMode(IRbeacon1, INPUT); // NOTE: because this is a pullup, a 1
indicates no beacon detected, 0 is yes beacon detected
  pinMode(IRbeacon2, INPUT);
  pinMode(IRbeacon3, INPUT);
  //servo and stepper
  myServo.attach(10);  // attaches the servo on pin 10 to the servo
object
  myServo.write(220);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  // Start button
 // pinMode(Start_Btn, INPUT_PULLUP); // intialized internally within
setWaitBtn function
  Serial.println("SYSTEM BOOT UP......");
  Serial.begin(115200);

  setupRSLK();
  /* Left button on Launchpad */
  //setupWaitBtn(LP_LEFT_BTN);
  setupWaitBtn(Start_Btn);
```

```
  /* Red led in rgb led */
  setupLed(RED_LED);
  String btnMsg = "Push button on Breadboard to begin calibration.\n";
  btnMsg += "Make sure the robot is on the floor away from the line.\n";
  /* Wait until button is pressed to start robot */
  waitBtnPressed(Start_Btn,btnMsg,RED_LED);
  delay(2000);
     /* Enable both motors */
  enableMotor(BOTH_MOTORS);
  for(int i = 0; i < 10 ; i++) Get_Distance(); // U_S noise
 Calibration_Sequence();  // turn this back on.
//  spin_left();
}
int d = 0;
void Test_once();
void  Test_delay();
void Test_S_R();
void Recorrection();
void loop() {
   IR_Scan();
  // Test_Sequence();
 // Tune_PID();
 //   Test_once();
 // Test_delay();
 // Test_S_R();
// Recorrection();
}

void Recorrection()
{
  for(int i = 0; i<SIZE; i++)
  {
      Wall.DIST[i] = 0;
      Wall.EN_RD_L[i] = 0;
      Wall.EN_RD_R[i] = 0;
  }
   for(int i = 0; i < 10 ; i++) Get_Distance(); // U_S noise
  spin_360();
  find_wall();
//  while(1){}
}
void Test_S_R()
{
    reload();
      shoot();
   while(1){}
}
void  Test_delay()
{
   readLineSensor(sensorVal);
  Go_forward();

  if( _Byte.BitArray(sensorVal) == 255)
    {
       pauseMotor(BOTH_MOTORS);
       Go_dist(4.5);
       pauseMotor(BOTH_MOTORS);
       spin_left();
       while(1){}
    }
```

```
}
void Test_once()
{
    if(d == 0)
  {
     spin_left();
     d++;
  }
}
void Calibration_Sequence()
{
  spin_360();
  find_wall();
  GO_TO_DEST();
  reload();
  shoot();
 // Recorrection();

}

void GO_TO_DEST()
{
   readLineSensor(sensorVal);
 /*
  while(1)
  { // Go_forward();
   spin_left();
  // readLineSensor(sensorVal);
 //  _Byte.BitArray(sensorVal);
//   LineFollowing(_Byte.FindAngle());
   if( _Byte.BitArray(sensorVal) == 255)
    {
      Serial.println("Line detected");
      delay(70); // just until COM is above line.
      pauseMotor(BOTH_MOTORS);
      break;
    }
    }
    */
   // Line detected
    spin_left();

   if (Distance_Verification() ==  LEFT)
   {
      digitalWrite(5, HIGH);
      Serial.println("LEFT wall");
      spin_revert(0,0);
      Go_forward();
    while(1){
       readLineSensor(sensorVal);
      _Byte.BitArray(sensorVal);
      if( _Byte.BitArray(sensorVal) == 255)
    {
       Serial.print("Line detected");
       pauseMotor(BOTH_MOTORS);
       Go_dist(4.5);
       spin_left();
       break;
    }
    }
```

```
    Go_forward();
    while (1)
    {
       readLineSensor(sensorVal);
       _Byte.BitArray(sensorVal);
      LineFollowing(_Byte.FindAngle());
      if( _Byte.BitArray(sensorVal) == 255)
    {
       Serial.print("Arrived to destination");
       pauseMotor(BOTH_MOTORS);
       Go_dist(4.5);
       break;
    }
    }

   }
   else
   {
        Serial.println("Right wall");
        digitalWrite(6, HIGH);
        Go_forward();
      while(1){
         readLineSensor(sensorVal);
        if( _Byte.BitArray(sensorVal) == 255)
      {
        Serial.print("Line detected");
        pauseMotor(BOTH_MOTORS);
        Go_dist(4.5);
        spin_right();
        pauseMotor(BOTH_MOTORS);
        break;
      }
      }
      Go_forward();
      while (1)
      {
         readLineSensor(sensorVal);
         _Byte.BitArray(sensorVal);
        LineFollowing(_Byte.FindAngle());
        if( _Byte.BitArray(sensorVal) == 255)
      {
        Serial.print("Arrived to destination");
        pauseMotor(BOTH_MOTORS);
        pauseMotor(BOTH_MOTORS);
        Go_dist(4.5);
        break;
      }
      }


   }

}


void Go_dist(float distance)
{
             Serial.print("--MOVING--");
             Serial.print(distance);
             Serial.println(" cm FORWARD");
```

```
              uint16_t x = countForDistance(wheelDiameter,
cntPerRevolution, distance * cm_inch);
              resetLeftEncoderCnt();
              resetRightEncoderCnt();

              setMotorDirection(BOTH_MOTORS,MOTOR_DIR_FORWARD);

              enableMotor(BOTH_MOTORS);

              /* Set motor speed */
              setMotorSpeed(BOTH_MOTORS,wheelSpeed);

              /* Drive motor until it has received x pulses */
              while(getEncoderLeftCnt() <=  x  || getEncoderRightCnt() <=
x);

               pauseMotor(BOTH_MOTORS);
}

LINE Distance_Verification()
{
  float mean = 0;
  Serial.println("Sample DIstance: ");
  Serial.print(Get_Distance());
  Serial.println(" cm");
  for(int i = 1; i<SAMPLE_SIZE; i++)
  {
    mean = (mean + Get_Distance()); // Calculating mean.
  }
  mean = mean / SAMPLE_SIZE;
  Serial.println("DIstance: ");
  Serial.print(mean);
  Serial.println(" cm");
  delay(1000);
  if(mean < WALL_DIST) return LEFT; // possible bug
  else return RIGHT;
}

float Get_mean()
{
  float mean = 0;
  for(int i = 1; i<SAMPLE_SIZE; i++)
  {
    mean = (mean + Get_Distance() ) / i ; // Calculating mean.
  }
  return mean;
}


void Go_forward()
{
          setMotorDirection(BOTH_MOTORS,MOTOR_DIR_FORWARD);
          enableMotor(BOTH_MOTORS);
          setMotorSpeed(BOTH_MOTORS,wheelSpeed);
          resumeMotor(BOTH_MOTORS);
}
```

```
void find_wall()
{

    Serial.println("----STARTING WALL SEARCH----");
    int Min_Value =  Wall.DIST[0]; // initialize max to the first element
of the array
    int Min_Index = 0;
     for (int i = 1; i < F_Index; i++) {
      if ( Wall.DIST[i] < Min_Value) {
         Min_Index = i;
         Min_Value =  Wall.DIST[i];
       }
     }
     Serial.print("----Min_Index ----");
     Serial.println(Min_Index);
     Serial.print("----Min_Value ----");
     Serial.println(Min_Value);
     Serial.print("----EN_Left ----");
     Serial.print(Wall.EN_RD_L[Min_Index]);
     Serial.print("----EN_Right ----");
     Serial.println(Wall.EN_RD_R[Min_Index]);

    spin_revert(Wall.EN_RD_L[Min_Index], Wall.EN_RD_R[Min_Index]);
    delay(1000);
    Serial.println("Going to line ....");


}
void check_intersection()
{
     if( _Byte.BitArray(sensorVal) == 255)
    {
       if((millis() - current_time) >= 900)
      I_S++;
      current_time = millis();
    }
}

void spin_left()
{
          Serial.println("Spinning left");
          uint16_t x = countForDistance(wheelDiameter, cntPerRevolution,
inchesToTravel);
              resetLeftEncoderCnt();
              resetRightEncoderCnt();

              setMotorDirection(LEFT_MOTOR,MOTOR_DIR_BACKWARD);
              setMotorDirection(RIGHT_MOTOR,MOTOR_DIR_FORWARD);

              enableMotor(BOTH_MOTORS);

              setMotorSpeed(BOTH_MOTORS,wheelSpeed);
              resumeMotor(BOTH_MOTORS);
              /* Drive motor until it has received x pulses */
              while(getEncoderLeftCnt() <= x/4 || getEncoderRightCnt() <=
x/4);
              /* pause motors */
             /*
              while(_Byte.BitArray(sensorVal) != CENTER)
              {
```

```
                int offset_recorrection = _Byte.BitArray(sensorVal) -
CENTER ;

                  if(offset_recorrection > 0)
                  {
                    Serial.println("Right Recorrection Starting ...");
                    setMotorDirection(LEFT_MOTOR,MOTOR_DIR_BACKWARD);
                    setMotorDirection(RIGHT_MOTOR,MOTOR_DIR_FORWARD);

                    enableMotor(BOTH_MOTORS);

                    setMotorSpeed(BOTH_MOTORS, recorrection_speed);
                    resumeMotor(BOTH_MOTORS);
                    delay(2);
                  }
                  else
                  {
                    Serial.println("Left Recorrection Starting ...");
                    setMotorDirection(LEFT_MOTOR,MOTOR_DIR_FORWARD);
                    setMotorDirection(RIGHT_MOTOR,MOTOR_DIR_BACKWARD);

                    enableMotor(BOTH_MOTORS);

                    setMotorSpeed(BOTH_MOTORS, recorrection_speed);
                    resumeMotor(BOTH_MOTORS);
                  }

               }
               */
            //   Serial.println("Recorrection finished");
               pauseMotor(BOTH_MOTORS);

}

void spin_right()
{
          Serial.println("Spinning right");
    uint16_t x = countForDistance(wheelDiameter, cntPerRevolution,
inchesToTravel);
               resetLeftEncoderCnt();
               resetRightEncoderCnt();

               setMotorDirection(LEFT_MOTOR,MOTOR_DIR_FORWARD);
               setMotorDirection(RIGHT_MOTOR,MOTOR_DIR_BACKWARD);

               enableMotor(BOTH_MOTORS);

               setMotorSpeed(BOTH_MOTORS,wheelSpeed);
               resumeMotor(BOTH_MOTORS);
               /* Drive motor until it has received x pulses */
               while(getEncoderLeftCnt() <= x/4 || getEncoderRightCnt() <=
x/4);
               /* pause motors */
               pauseMotor(BOTH_MOTORS);

}
void spin_revert(int EN_LEFT, int EN_RIGHT)
{
```

```
              Serial.println("----STARTING REVERT----");

              uint16_t x = countForDistance(wheelDiameter,
cntPerRevolution, inchesToTravel);
              resetLeftEncoderCnt();
              resetRightEncoderCnt();

              setMotorDirection(LEFT_MOTOR,MOTOR_DIR_FORWARD);
              setMotorDirection(RIGHT_MOTOR,MOTOR_DIR_BACKWARD);

              /* "Turn on" the motor */
              enableMotor(BOTH_MOTORS);

              /* Set motor speed */
              setMotorSpeed(BOTH_MOTORS,wheelSpeed);
              resumeMotor(BOTH_MOTORS);
              /* Drive motor until it has received x pulses */
              while(getEncoderLeftCnt() <= EN_LEFT + x/2  ||
getEncoderRightCnt() <= EN_RIGHT + x/2);

               pauseMotor(BOTH_MOTORS);

}

void spin_360()
{
              int i = 0;
              /* Set the encoder pulses count back to zero */
              uint16_t x = countForDistance(wheelDiameter,
cntPerRevolution, inchesToTravel);
              resetLeftEncoderCnt();
              resetRightEncoderCnt();
              Wall.DIST[0] = Get_Distance();
              Wall.EN_RD_L[0] = getEncoderLeftCnt();
              Wall.EN_RD_R[0] = getEncoderRightCnt();

                Serial.print("--- LEFT --- ");
                Serial.print( Wall.EN_RD_L[0]);
                Serial.print(" --- RIGHT --- ");
                Serial.print( Wall.EN_RD_R[0]);
                Serial.print(" --- DISTANCE --- ");
                Serial.print( Wall.DIST[0]);
                Serial.print(" cm --- i = ");
                Serial.println(i);
                i++;
              /* Cause the robot to drive forward */
              setMotorDirection(LEFT_MOTOR,MOTOR_DIR_FORWARD);
              setMotorDirection(RIGHT_MOTOR,MOTOR_DIR_BACKWARD);

              /* "Turn on" the motor */
              enableMotor(BOTH_MOTORS);

              /* Set motor speed */
              setMotorSpeed(BOTH_MOTORS,wheelSpeed);
            resumeMotor(BOTH_MOTORS);
              /* Drive motor until it has received x pulses */
              while(getEncoderLeftCnt() <= x || getEncoderRightCnt() <=
x)
              {
                Wall.DIST[i] = Get_Distance();
```

```
                Wall.EN_RD_L[i] = getEncoderLeftCnt();
                Wall.EN_RD_R[i] = getEncoderRightCnt();
                Serial.print("--- LEFT --- ");
                Serial.print( Wall.EN_RD_L[i]);
                Serial.print(" --- RIGHT --- ");
                Serial.print( Wall.EN_RD_R[i]);
                Serial.print(" --- DISTANCE --- ");
                Serial.print( Wall.DIST[i]);
                Serial.print(" cm --- i = ");
                Serial.println(i);
                i++;

              }
              F_Index = i;
              /* pause motors */
              pauseMotor(BOTH_MOTORS);
              delay(1000);

}

float Get_Distance()
{
  restart:
  // Clear the trigPin by setting it LOW:
  digitalWrite(trigPin, LOW);
  delayMicroseconds(5);

  // Trigger the sensor by setting the trigPin high for 10 microseconds:
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(20);
  digitalWrite(trigPin, LOW);

  // Read the echoPin, pulseIn() returns the duration (length of the
pulse) in microseconds:
  duration = pulseIn(echoPin, HIGH);
  // Calculate the distance:
  distance = (duration * 0.034 / 2);

  if(distance > 400 || distance < 2)
  {
    goto restart;
  }

  // Print the distance on the Serial Monitor (Ctrl+Shift+M):

  delay(10);
  return distance;
}


float distanceTraveled(float wheel_diam, uint16_t cnt_per_rev, uint8_t
current_cnt) {
  float temp = (wheel_diam * PI * current_cnt) / cnt_per_rev;
  return temp;
}


uint32_t countForDistance(float wheel_diam, uint16_t cnt_per_rev,
uint32_t distance) {
  float temp = (wheel_diam * PI) / cnt_per_rev;
```

```
  temp = distance / temp;
  return int(temp);
}

void LineFollowing(int theta){
  Serial.println("theta: " + (String)theta);
    error = -theta; // error with respect to 0 reference
    total_error += error; //accumalates the error - integral term
 //   Serial.print("error: ");
 //   Serial.println(error);
 //   Serial.print("total_error: ");
 //   Serial.println(total_error);
    if (total_error >= Integral_max) total_error = Integral_max;
    else if (total_error <= Integral_min) total_error = Integral_min;
    delta_error = error - last_error; //difference of error for
derivative term
    Serial.println("Kp*error: " + (String)(error*Kp));
    Serial.println("Ki*total_error: " + (String)(Ki*total_error));
    Serial.println("Kd*delta_error: " + (String)(Kd*delta_error));
    u  = error*Kp + Ki*total_error + (Kd)*delta_error; // u  = error*Kp +
Ki*total_error + (Kd)*delta_error;
    last_error = error;
/*
  Serial.println("Accessing Control Loop");
  error = -theta; // error with respect to 0 reference
  total_error += error; //accumalates the error - integral term
  Serial.print("error: ");
  Serial.println(error);
  Serial.print("total_error: ");
  Serial.println(total_error);
  if (total_error >= max_control) total_error = 0;
  else if (total_error <= min_control) total_error = 0;

  delta_error = error - last_error; //difference of error for derivative
term

  u = Kp*error + (Ki*T)*total_error + (Kd/T)*delta_error; //PID control
compute
  Serial.print("u: ");
  Serial.println(u);
  if (u >= max_control) u = max_control;
  else if (u <= min_control) u = min_control;
  last_error = error;
 */
    if (u >= max_control) u = max_control;
    else if (u <= min_control) u = min_control;
    v_l = v  + u;  // right wheel velocity
    v_r = v - u; // left wheel velocity


    Serial.println("--- Motor Velocities before---");
    Serial.println(v_l);
    Serial.println(v_r);
    Serial.println("---------------");

    if(v_l>vmax){
        v_l = vmax;
    }
    else if(v_l<0){
        v_l = 8;
```

```
    }

    if(v_r>vmax){
        v_r = vmax;
    }
    else if(v_r<0){
        v_r = 8;
    }
 //   Serial.println("--- Motor Velocities ---");
 //   Serial.println(v_l);
 //   Serial.println(v_r);
  //  Serial.println("---------------");
    setMotorSpeed(LEFT_MOTOR,v_l);
    setMotorSpeed(RIGHT_MOTOR,v_r);
}
int buff[3] = {0,0,0};
int j = 0;
int INDEX = 0;
void IR_Scan(){
// read the input pin:
  unsigned long currentMillis = millis();
    Serial.println("Scanning ...");
  //  if((counter % 3) == 0 && counter != 0) {Recorrection();counter++;}
    LC++;
    int IRStateL = digitalRead(IRbeacon1);
    int IRStateC = digitalRead(IRbeacon2);
    int IRStateR = digitalRead(IRbeacon3);

    Serial.print("Left state: ");
    Serial.println(IRStateL);
    Serial.print("Centre state: ");
    Serial.println(IRStateC);
    Serial.print("Right state: ");
    Serial.println(IRStateR);

    // print out the state of the button:
    //Serial.println(IRState1); // Note - it's 0 if beacon is ON, 1 if
beacon is OFF. Do some if or case statements to print out more meaningful
information.
    /*
    if (IRStateL == 0) {
      IR1CNTL++;
     // Serial.println("Beam Detected Left");
    }
    if(IRStateC == 0) {
      IR2CNTC++;
   //   Serial.print("\t");
   //   Serial.println("Beam Detected Center");
    }
     if(IRStateR == 0) {
      IR3CNTR++;
    //  Serial.print("\t");Serial.print("\t");
    //  Serial.println("Beam Detected Right");
    }
    */
    for(j = 0 ; j < 3; j++)
    {

        if(j==1) rotateLeft();
        if(j==2) rotateRight();
```

```
    for(int m = 0; m < 110 ; m++)
    {
       if(IRStateC == 0) {
       buff[j]++;
       }
    }
    if(j == 1)
    {
     rotateRight();
    }
    if(j == 2)
    {
     rotateLeft();
    }
  }
  int s = 0;
  int maximum = buff[0];
  s++;
  for(s; s<3; s++)
  {
    if(buff[s] > buff[s-1])
    {
      maximum = buff[s];
      INDEX = s;
    }
  }
  if(maximum == 0)
  {
    reload();
    rotateLeft();
    shoot();
    rotateRight();
  }
  else if(INDEX == 0)
  {
    reload();
    shoot();
  }
  else if(INDEX == 1)
  {
    reload();
    rotateLeft();
    shoot();
    rotateRight();
  }
  else
  {
    reload();
    rotateRight();
    shoot();
    rotateLeft();
  }

  /*
if(LC >= SAMPLES)
{

  int IRS[3] = {IR1CNTL,IR2CNTC,IR3CNTR};
  int i = 0;
  int maximum = IRS[i];
```

```
    i++;
    for(i; i<3; i++)
    {
      if(IRS[i] > IRS[i-1])
      {
        maximum  = IRS[i];
      }
    }

    Serial.print("Maximum: ");
    Serial.println(maximum);
    Serial.print("Left counts: ");
    Serial.println(IR1CNTL); //a
    Serial.print("Centre counts: ");
    Serial.println(IR2CNTC);//b
    Serial.print("Right: ");
    Serial.println(IR3CNTR);//c

    if(maximum == 0)
    {
      Serial.println("ALL OFF");
     // delay(2000);
    }
    else
    {
    if(maximum == IR1CNTL) {
      counter++;
      Serial.println("LEFT BEAM");
      reload();
      rotateLeft();
      shoot();
      rotateRight();
    }

    else if(maximum == IR2CNTC) {
      counter++;
      Serial.println("CENTRE BEAM");
      reload();
      shoot();
    }
    else if(maximum == IR3CNTR){
      counter++;
      Serial.println("RIGHT BEAM");
      reload();
      rotateRight();

      shoot();
      rotateLeft();
    }
 //   delay(2000);
    }
    IR1CNTL = 0;
    IR2CNTC = 0;
    IR3CNTR = 0;
    LC = 0;
  }
  */
}

void reload(){
```

```
  for(int i=0; i<410; i++){ // What is 4096? You'll have to change things
here to do rotational control
  stepperHALF(1);
  Serial.println(Steps);
  delayMicroseconds(1500); // Don't change this number - see if you can
find it on the scope!
  }

  delay(400);
}

void shoot()
{
    moveStartTime = millis();
    progress = 0;
    while (progress <= MOVING_TIME) {
        progress = millis() - moveStartTime;
        angle = map(progress, 0, MOVING_TIME, startAngle, stopAngle);
        Serial.println(angle);
        myServo.write(angle);
    }

    angle = 0;
    myServo.write(220);
    delay(1000);
    myServo.write(150);
}

void stepperHALF(int xw) {
  for (int x = 0; x < xw; x++) {
  switch (Steps) {
  case 0:
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  break;

  case 1:
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  break;

  case 2:
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  break;

  case 3:
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  break;
```

```
  case 4:
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  break;

  case 5:
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, HIGH);
  break;

  case 6:
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  break;

  case 7:
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  break;

  default:
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  break;
  }
  SetDirection();
  }
  }

void SetDirection() {
  if (Direction == 1) {
  Steps++;
  }
  if (Direction == 0) {
  Steps--;
  }
  if (Steps > 7) {
  Steps = 0;
  }
  if (Steps < 0) {
  Steps = 7;
  }
}

void rotateLeft()
{
    Serial.println("Spinning left");
    uint16_t x = countForDistance(wheelDiameter, cntPerRevolution,
inchesToTravel);
    resetLeftEncoderCnt();
```

```
    resetRightEncoderCnt();

    setMotorDirection(LEFT_MOTOR,MOTOR_DIR_BACKWARD);
    setMotorDirection(RIGHT_MOTOR,MOTOR_DIR_FORWARD);

    enableMotor(BOTH_MOTORS);

    setMotorSpeed(BOTH_MOTORS,wheelSpeed + 5);
    resumeMotor(BOTH_MOTORS);
    /* Drive motor until it has received x pulses */
    while(getEncoderLeftCnt() <= x/16 || getEncoderRightCnt() <= x/16);
    //   Serial.println("Recorrection finished");
    pauseMotor(BOTH_MOTORS);
}

void rotateRight()
{
    Serial.println("Spinning right");
    uint16_t x = countForDistance(wheelDiameter, cntPerRevolution,
inchesToTravel);
    resetLeftEncoderCnt();
    resetRightEncoderCnt();

    setMotorDirection(LEFT_MOTOR,MOTOR_DIR_FORWARD);
    setMotorDirection(RIGHT_MOTOR,MOTOR_DIR_BACKWARD);

    enableMotor(BOTH_MOTORS);

    setMotorSpeed(BOTH_MOTORS,wheelSpeed + 5);
    resumeMotor(BOTH_MOTORS);
    /* Drive motor until it has received x pulses */
    while(getEncoderLeftCnt() <= x/16 || getEncoderRightCnt() <= x/16);
    /* pause motors */
    pauseMotor(BOTH_MOTORS);
}
void Test_Sequence()
{
  delay(1000);
  if(T_C == 1) {
      Serial.println("LEFT BEAM");
      rotateLeft();
      reload();
      shoot();
      rotateRight();
      T_C++;
    }
    if(IR2CNTC || T_C == 2) {
      Serial.println("CENTRE BEAM");
      reload();
      shoot();
      T_C++;
    }
    if(IR3CNTR || T_C == 3){
      Serial.println("RIGHT BEAM");
      rotateRight();
      reload();
      shoot();
      rotateLeft();
      T_C++;
    }
```

```
    (T_C == 4) ? T_C = 1 : 0;
    delay(2000);
}
/*void Recorrection()
{
  RESET_MATRIX();
  spin_360();
  find_wall();
}
*/

void RESET_MATRIX()
{
    memset(Wall.EN_RD_L, 0, SIZE * sizeof(int));
    memset(Wall.EN_RD_R, 0, SIZE * sizeof(int));
    memset(Wall.DIST, 0, SIZE * sizeof(int));
}

void  Tune_PID()
{
   readLineSensor(sensorVal);
   _Byte.BitArray(sensorVal);
    LineFollowing(_Byte.FindAngle());
}

//add this class Center
Measurements Center;
 int  E_F_L = 0;
 int  E_F_R =  0;

void recorrection(){ // russell
      Serial.println("Stepping forward");
    uint16_t x = countForDistance(wheelDiameter, cntPerRevolution,
inchesToTravel);
    resetLeftEncoderCnt();
    resetRightEncoderCnt();

    setMotorDirection(LEFT_MOTOR,MOTOR_DIR_BACKWARD);
    setMotorDirection(RIGHT_MOTOR,MOTOR_DIR_FORWARD);

    enableMotor(BOTH_MOTORS);

    setMotorSpeed(BOTH_MOTORS,wheelSpeed);
    resumeMotor(BOTH_MOTORS);
    /* Drive motor until it has received x pulses */
    while(getEncoderLeftCnt() <= x/24 || getEncoderRightCnt() <= x/24);
    pauseMotor(BOTH_MOTORS);
    delay(20);
            int i = 0;
        /* Set the encoder pulses count back to zero */
         x = countForDistance(wheelDiameter, cntPerRevolution,
inchesToTravel);
        resetLeftEncoderCnt();
        resetRightEncoderCnt();
        Center.DIST[0] = Get_Distance();
        Center.EN_RD_L[0] = getEncoderLeftCnt();
        Center.EN_RD_R[0] = getEncoderRightCnt();

        Serial.print("--- LEFT --- ");
        Serial.print( Center.EN_RD_L[0]);
```

```
        Serial.print(" --- RIGHT --- ");
        Serial.print( Center.EN_RD_R[0]);
        Serial.print(" --- DISTANCE --- ");
        Serial.print( Center.DIST[0]);
        Serial.print(" cm --- i = ");
        Serial.println(i);
        i++;
        /* Cause the robot to drive forward */
        setMotorDirection(LEFT_MOTOR,MOTOR_DIR_FORWARD);
        setMotorDirection(RIGHT_MOTOR,MOTOR_DIR_BACKWARD);

        /* "Turn on" the motor */
        enableMotor(BOTH_MOTORS);

        /* Set motor speed */
        setMotorSpeed(BOTH_MOTORS,wheelSpeed);//changed speed
    resumeMotor(BOTH_MOTORS);
        /* Drive motor until it has received x pulses */
        while(getEncoderLeftCnt() <= x/12 || getEncoderRightCnt() <=
x/12)
        {
        Center.DIST[i] = Get_Distance();
        Center.EN_RD_L[i] = getEncoderLeftCnt();
        Center.EN_RD_R[i] = getEncoderRightCnt();
        Serial.print("--- LEFT --- ");
        Serial.print( Center.EN_RD_L[i]);
        Serial.print(" --- RIGHT --- ");
        Serial.print( Center.EN_RD_R[i]);
        Serial.print(" --- DISTANCE --- ");
        Serial.print( Center.DIST[i]);
        Serial.print(" cm --- i = ");
        Serial.println(i);
        i++;

        }
        E_F_L = Center.EN_RD_L[i-1];
        E_F_R = Center.EN_RD_R[i-1];
        F_Index = i;
        /* pause motors */
        pauseMotor(BOTH_MOTORS);
        delay(100);
        Serial.println("----STARTING Center SEARCH----");
        int Min_Value =  Center.DIST[0]; // initialize max to the first
element of the array
        int Min_Index = 0;
        for (int i = 1; i < F_Index; i++) {
        if ( Center.DIST[i] < 10) {
            Center.DIST[i] = 999;
        }
        else if( Center.DIST[i] < Min_Value) {
            Min_Index = i;
            Min_Value =  Center.DIST[i];
        }
        }
        Serial.print("----Min_Index ----");
        Serial.println(Min_Index);
        Serial.print("----Min_Value ----");
        Serial.println(Min_Value);
        Serial.print("----EN_Left ----");
        Serial.print(Center.EN_RD_L[Min_Index]);
```

```
        Serial.print("----EN_Right ----");
        Serial.println(Center.EN_RD_R[Min_Index]);
        Serial.println("----STARTING Center----");

        x = countForDistance(wheelDiameter, cntPerRevolution,
inchesToTravel);
        resetLeftEncoderCnt();
        resetRightEncoderCnt();

        setMotorDirection(LEFT_MOTOR,MOTOR_DIR_BACKWARD);
        setMotorDirection(RIGHT_MOTOR,MOTOR_DIR_FORWARD);

        /* "Turn on" the motor */
        enableMotor(BOTH_MOTORS);

        /* Set motor speed */
        setMotorSpeed(BOTH_MOTORS,wheelSpeed+3);//changed speed
        resumeMotor(BOTH_MOTORS);
        /* Drive motor until it has received x pulses */
        while(getEncoderLeftCnt() <= (E_F_L-(Center.EN_RD_L[Min_Index]))
|| getEncoderRightCnt() <= (E_F_R-(Center.EN_RD_R[Min_Index])));
        pauseMotor(BOTH_MOTORS);
        delay(100);
}
```