Operations Manual

Isabel Barnola, David Bowen, Diego Campos, Alex Ndekeng, Abiel Souverain

FAMU-FSU College of Engineering

The robot in this manual was designed to be run on the playfield in Figure 1. The information given does not apply to any circumstances outside of the playfield.

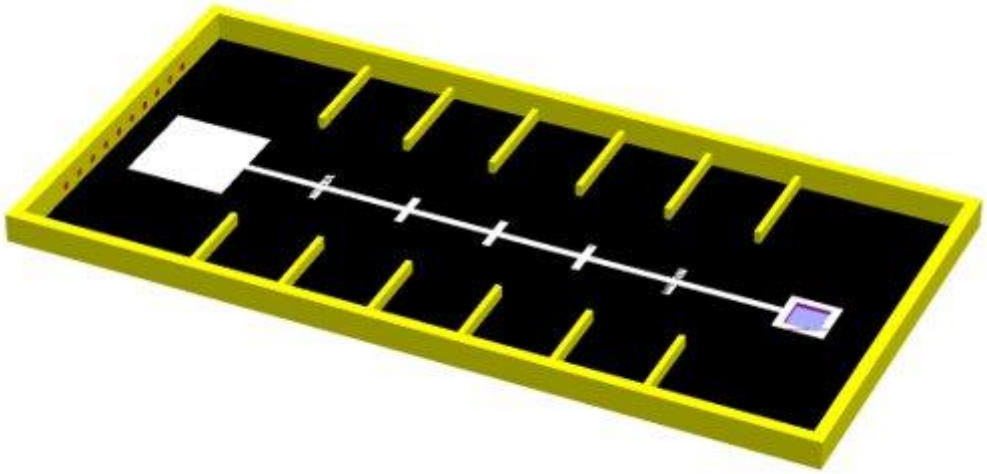


Figure 1. Playfield for robot

The fully assembled product is shown below in Figure 2.

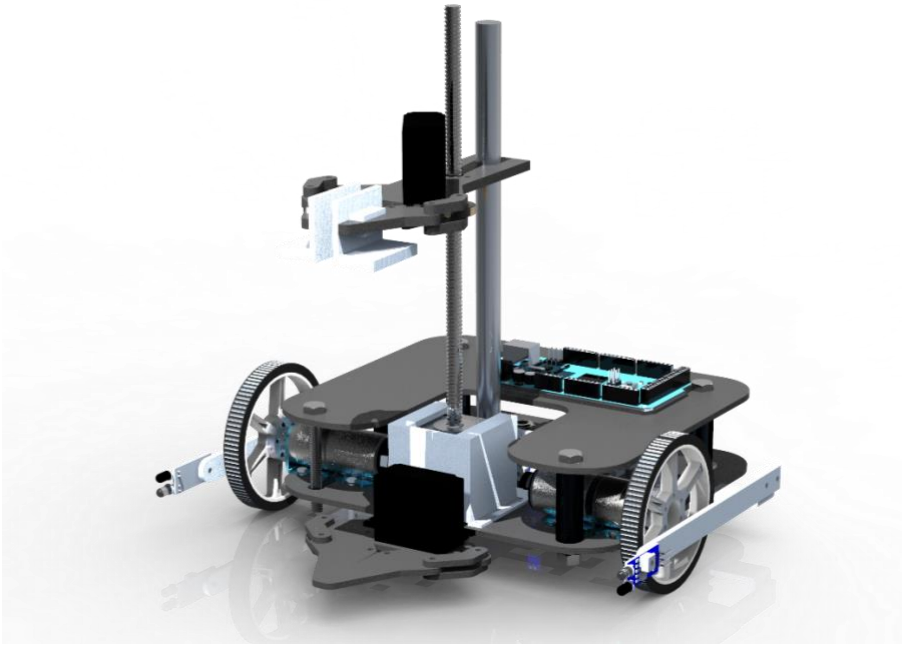


Figure 2. Robot - Model Ezio

Components

- 1x Arduino Mega 2560
- 1x A4988 Stepper Motor Drive Carrier
- 1x 2×15A DC Motor Driver
- 1x Continuous Rotation Servo - FS5103R
- 1x Hitec servo
- 2x Pololu 99:1 Metal Gearmotor 25Dx69L mm LP 6V with 48 CPR Encoder
- 2x Pololu Metal DC motor mounts
- 2x Pololu Pololu Multi-Hub Wheel 80x10mm
- 1x Recessed Flange-Mount Ball Transfer
- 1x Pololu Stepper Motor with 28cm Lead Screw: Bipolar, 200 Steps/Rev, 42×38mm, 2.8V, 1.7 A/Phase, cut to 25cm
- 1x 10in Aluminum Guide Rod
- 9x Obstacle Avoidance IR Sensor
- 1x Assorted Jumper Wires
- 1x 9V Battery
- 1x 12V 2200mAh Li-po Battery

Machined Parts – Drawings on page 10

1x Lower base

1x Upper base

1x 3-IR L-bracket

2x IR L-bracket

2x Side-IR L-bracket

2x Side-IR Arm

1x Stepper Mount

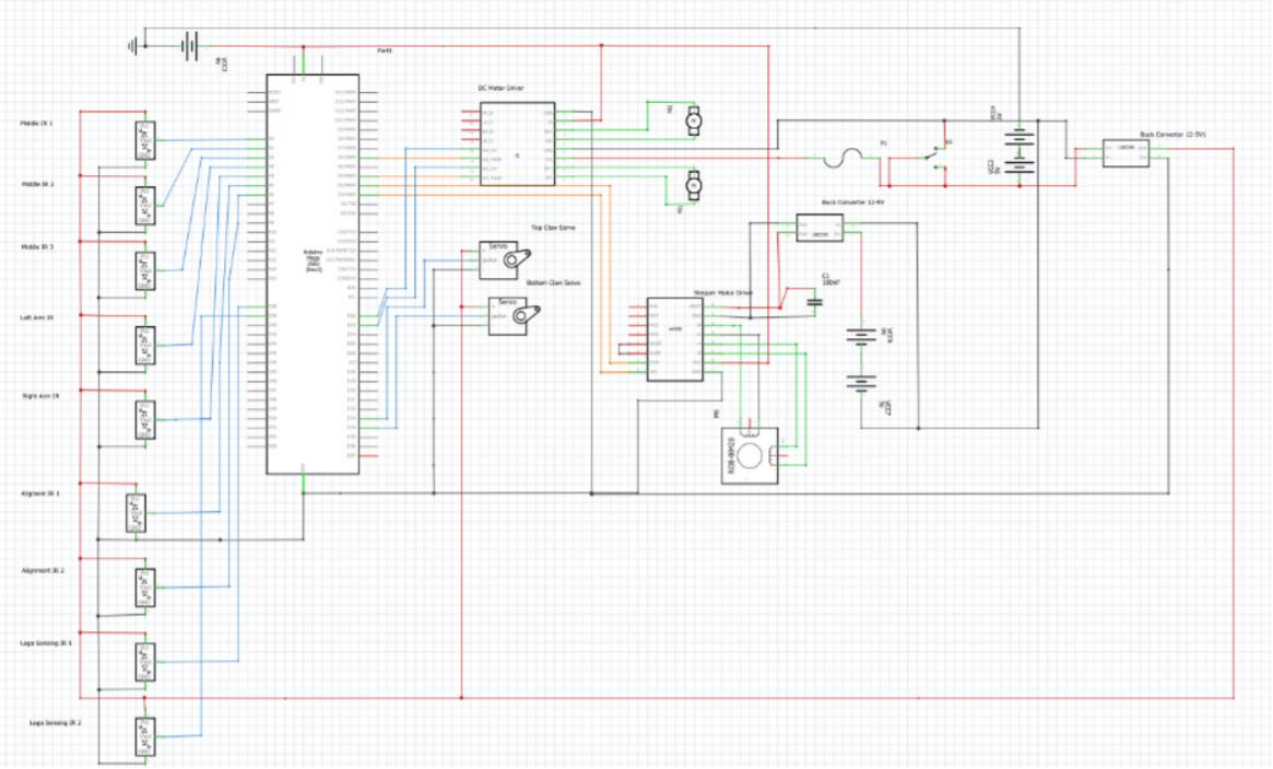
1x Upper Claw

1x Lower Claw

Hardware

- 4x ¼-20 – 1 1/4" Hex bolts
- 8x ¼-20 nuts
- 100x m3x14mm screws
- 100x m3x14mm flat screw
- 100x m3x18mm screws
- 100x m3x12mm screws
- 100x m3 locknuts

Wiring Diagram



Pseudocode

The following code gives an example of how the software for the robot should be written to achieve proper function. The first section contains the main code and its organization. The following section contains the subfunctions used, which include stacking and line following. The stacking function incorporates the use of the stepper motor lead screw as well as the upper and lower claws, controlled by the servos. Because the servo used by the upper claw is a continuous rotation servo, the power to it must be cycled so that the motor does not close too far.

Main code

```
/*
FILE :  Stacking_challenge_7digits
DESCRIPTION: 1st Challenge of SoutheastCon Hardware competition 2020
AUTHOR : ISABEL BARNOLA
```

```
  DATE: 03/03
*/

// ================================================================
// MOTORS
// ================================================================
#define L_Forward  HIGH //LOW
#define R_Forward  LOW  //HIGH
#define L_Backward LOW  //HIGH
#define R_Backward HIGH //LOW
#define plus_factor 5
#define alignement_factor 10
#define go_to_factor 6
#define turn_speed  50
#define turn_speed1 600
int speed_R  = 66;// MAX = 255
int speed_L = 62; // MAX = 255
int slow_R = 55;
int slow_L = 55;

//-----------------------
// Motor : Right
//-----------------------
// M1 PWM
//ORANGE
int R_ME  = 3;  //Enable Pin of the Right Motor (must be PWM)
// M1 EN
// YELLOW
int R_M1 = 23;  //Control Pin

//-----------------------
// Motor : Left
//-----------------------
// GREEN
int L_ME  = 2;   //Enable Pin of the Left Motor (must be PWM)
// BLUE
int L_M1 = 22;
```

```
int dir = 0;
int cnt = 0;

long previousMillis = 0;
long currentMillis = 0;
int interval = 1000;

// ===================================================================
// ENCODERS
// ===================================================================
#include "mrobot.h" //Provided by Dr. Chuy

// ===================================================================
// STACK
// ===================================================================
#include<Servo.h>
#include <Stepper.h>

// change this to fit the number of steps per revolution for your motor
const int stepsPerRevolution = 200;
Servo servo_top, servo_bot;
Stepper myStepper(stepsPerRevolution, 4, 5, 6, 7);
int ir = 5;

//Servo position for each claw
int bopen = 180;
int bclose = 0;
int topen = 80;
int tclose = 110;

// direction of stepper
const int up = 1;
const int down = -1;

// Position of claw
float pos_claw = 0;
```

```
// Stack height
float stack = 0;

// ====================================================================
// IR Sensor
// ====================================================================
//-----------------------
// Sensor: Right
//-----------------------
const int right_sensor_pin =10; // ORANGE
int right_sensor_state;

//-----------------------
// Sensor: Left
//-----------------------
const int left_sensor_pin  = 11; // BLUE
int left_sensor_state;

//-----------------------
// Sensor: Middle
//-----------------------
const int middle_sensor_pin = 12; // WHITE
int middle_sensor_state;

//-----------------------
// Sensor: Left arm
//-----------------------
const int left_arm_sensor_pin  = 13; // GREEN
int left_arm_sensor_state;

//-----------------------
// Sensor: Right arm
//-----------------------
const int right_arm_sensor_pin = 28; // BLUE
int right_arm_sensor_state;

//-----------------------
```

```
// Sensor: FRONT Right
//-----------------------
const int front_right_pin = 29;  // GREEN
int front_right_sensor_state;

//-----------------------
// Sensor: FRONT left
//-----------------------
const int front_left_pin = 30;  // YELLOW
int front_left_sensor_state;

//-----------------------
// Sensor: low right
//-----------------------
const int low_right_pin = 31;  // blue
int low_right_sensor_state;

const int led_r = 33;

//-----------------------
// Sensor: low left
//-----------------------
const int low_left_pin = 32;  // green
int low_left_sensor_state;

const int led_l = 34;


// ================================================================
//TURN_BIN
// ================================================================
//Left
#define t_back_tlb          16  // time to back up on left
#define time_turn_lb        1300  // time to turn on left

//Right
#define t_back_trb          16//1000 // time to back up on right
```

```
#define time_turn_rb        18//1200 // time to turn on right

// ================================================================
//TURN_LINE
// ================================================================
//Left
#define t_back_tll          300  // time to back up on left
//#define time_turn_l_l       1000 // time to turn on left
 int time_turn_l_l[] = {18,0};
int ll = 0;
// Right
#define t_back_tlr          600  // time to back up on right
//#define time_turn_r_l       1100 // time to turn on right
int time_turn_r_l[] ={13,13};
int rl = 0;

// ================================================================
//TURN_BIN 180
// ================================================================
#define time_turn_180_r 2200000
#define time_turn_180_l 2100000

// ================================================================
//TURN_BIN 180
// ================================================================
//Left
#define t_back_tlb_180          700  // time to back up on left
#define time_turn_lb_180        1200  // time to turn on left

//Right
#define t_back_trb_180          850//1000 // time to back up on right
#define time_turn_rb_180        1000//1200 // time to turn on right


// ================================================================
//TURN_LINE 180
// ================================================================
```

```
//Left
#define t_back_tll_180          300  // time to back up on left
#define time_turn_l_l_180       1000 // time to turn on left
// Right
#define t_back_tlr_180          600  // time to back up on right
#define time_turn_r_l_180       1200 // time to turn on right


int e_turn = 0;
int pos = 0;   // used to determine robot's current position

// ================================================================
// STATES
// ================================================================
#define STOP            0
#define LINE_FOLLOWING   1
#define TURN_RIGHT_BIN   2
#define TURN_LEFT_BIN    3
#define TURN_RIGHT_LINE  4
#define TURN_LEFT_LINE   5
#define BACK_TO_LINE     6
#define STACK           7
#define OPPOSITE_R      8
#define GO_TO_BIN       9
#define OPPOSITE_L      10
#define test            11
#define aligned         12

const int num_states_to_do = 38;
int states_to_do[num_states_to_do] = {
                                LINE_FOLLOWING,
                                TURN_LEFT_BIN,
                                GO_TO_BIN,
                                STACK,
                                BACK_TO_LINE,
                                TURN_RIGHT_LINE,
                                aligned,
```

```
                    LINE_FOLLOWING,
                    TURN_LEFT_BIN,
                    GO_TO_BIN,
                    STACK,
                    BACK_TO_LINE,
                    test,
                    aligned,
                    GO_TO_BIN,
                    STACK,
                    BACK_TO_LINE,
                    STOP,
                    test,
                    aligned,
                    GO_TO_BIN,
                    STACK,
                    BACK_TO_LINE,
                    TURN_RIGHT_LINE,
                    aligned,
                    LINE_FOLLOWING,
                    TURN_RIGHT_BIN,
                    GO_TO_BIN,
                    STACK,
                    BACK_TO_LINE,
                    TURN_LEFT_LINE,
                    aligned,
                    LINE_FOLLOWING,
                    aligned,
                    STACK,
                    BACK_TO_LINE,
                    STOP
                    };

int state = LINE_FOLLOWING;
int itr_s = -1;          // used to iterate through array of states
#define RIGHT 10
#define LEFT  11
int last_dir = 0;
```

```
// ================================================================
//LINE_FOLLOWING
// ================================================================
#define BLACK HIGH
#define WHITE LOW
#define plus_dir  1
#define minus_dir -1
const int bins_to_go_LF = 3; // the total number of bins to go with LF
int white_lines = 0;
int lines_to_do[] = {3,2,2,2};
//int direc_to_do[bins_to_go_LF] = {plus_dir};
int do_lf = 0;

// ================================================================
//FUNCTIONS
// ================================================================
#include "FUNCTIONS.h"

// ================================================================
// SETUP
// ================================================================
void setup()
{ // opening setup

  //-----------------------
  // Motors
  //-----------------------
  pinMode(R_M1, OUTPUT);  // Right
  pinMode(L_M1, OUTPUT);  // Left

  //-----------------------
  // IR Sensors
  //-----------------------
  pinMode(right_sensor_pin,   INPUT);   // Right
  pinMode(left_sensor_pin,    INPUT);   // Left
  pinMode(middle_sensor_pin,  INPUT);   // Middle
```

```
  pinMode(left_arm_sensor_pin, INPUT);  // left arm
  pinMode(right_arm_sensor_pin, INPUT); // right arm
  pinMode(low_left_pin, INPUT);         // lower left
  pinMode(low_right_pin, INPUT);        // lower right
  pinMode(front_left_pin,INPUT);        // front left
  pinMode(front_right_pin,INPUT);       // front right
  pinMode(led_r,OUTPUT);
  pinMode(led_l,OUTPUT);

  //----------------------
  // Encoders
  //----------------------
  encoder_init();

  //----------------------
  // Stacking
  //----------------------
  servo_top.attach(9);
  servo_bot.attach(8);
  myStepper.setSpeed(275);

  Serial.begin(9600);  // For serial print (debugging)

}
  // closing setup

  int ii = 0;
// ================================================================
// LOOP
// ================================================================
void loop()
{ // opening loop

  //--------------------------------------
  // STATE: Update
  //--------------------------------------
```

```
state = set_state();
//--------------------------------------
// IR Sensors: READ
//--------------------------------------
 read_ir();

//--------------------------------------
// FSM
//--------------------------------------

switch(state)
{//opening switch

  //-----------------------
  //LINE_FOLLOWING
  //-----------------------
  case LINE_FOLLOWING:
    { //opening LINE_FOLLOWING

       print_state(); // prints the current state to serial port
       Serial.println("LINE_FOLLOWING");
       // Line following - parameters set on top
       line_following(lines_to_do[do_lf],plus_dir);
       do_lf = do_lf + 1;
    }
      //closing LINE_FOLLOWING
    break;

  //-----------------------
  // TURN_RIGHT_BIN
  //-----------------------
  case TURN_RIGHT_BIN:
    { //opening TURN_RIGHT_BIN
      print_state(); // prints the current state to serial port
      Serial.println("TURN_RIGHT_BIN");
      delay(100);
      back_before_turn(t_back_trb);
```

```
    delay(200);
    turn_left_bin(time_turn_rb);
    last_dir = RIGHT;
   }
    //closing TURN_RIGHT_BIN
   break;

//-----------------------
//TURN_LEFT_BIN
//-----------------------
case TURN_LEFT_BIN:
  { //opening TURN_LEFT_BIN
    print_state(); // prints the current state to serial port
    //print_state(); // prints the current state to serial port
    Serial.println("TURN_LEFT_BIN");
    delay(100);
    back_before_turn(t_back_tlb);
    turn_right_bin(time_turn_lb);
    last_dir = LEFT;
  }
    //closing TURN_LEFT_BIN
  break;

//-----------------------
//TURN_RIGHT_LINE
//-----------------------
case TURN_RIGHT_LINE:
  { //opening TURN_RIGHT_LINE
    print_state(); // prints the current state to serial port
    Serial.println("TURN_RIGHT_LINE");
    delay(100);
    back_off_time(t_back_tlr);
    back_off(speed_R, speed_L);
    back_off(speed_R, speed_L);
    //back-off();
    delay(200);
    long int t = time_turn_r_l[rl];
```

```
    rl++;
    turn_right_line(t);
    back_before_turn(3);
  }
    //closing TURN_RIGHT_LINE
  break;

//-----------------------
//TURN_LEFT_LINE
//-----------------------
case TURN_LEFT_LINE:
  { //opening TURN_LEFT_LINE
    print_state(); // prints the current state to serial port
    Serial.println("TURN_LEFT_LINE");
    delay(100);
    back_before_turn(8);

     //advance_before_tll(400);
     stopp();
     delay(300);
     int tt = time_turn_l_l[ll];
    turn_left_line(tt);
    ll++;
  }
    //closing TURN_LEFT_LINE
  break;


//-----------------------
// GO_TO_BIN
//-----------------------
case GO_TO_BIN:
  { //opening GO_TO_BIN
    print_state(); // prints the current state to serial port
    Serial.println("GO_TO_BIN");
     go_to_bin();
  }
```

```
    //closing GO_TO_BIN
   break;

 //-----------------------
 // BACK_TO_LINE
 //-----------------------
 case BACK_TO_LINE:
  { //opening BACK_TO_LINE
    print_state(); // prints the current state to serial port
    Serial.println("BACK_TO_LINE");
    back_to_line();
    stopp();
    delay(1300);
  }
    //closing BACK_TO_LINE
   break;

 //-----------------------
 //STACK
 //-----------------------
 case STACK:
  { //opening STACK
    print_state(); // prints the current state to serial port
    Serial.println("STACK");
     if(stack == 0)
      firstStack();
     else if(stack<5)
      bstack();
     else
      lastStack();
  }
    //closing STACK
   break;

 //-----------------------
 //STOP
 //-----------------------
```

```
    case STOP:
     { //opening STOP
       print_state(); // prints the current state to serial port
       Serial.println("stop");
       stopp();
     }
       //closing STOP
     break;

    //-----------------------
    // ALIGNED
    //-----------------------
    case aligned:
    {

      back_before_turn(6);
      delay(300);
     aligned_bin(millis(),1000);

      }
    break;
    //-----------------------
    // default
    //-----------------------
    default:
     { //opening default
       stopp();
     }
       //closing default
     break;

  }//closing switch

 ii++;
}
  // closing loop
```

Functions

```
// ================================================================
// Function
// ================================================================
void back_before_turn(long  timee);

void turn_180_r_bin (long  turntime);


void turn_180_l_bin(long  turntime);

void print_state()
{
  if (state == STOP)
  {
    Serial.println("STATE = STOP ###################");
  }
  else if (state == LINE_FOLLOWING)
  {
    Serial.println("STATE = LINE_FOLLOWING ###################");
  }
  else if (state == TURN_LEFT_BIN)
  {
    Serial.println("STATE = TURN_LEFT_BIN ###################");
  }
  else if (state == TURN_RIGHT_BIN)
  {
    Serial.println("STATE = TURN_RIGHT_BIN ###################");
  }
  else if (state == TURN_LEFT_LINE)
  {
    Serial.println("STATE = TURN_LEFT_LINE ###################");
```

```
 }
 else if (state == TURN_RIGHT_LINE)
 {
  Serial.println("STATE = TURN_RIGHT_LINE ##################");
 }
 else if(state == GO_TO_BIN)
 {
  Serial.println("STATE = GO_TO_BIN ########################");
 }
 else if(state == BACK_TO_LINE)
 {
  Serial.println("STATE = BACK_TO_LINE #####################");
 }
 else if (state == STACK)
 {
  Serial.println("STATE = STACK ############################");
 }
 else if (state == OPPOSITE_R)
 {
  Serial.println("STATE = OPPOSITE_R #######################");
 }
 else if (state == OPPOSITE_L)
 {
  Serial.println("STATE = OPPOSITE_L #######################");
 }
 else if (state == test)
 {
    Serial.println("STATE = test ############################");
  }
}

int set_state()
{
  if (itr_s < num_states_to_do -1)
  {
   itr_s = itr_s +1;
   return states_to_do[itr_s];
```

```
 }

  return 0 ;
}

void stopp(void)                  //Stop
{
  analogWrite(R_ME,0);
  digitalWrite(R_M1,LOW);
  analogWrite(L_ME,0);
  digitalWrite(L_M1,LOW);
}

void advance(char a,char b)         //Move forward
{

  digitalWrite(L_M1,L_Forward);
   //delay(1);
  digitalWrite(R_M1,R_Forward);
  // delay(1);
  analogWrite (L_ME,a);  //PWM Speed Control
    //delay(1);
  analogWrite (R_ME,b);
  delay(1);

}

void read_ir()
{
  left_sensor_state        =  digitalRead(left_sensor_pin);
  right_sensor_state       =  digitalRead(right_sensor_pin);
  middle_sensor_state      =  digitalRead(middle_sensor_pin);
  left_arm_sensor_state    =  digitalRead(left_arm_sensor_pin);
  right_arm_sensor_state   =  digitalRead(right_arm_sensor_pin);
  front_right_sensor_state =  digitalRead(front_right_pin);
  front_left_sensor_state  =  digitalRead(front_left_pin);
  low_right_sensor_state   =  (digitalRead(low_right_pin));
```

```
  if (low_right_sensor_state == 0)
    digitalWrite(led_r, HIGH);
  else
    digitalWrite(led_r, LOW);
  low_left_sensor_state     =  (digitalRead(low_left_pin));
    if (low_left_sensor_state == 0)
    digitalWrite(led_l, HIGH);
  else
    digitalWrite(led_l, LOW);
}

void back_off (char a,char b)          //Move backward
{
  analogWrite (R_ME,a);
  digitalWrite(R_M1,R_Backward);
  analogWrite (L_ME,b);
  digitalWrite(L_M1,L_Backward);
}

void turn_L (char a,char b)            //Turn Left
{
  analogWrite (R_ME,a);
  digitalWrite(R_M1,R_Forward);
  analogWrite (L_ME,b);
  digitalWrite(L_M1,L_Backward);
}

void turn_L_line (char a)            //Turn Left
{
  analogWrite (R_ME,a);
  digitalWrite(R_M1,R_Forward);
  analogWrite (L_ME,0);
  digitalWrite(L_M1,LOW);
}

void turn_R (char a,char b)            //Turn Right
{
```

```
  analogWrite (R_ME,a);
  digitalWrite(R_M1,R_Backward);
  analogWrite (L_ME,b);
  digitalWrite(L_M1,L_Forward);
}

void turn_R_line (char b)            //Turn Right
{
  analogWrite (R_ME,0);
  digitalWrite(R_M1,LOW);
  analogWrite (L_ME,b);
  digitalWrite(L_M1,L_Forward);
}

void go_straight()
{
  int temp1 = abs(cur_wvel[1] -  cur_wvel[0]);

  if (temp1 >= 0.05 && temp1 <= 0.5 )
  {
    //speed_L =speed_L;
    speed_R =speed_R;
    }
   else if (cur_wvel[1] > cur_wvel[0] )
      speed_R--;
      //speed_L--;
    else if (cur_wvel[1] < cur_wvel[0])
      speed_R++;
      //speed_L++;
}

void go_straight2()
{
  int temp1 = abs(cur_wvel[1] -  cur_wvel[0]);

  if (temp1 >= 0.05 && temp1 <= 0.5 )
  {
```

```
    slow_L =slow_L;
    }
   else if (cur_wvel[1] > cur_wvel[0] )
     slow_L--;
    else if (cur_wvel[1] < cur_wvel[0])
     slow_L++;
}

void fix_turn_r_90_line(int right, int left)
{
  read_ir();
  while(left_sensor_state != BLACK && middle_sensor_state != BLACK && left_sensor_state != BLACK)
  {
    analogWrite (R_ME,right);
    analogWrite (L_ME,left);
    digitalWrite(R_M1,R_Forward);
    digitalWrite(L_M1,L_Backward);
    read_ir();
  }
}

void fix_turn_l_90_line(int right, int left)
{
  read_ir();
  while(left_sensor_state != BLACK && middle_sensor_state != BLACK && left_sensor_state != BLACK)
  {

    analogWrite (R_ME,right);
    analogWrite (L_ME,left);
    digitalWrite(R_M1,R_Backward);
    digitalWrite(L_M1,L_Forward);
    read_ir();
  }
}

void turn_90_r(int right, int left, long time2, int time_to_turn)
{
```

```
  analogWrite (R_ME,right);
  analogWrite (L_ME,left);
  long TimeMillis = 0;
  time2= millis();
 int t = 0;
  while(TimeMillis - time2< time_to_turn)
   {                //Time to turn towards bin
    Serial.println("---------------------Time to turn:  ");
    Serial.println(TimeMillis - time2);
    digitalWrite(R_M1,R_Forward);
    digitalWrite(L_M1,L_Backward);
    TimeMillis = millis();
    t++;
  }
  analogWrite (R_ME,0);
  analogWrite (L_ME, 0);
  //fix_turn_r_90_line(right,left);
}

void turn_90_l(int right, int left, long time2, int time_to_turn)
{
  analogWrite (R_ME,right);
  analogWrite (L_ME,left);
  long TimeMillis = 0;
  time2 = millis();
  int t = 0;
  while(t< time_to_turn)
   {                //Time to turn towards bin
    Serial.println("turn 90 l----------------------Time to turn:  ");
    Serial.println(TimeMillis - time2);
    digitalWrite(R_M1,R_Backward);
    digitalWrite(L_M1,L_Forward);
    TimeMillis = millis();
    t++;
  }
  analogWrite (R_ME,0);
  analogWrite (L_ME, 0);
```

```
  //fix_turn_l_90_line(right,left);
}


void advance_before_tll(long timee)
{
  Serial.println("FUNCTION  advance_before_tll");
  long TimeMillis = 0;

    long time = millis();
    while(TimeMillis - time < timee) //700)
    {                 //Time to turn towards bin
      Serial.println("  advance_before_tll:  ");
      Serial.println(TimeMillis - time);
      get_current_status();
      go_straight();
      advance(speed_R,speed_L);
      TimeMillis = millis();
    }

  }

void aligned_bin(int time1, long correct_time)
{

    int r = speed_R;
    int l = speed_L;
    speed_R = speed_R - alignement_factor;
    speed_L = speed_L - alignement_factor;
    //int correct_time = 200;
    //int time1 = millis();
    Serial.println("==========alignee=======");
     read_ir();

    int exit = 0;
    long TimeMillis = 0;
```

```
int invert = 0;
int forward = 0;
while (exit == 0)
{

   Serial.print(front_left_sensor_state);
   Serial.print("\t");
   Serial.print(front_right_sensor_state);
   Serial.print("\t");
   Serial.println(TimeMillis);

   if (TimeMillis - time1 < correct_time)
    invert =1;

   read_ir();

   //L(B) M(?) R(W)
   if (front_left_sensor_state == BLACK && front_right_sensor_state == WHITE )
   {
    Serial.println("front_left_sensor_state == BLACK && front_right_sensor_state == WHITE");
    //back_off(0, turn_speed);
    if (invert == 0)
    {
       advance(turn_speed+7, 0);
     Serial.println("advance(0, turn_speed");
    }
    //advance(0, turn_speed -7);
    else
    {
     back_off(turn_speed+7, 0); // worked
     Serial.println(" back_off(0,turn_speed);");
    }


   }

   else if (front_left_sensor_state == WHITE && front_right_sensor_state == BLACK )
```

```
{
  Serial.print(" front_left_sensor_state == WHITE && front_right_sensor_state == BLACK left correction");
  //back_off(turn_speed, 0);
  if (invert == 0)
  {
    advance(0, turn_speed+7); // worked = turn_l_line
     Serial.println(" advance(turn_speed, 0);");
  }
  //advance  (0,turn_speed -7);
  else
  {
    back_off(0,turn_speed+7); // worked
    Serial.println("back_off(turn_speed, 0);");
  }

}
else if(front_left_sensor_state == BLACK && front_right_sensor_state == BLACK)
{

 /*
 if (invert == 1)
 {
  back_off (turn_speed, turn_speed);
   Serial.print("bCK");
 }
 else
 {*/
    advance(speed_R,speed_L);
    Serial.print("advance");
 //  }
 }
else
{
  Serial.println("aligned");
  stopp();
  delay(200);
    exit = 1;
```

```
      }
      TimeMillis = millis();
   }
   speed_R = r;
   speed_L = l;
}

void aligned_bin_right(long time1, long correct_time)
{
  Serial.println("FUNCTION aligned_bin_right");
   //int correct_time = 200;
   //int time1 = millis();
   Serial.println("==========aligned_bin_right=======");
    read_ir();

   int exit = 0;
   long TimeMillis = 0;
   int invert = 0;
   int forward = 0;
   while (exit == 0)
   {
       Serial.print("forward ");
          Serial.println(forward);
      Serial.print(front_left_sensor_state);
      Serial.print("\t");
      Serial.print(front_right_sensor_state);
      Serial.print("\t");
      Serial.print("front_left_sensor_state ");
      Serial.println(front_left_sensor_state);

      //Serial.println(TimeMillis);

      if (TimeMillis - time1 < correct_time)
        invert =1;

      Serial.print("Invert ");
      Serial.println(invert);
```

```
read_ir();

//L(B) M(?) R(W)
if (front_left_sensor_state == BLACK && front_right_sensor_state == WHITE )
{
  //Serial.println("front_left_sensor_state == BLACK && front_right_sensor_state == WHITE");
  //back_off(0, turn_speed);
  if (invert == 0)
  {
     //advance(turn_speed+7, 0);
     advance(0, turn_speed+15);
    Serial.println("advance(0, turn_speed+7)");
  }
  //advance(0, turn_speed -7);
  else
  {
    //back_off(turn_speed+7, 0); // worked
    back_off(0, turn_speed+10);
    Serial.println(" back_off(0, turn_speed+7);");
  }


}

else if (front_left_sensor_state == WHITE && front_right_sensor_state == BLACK )
{
  //Serial.print(" front_left_sensor_state == WHITE && front_right_sensor_state == BLACK left correction");
  //back_off(turn_speed, 0);
  if (invert == 0)
  {
    //advance(0, turn_speed+7); // worked = turn_l_line
    advance(turn_speed+10, 0);
     Serial.println(" advance(turn_speed+7, 0)");
  }
  //advance  (0,turn_speed -7);
  else
```

```
        {
          //back_off(0,turn_speed+7); // worked
          back_off(turn_speed+10, 0) ;
          Serial.println("back_off(turn_speed+7, 0) ;");
        }

      }
      else if(front_left_sensor_state == BLACK && front_right_sensor_state == BLACK)
      {



        if (invert == 0 )
        {
          back_off (turn_speed, turn_speed);
          Serial.println("Invert == 1back_off (turn_speed, turn_speed);");
        }
        else
        {
          advance(speed_R,speed_L);
        }
       }
      else
      {
        Serial.println("aligned");
        stopp();
          exit = 1;
      }
      TimeMillis = millis();
   }
}

void line_following(int lines, int dir)
{
  while(white_lines != lines)
  {
    read_ir();
```

```
// LEFT
//      | | |
//       L M  R
//      | | |
// <- L
// L(W) M(W) R(B)
if(left_sensor_state == WHITE      && middle_sensor_state == WHITE    && right_sensor_state == BLACK )
{
  Serial.println("LINE FOLLOWING  turning left L(W) M(W) R(B) ");
   advance(speed_R, speed_L+ plus_factor);
   //advance(speed_R + plus_factor , speed_L);
   Serial.println("advance(speed_R, speed_L+ plus_factor);");

}
else
{  // opening else - right eft - left - stop - forward
  read_ir();
  // RIGHT
  //      | | |
  //       R
  //     M  | |
  //   L  | | |
  //-> R
  // L(B) M(B) R(W)
if (left_sensor_state == BLACK       && middle_sensor_state == BLACK    &&  right_sensor_state == WHITE )
{
  Serial.println("LINE FOLLOWING  going right L(B) M(B) R(W)");
   advance(speed_R + plus_factor , speed_L);
   //advance(speed_R , speed_L + plus_factor);
   Serial.println("advance(speed_R + plus_factor, speed_L);");
}
else
{// opening else - left - left - stop - forward
  read_ir();
  // RIGHT
  //     ||||
  //   L  M R
```

```
//     ||||
// L(B) M(W) R(W)
// ->R
  if(left_sensor_state == BLACK     && middle_sensor_state == WHITE    && right_sensor_state == WHITE )
{// opening if - go right
  Serial.println("LINE FOLLOWING going left L(B) M(W) R(W)");
  //advance(speed_R +8, speed_L );

 // advance(speed_R , speed_L + plus_factor );
    advance(speed_R + plus_factor, speed_L);



  //advance(speed_R  + plus_factor , speed_L);
  Serial.println(" advance(speed_R , speed_L + plus_factor);");



}// close if - go left
else
{ // opening else - left - stop - forward
  read_ir();
  // LEFT
  // L(W) M(B) R(B)
  //      | | |
  //        L
  //      | |  M
  //      | |    R
  // <-
  if (left_sensor_state == WHITE && middle_sensor_state == BLACK && right_sensor_state == BLACK  )
  { // opening if - going left
    Serial.println("LINE FOLLOWING turning right L(W) M(B) R(B) ");
     advance(speed_R , speed_L + plus_factor);
     //advance(speed_R + plus_factor, speed_L );
     Serial.println("advance(speed_R , speed_L + plus_factor);");

  } // closing if - going left
  else
  { // opening else - stop & forward
```

```
read_ir();
// STOP
if(left_sensor_state == WHITE && middle_sensor_state == WHITE && right_sensor_state == WHITE)
{ // opening if - stop
  if (white_lines == lines)
  { // opening if - stop
     Serial.println("LINE FOLLOWING stop        =     =     =     =     =     =     =     =     =     =");
     stopp();

     //state = TURN_TO_BIN;
  } // closing if - stop
  else
  { // opening else - pass white line
    while(left_sensor_state == WHITE && middle_sensor_state == WHITE && right_sensor_state == WHITE)
    { // opening while loop - pass white line
        read_ir();
       Serial.print("|||||||||||||||| PASSING WHITE||||||||||||||||||  ");
       Serial.println(white_lines);
       get_current_status();     // get velocity

       advance(speed_R,speed_L); // move forward
       go_straight();           // correct velocity
        e_turn =0;
    } // closing while loop - pass white line
    white_lines++;
   // state++;
  } // close else - pass white line
} // closing if - stop
else
{ //opening else - froward
  read_ir();
  // FORWARD
  if(left_sensor_state == BLACK && middle_sensor_state == WHITE && right_sensor_state == BLACK)  // ASK
ISA
  { // opening if - forward
    Serial.println("LINE FOLLOWING: going forward");
    get_current_status();            // get velocity
```

```
            // Serial.println(speed_L);
            // Serial.print("\t");
            // Serial.print(speed_R);
             advance(speed_R,speed_L);        // advance
             go_straight();                   // correct velocity
                e_turn =0;
           } // closing if - forward
         } //closing else - froward
       } // closing else - stop & forward
     } // closing else - left - stop - forward
    }// closing else - left - left - stop - forward
  }// closing else - right eft - left - stop - forward
 }

 if (dir == plus_dir)
   pos = pos + white_lines;
 else if (dir == minus_dir)
   pos = pos - white_lines;

 white_lines = 0;
}

void back_to_line_front()
{
  Serial.println("BACK TO LINE FRONT");

  int r = speed_R;
  int l = speed_L;
  speed_R = speed_R -15;
  speed_L = speed_L - 7;
  int l1_r2= 0;
  int r1_l2 = 0;
   read_ir();

  long TimeMillis = 0;
   int correct = 0;
  long time1 = millis();
```

```
 while((front_left_sensor_state == BLACK || front_right_sensor_state == BLACK)  &&  correct == 0 )
{ //Time to turn backwards bin

  /*if (front_left_sensor_state == WHITE )
  {
     Serial.print("l1_r2 ");
       Serial.println(l1_r2);
     l1_r2 = 1;
  }
  if(front_right_sensor_state == WHITE )
  {
    Serial.print("r1_l2 ");
       Serial.println(r1_l2);
     r1_l2 = 1;
  }



  if (l1_r2 == 1 && front_right_sensor_state == WHITE)
  {
    l1_r2 = 2;
    Serial.print("l1_r2 ");
       Serial.println(l1_r2);
   }

      if (r1_l2 == 1 && front_left_sensor_state == WHITE)
  {
    Serial.print("r1_l2 ");
       Serial.println(r1_l2);
     r1_l2 = 2;
   }*/
   if ( TimeMillis - time1 < 1300)
   {
      get_current_status();
       back_off(speed_R,speed_L);
       go_straight();
```

```
        read_ir();
        TimeMillis = millis();

        Serial.print("SPEED_back_to_line_front");
        Serial.println(TimeMillis - time1);
        Serial.println(" back");

      }
      else
      {
       get_current_status();
        advance(speed_R,speed_L);
        go_straight();
        read_ir();
        TimeMillis = millis();

        Serial.println("SPEED_back_to_line_front ");
        Serial.println(TimeMillis - time1);
        Serial.println(" advance");

        if (front_left_sensor_state == BLACK || front_right_sensor_state == BLACK)
          correct = 1;
      }




  }
  get_current_status();
 // back_off(speed_R,speed_L);
  go_straight();
  speed_R = r;
  speed_L= l;

}

void turn_left_bin(int turntime)
```

```
{
    turn_90_l(speed_R,speed_R, millis(), turntime);
    delay(1000);
    read_ir();

     // back_to_line_front();
     back_before_turn(9);
    aligned_bin(millis(),1000);
}

void turn_right_line(long max)
{
   analogWrite (R_ME,speed_R);
  analogWrite (L_ME,speed_L);
  long TimeMillis = 0;
  long correct = 0;
 long time1 = millis();
 int i = 0;
  while(i < 18)
   {                //Time to turn towards bin
    Serial.println("turn_right_line ----------------------Time to turn:  ");
    Serial.println(TimeMillis - time1);
    digitalWrite(R_M1,R_Backward);
    digitalWrite(L_M1,L_Forward);

    i++;
  }
    analogWrite (R_ME,0);
    analogWrite (L_ME, 0);
}

void turn_left_line( int turntime)
{
  Serial.println("FUNCTION turn_left_line");
  analogWrite (R_ME,speed_R+10);
  analogWrite (L_ME,speed_L);
  unsigned int TimeMillis = 0;
```

```
 unsigned int time1 = millis();
 unsigned int correct = time1;
 long i = 0;
  while(i < turntime)

   {                //Time to turn towards bin
    Serial.println("turn_left_line ----------Time to turn:  ");
    Serial.println(TimeMillis - time1);
    digitalWrite(R_M1,R_Forward);
    digitalWrite(L_M1,L_Backward);
    //TimeMillis = millis();

    i++;
  }

/*
  if(correct == 2)
  {
    read_ir();
    while (front_left_sensor_state == BLACK)
    {
      digitalWrite(R_M1,R_Backward);
       digitalWrite(L_M1,L_Forward);
       read_ir();
     }
     analogWrite (R_ME,0);
    analogWrite (L_ME, 0);

   }
   */
    analogWrite (R_ME,0);
    analogWrite (L_ME, 0);
}

void back_to_line()
{
```

```
  Serial.println("BACK TO LINE");

   int r = speed_R;
   int l = speed_L;
   speed_R = speed_R - 9;
   speed_L = speed_L - 6;

    read_ir();
     while(left_sensor_state == BLACK || right_sensor_state == BLACK)
    { //Time to turn backwards bin
      get_current_status();
      back_off(speed_R,speed_L);
      go_straight();
      read_ir();

      Serial.print("SPEED_back_to_line ");
      Serial.print(speed_L);
      Serial.print("\t");
      Serial.println(speed_R);
    }


    /*while(left_sensor_state != WHITE && middle_sensor_state != WHITE && right_sensor_state != WHITE )
    { //Time to turn backwards bin
      get_current_status();
      read_ir();
      back_off(speed_L,speed_R);
      go_straight();
     // read_ir();
    }*/
    speed_R =r;
    speed_L = l;
}

void turn_right_bin(int turntime)
{
    Serial.print("turn_right_bin(");
```

```
    turn_90_r(speed_R,speed_R, millis(), turntime);
    delay(100);

      back_before_turn(9);
      delay(300);

   //aligned_bin_right(millis(),1000);
   aligned_bin(millis(),1000);
}

void back_before_turn(long timee)
{
  Serial.println("FUNCTION Back before turn");
  print_state();
  if (state != test && state != OPPOSITE_R ){
  long TimeMillis = 0;
    int t = 0;
    long time = millis();
    while(t < timee) //700)
    {                 //Time to turn towards bin
      Serial.println(" back_before_turn() ----------backwardsn:  ");
      Serial.println(TimeMillis - time);

      back_off(speed_R,speed_L);
      get_current_status();
      go_straight();

     // TimeMillis = millis();
      t++;
    }
  }
  else
  {
    turn_180_l_bin(2200);
    }
}
```

```
void turn_180_r_bin( long turntime)
{
  Serial.println("FUNCTION turn_180_r_bin");
  print_state();

  int right = speed_R;
  int left = speed_R;

  analogWrite (R_ME,right);
  analogWrite (L_ME,left);
   long TimeMillis = 0;
   long time2 = micros();
  Serial.println(TimeMillis);
  Serial.println(time2);
  Serial.println(turntime);
  Serial.println(TimeMillis - time2);

  while(TimeMillis - time2 < turntime)
   {                //Time to turn towards bin
    Serial.println("--------------------Time to turn:  ");
    Serial.println(TimeMillis - time2);
    digitalWrite(R_M1,R_Forward);
    digitalWrite(L_M1,L_Backward);
    TimeMillis = micros();
  }
  analogWrite (R_ME,0);
  analogWrite (L_ME, 0);

}




void turn_180_l_bin(long turntime)
{
  Serial.println("FUNCTION turn_180_l_bin");
  print_state();
```

```
  int right = speed_R;
 int left = speed_R;
 analogWrite (R_ME,right);
 analogWrite (L_ME,left);
 long TimeMillis = 0;
 long time2 = millis();

 while(TimeMillis - time2 < turntime)
  {                //Time to turn towards bin
   Serial.println("----------------------Time to turn:  ");
   Serial.println(TimeMillis - time2);
   digitalWrite(R_M1,R_Backward);
   digitalWrite(L_M1,L_Forward);
   TimeMillis = millis();
 }
 analogWrite (R_ME,0);
 analogWrite (L_ME, 0);


}

void go_to_bin()
{
 Serial.println("GO TO BIN FUNCTION");
 int exit = 0;
  int r = speed_R;
  int l = speed_L;
  speed_R = speed_R - go_to_factor;
  speed_L = speed_L - go_to_factor ;



 /*
   int TimeMillis = 0;
   int correct = 0;
  int time1 = millis();
   while(TimeMillis - time1 < 1500)
```

```
  {               //Time to turn towards bin
   Serial.println("Arbitrary time going forward to avoid line detection:  ");
   Serial.println(TimeMillis - time1);
   advance(speed_R, speed_L);
   TimeMillis = millis();
   read_ir();
  }
 */

  while (exit == 0)
  {
     Serial.print("front_right_sensor_state ");
     Serial.println(front_right_sensor_state);
     read_ir();
     if (
       ((left_arm_sensor_state == WHITE && right_arm_sensor_state == WHITE ) && (low_right_sensor_state ==
WHITE  ) || (low_right_sensor_state == WHITE  ))
        || (low_right_sensor_state == WHITE  ) || (low_right_sensor_state == WHITE  ))
     {
      Serial.print(left_arm_sensor_state);
       Serial.print("\t");
      Serial.print(right_arm_sensor_state);
       Serial.print("\t");
      Serial.print(low_left_sensor_state);
       Serial.print("\t");
      Serial.println(low_right_sensor_state);


      stopp();
      exit = 1;
     }
     else if (left_arm_sensor_state == WHITE && right_arm_sensor_state == BLACK)
     {
      Serial.print("left arm");
      turn_R_line(turn_speed+5);
     }
     else if(left_arm_sensor_state == BLACK && right_arm_sensor_state == WHITE)
     {
```

```
        turn_L_line(turn_speed+6);

      }
      else
      {
       Serial.println("Forward");
       get_current_status();
       go_straight();
       advance(speed_R,speed_L); // move forward
      }
   }


   speed_R = r;
   speed_L = l;
 }

void back_off_time(long time_t)
{
  Serial.println("FUNCTION back_off_time");
 long TimeMillis = 0;
long time1 = micros();
 while(TimeMillis - time1 < time_t)
  {
    Serial.print("Timemillis ");
   Serial.println(TimeMillis);
   Serial.print("Time1 ");
   Serial.println(time1);
    Serial.println(TimeMillis - time1 );
    Serial.println("back off to turn");
    back_off(speed_L-15,speed_R-15);
   TimeMillis = micros();

 }
}

int d2s(long block, int dir)
```

```
{ // opening d2s
  //This function converts the block height to a number of
  // steps for the stepper to take.
  //  block is the desired travel distance for the claw
  //  dir is the desired direction of travel; 1 to move up
  // and -1 to move down.
  int steps = 0;
  steps = -dir*block*19.2*200/8;
  pos_claw += block*dir;
     Serial.print(pos_claw);
  return steps;
}
 // closing d2s

void firstStack()
{ // opening firstStack
  servo_bot.write(bclose);
  servo_top.write(tclose);
  delay(500);
  servo_top.write(90);
  servo_bot.write(bopen);
  myStepper.step(d2s(3,up));
  stack+=1;
}
  // closing firstStack

void lastStack()
{ // opening lastStack
  servo_bot.write(bclose);
  servo_top.write(tclose);
  delay(500);
  myStepper.step(d2s(stack,down));
}
  // closing lastStack

void bstack()
{ // opening bstack
```

```
stack+=1;
Serial.print("\n Stacking");
//  Close bottom claw on block
  servo_bot.write(bclose);
  delay(1000);

// Actively close on block
Serial.print("\n Closing Top");
servo_top.write(tclose);
delay(500);

// Lower Stepper to place stack on block
Serial.print("\n Moving Down");
myStepper.step(d2s(pos_claw-stack+1.4,down));
delay(1500);

//open top claw
Serial.print("\n Opening Claw");
servo_top.write(topen);
delay(1500);
servo_top.write(90);
delay(500);

// Move top Claw above stack
Serial.print("\n Moving Up");
myStepper.step(d2s(stack+1-pos_claw,up));
delay(1500);

// Close top claw
Serial.print("\n Closing Claw");
servo_top.write(tclose);
delay(500);
servo_top.write(90);

// push down stack from top
Serial.print("\n Moving Down");
myStepper.step(d2s(pos_claw-stack,down));
```

```
  delay(1500);
  myStepper.step(d2s(stack/2,up));

  // open top claw
  Serial.print("\n Open Claw");
  servo_top.write(topen);
  delay(1000);
  servo_top.write(90);

  // Reposition top claw to above cg of stack
  Serial.print("\n Moving Down");
   myStepper.step(d2s(pos_claw-(stack/2),down));
  delay(1500);

  // Close top claw
  Serial.print("\n Closing Claw");
  servo_top.write(tclose);
  delay(500);
  servo_top.write(90);

  // open bottom claw
  servo_bot.write(bclose);

  // Raise stack above bottom claw
  Serial.print("\n Moving Up");
  myStepper.step(d2s(pos_claw+1,up));
  delay(1500);
}
```

Operation instructions

1. Alter code to desired performance requirements.

2. Upload code to Arduino Mega 2560.
3. Ensure all wires are properly connected using the wiring diagram.
4. Ensure batteries are fully charged and connected.
5. Place Ezio in the white starting square of the playfield.
6. Flip the switch to ensure power is running through entire system.
7. Cease interaction until the product completes its task.
8. If Ezio performs unexpected actions, flip the switch to stop motion and proceed to troubleshooting.

Maintenance

After each run the ball caster should be cleaned of any debris that may have accumulated. The sensors should be inspected before each run to ensure there are no obstructions. The Arduino should be reset periodically to clear the memory. Check wires and connections periodically for signs of burning or other wear.

Troubleshooting

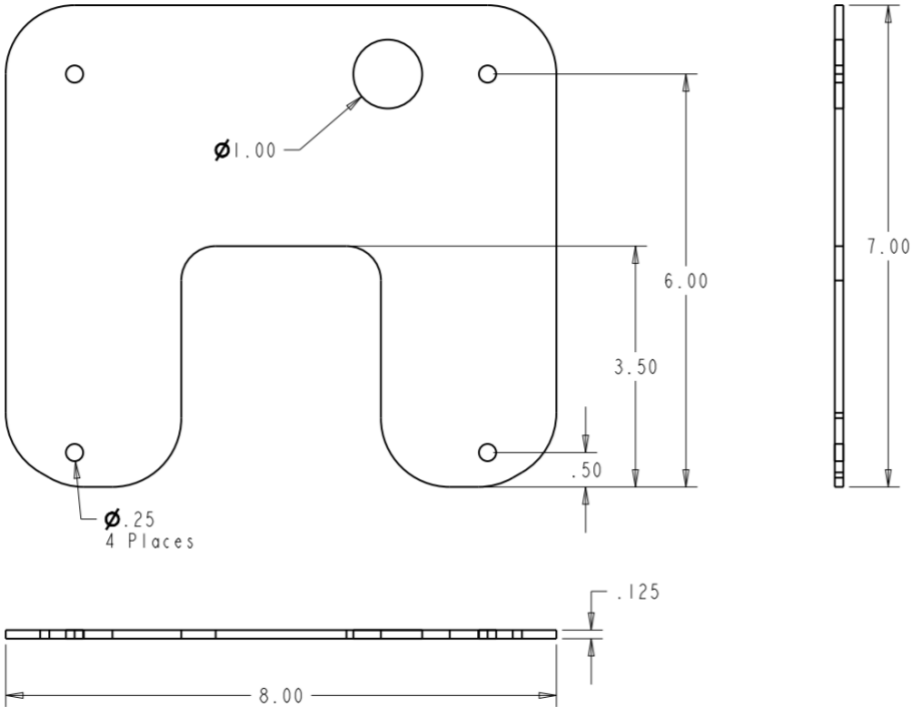
Problem	Possible Solution
No movement occurs after flipping the switch	Ensure 12V and 9V batteries are fully charged and properly connected.
	Ensure Arduino and motor controllers are properly grounded.
Upper/Lower claw does not open/close properly	Check code for possible mistakes.
	Ensure gear teeth are properly meshed.
Robot is not line following properly/ veers off given path	Ensure IR sensors are properly connected. Indicator lights should illuminate when the sensor senses a change in color.
	Ensure IR sensors are parallel with the floor.
Buzzing sound coming from either of the two DC motors	Increase the speed of motors in the code
Stepper Motor with lead screw doesn't turn despite correct connections	Flip 1A and 1B wires going to stepper motor connections
No Robot movement and IR sensors are off after flipping the power switch	Ensure that no metal parts are touching IR sensor connections.

	Check if the buck converter light is lit. If light is off, ensure that the battery is plugged. If buck converter light is not lit after plugging battery, change the buck converter.
Robot doesn't go towards the bins after turning from the center line	Check that one of the bottom IR sensors is not stuck in the Active state.
There is smoke or burning components coming from the robot	Safely unplug the batteries and check connections according to the wiring diagram.
Arduino doesn't successfully upload new code	Disconnect the batteries or any power connections and try to upload the code again.
The DC motor battery is connected but the power switch remains off when switched.	Check the battery fuse holder, and replace any blown fuses. Refer to the wiring diagram for proper connections.

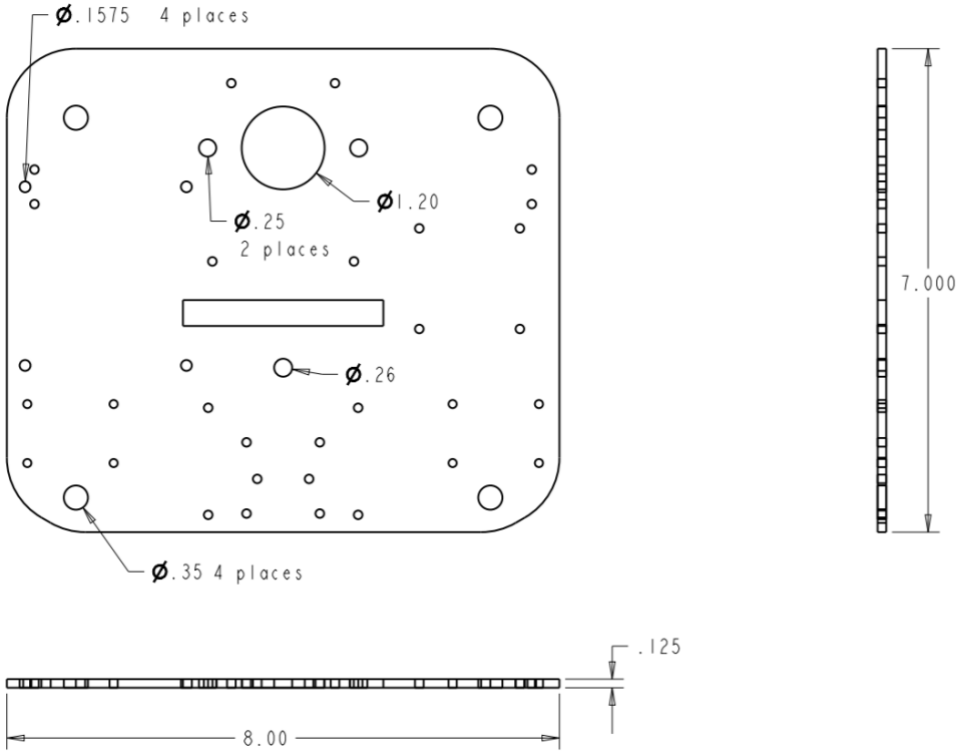
Drawings

Note: All drawing dimensions are in US units, except for the gears on the claws, which are metric. All holes are 3mm, or 1/8" unless otherwise specified.

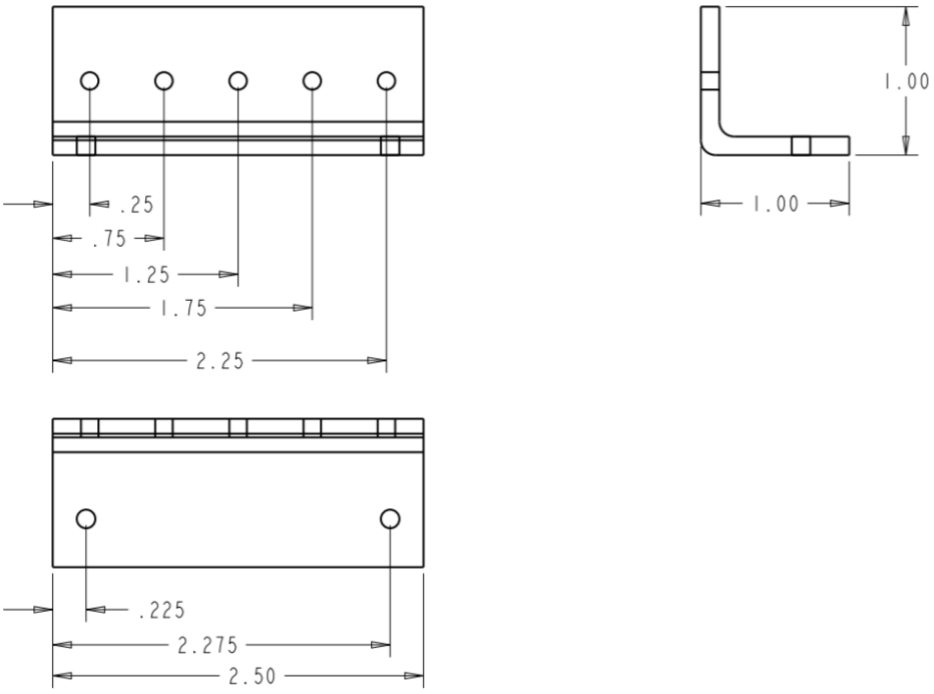
Upper Base



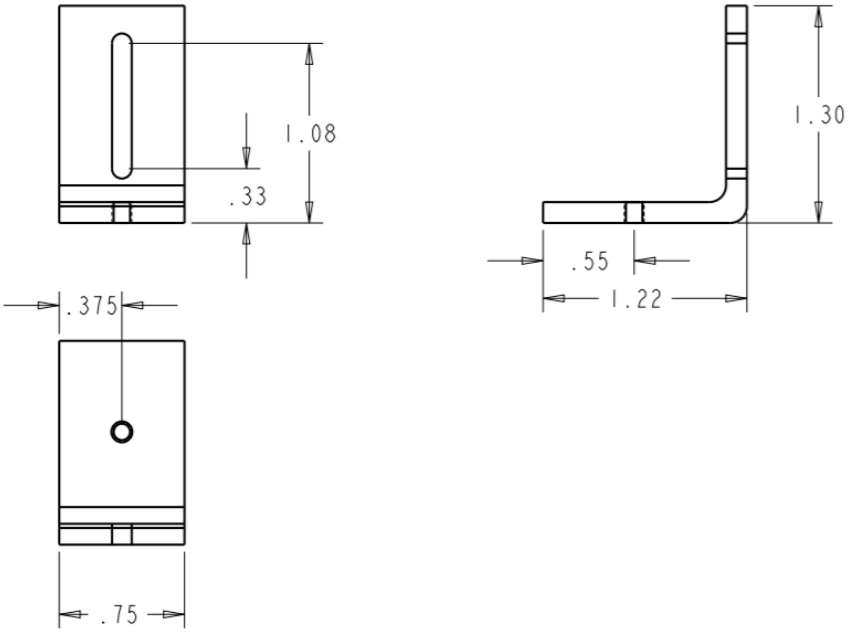
Lower Base



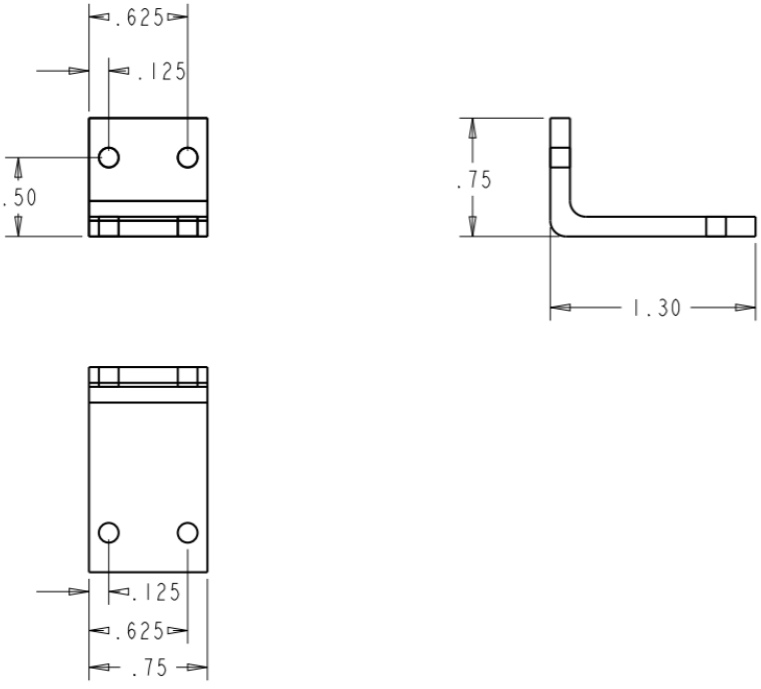
3-IR L-bracket



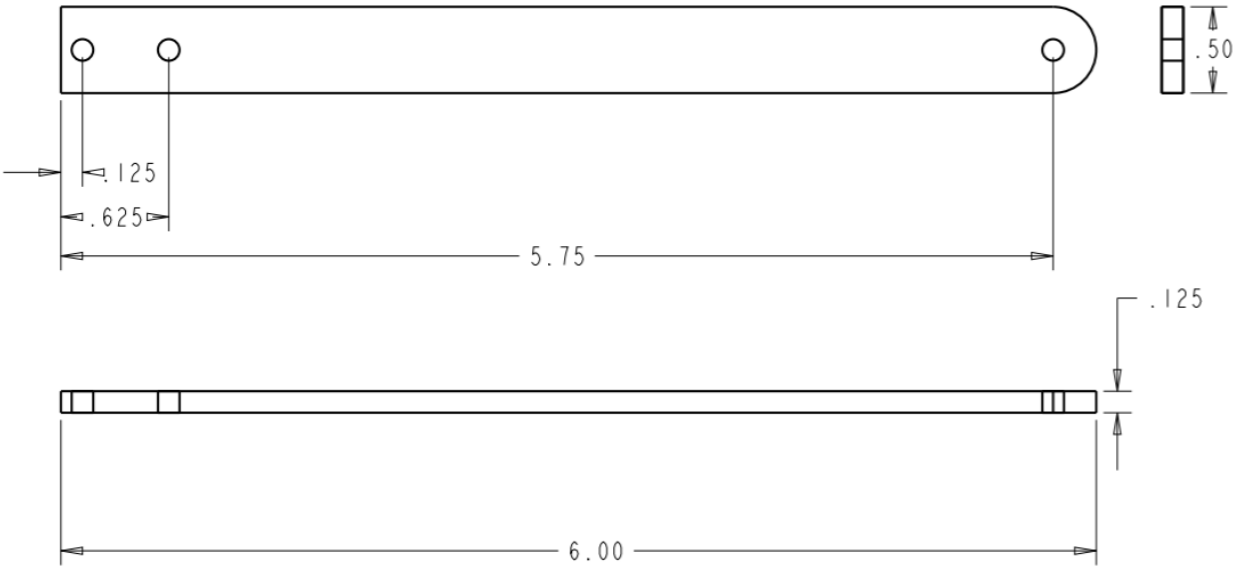
IR L-bracket



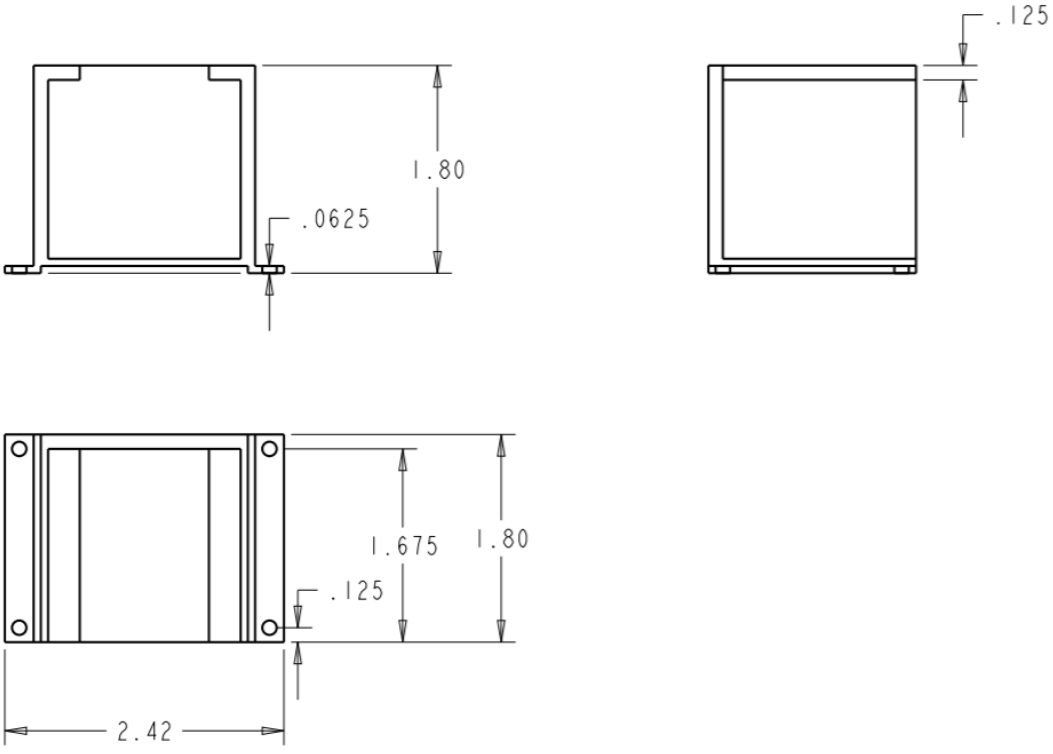
Side-IR L-bracket



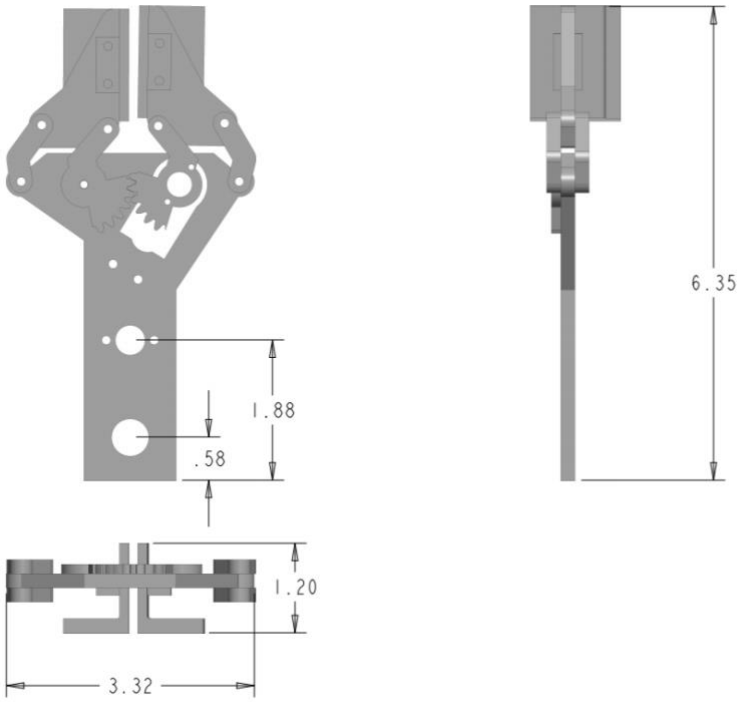
Side-IR Arm



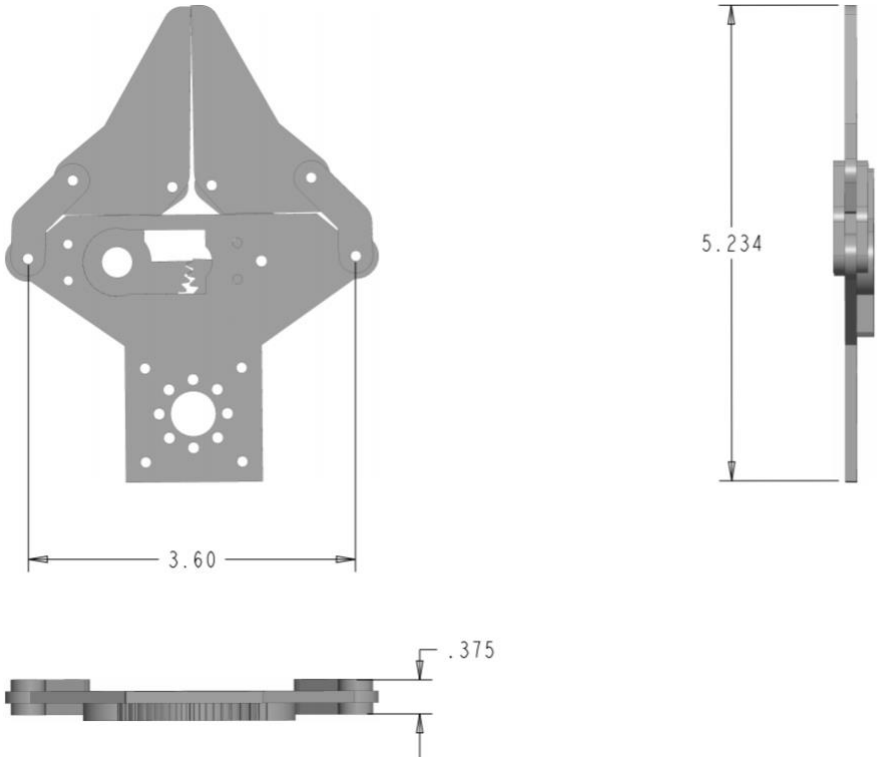
Stepper Mount



Upper Claw



Lower Claw



Endzone Base