RaspberryPi + Parallax Propeller

Yash Patel (yp2285)
Anais Roche (aer523)
Rushi Shah (rs7236)

Hardware Setup

Two continuous servo motors are used for maneuvering the Boe Bot rover. Values at 1400 and 1390 indicate no rotation for the right and left wheel respectively, incrementing will cause the motor to rotate in one direction and decrementing will cause the motor to rotate in the opposite direction. A standard servo motor is used to rotate the PING sensor where the input value is the degree of rotation scaled by 10. (0 is 0, 90 is 900, and 180 is 1800). Another standard servo motor is used to rotate the device that will strike enemies detected. This servomotor is mounted on the back of the Boe Bot. The servo motors are all connected via pin 12, 13, 14, and 15 at the top right corner of the Propeller microcontroller respectively.

One PING ultrasonic sensor is mounted on the standard servo at the front of the bot and is initialized to look for objects 35 centimeters when detecting obstacles. There is a 5 centimeter margin from the actual distance to ensure the robot detects the object. The second ultrasonic sensor is a HC-SR04 sensor - which is mounted on the left side of the Boe Bot rover. This sensor is responsible for detecting objects 10 cm to ensure the Boe Bot comes to a complete stop for the Camera module on the Raspberry pi to complete its task. It is attached to the Raspberry Pi.

QTR-HD-15A Reflectance Sensor Array is used for line detection. It is an array of 15 IR LED and phototransistor pairs and four of the 15 pairs are selected to be used to detect lines. Values above 1 indicate a black line is detected while values smaller than 1 indicate no line (on a white background). The servo actuation is determined based on the readings of the two inner sensors while the outer two sensors determine intersection detection.

The Parallax Propeller is mounted on a base plate attached to the top of the Boe Bot. It has three LEDs- Red, Yellow, and Green - attached with resistors to pins 1, 2, and 3. The PING sensor also has a resistor connected to the signal pin to ensure the Propeller chip is safe.

The Raspberry Pi Model 3 is mounted on an elevated platform above the Parallax Propeller Microcontroller. Its ground is connected to the ground of the Propeller and its GPIO7 pin is connected to pin 11 of the Propeller for communication between the two microcontrollers. The camera module is connected to the Raspberry Pi via the Camera Module Port.

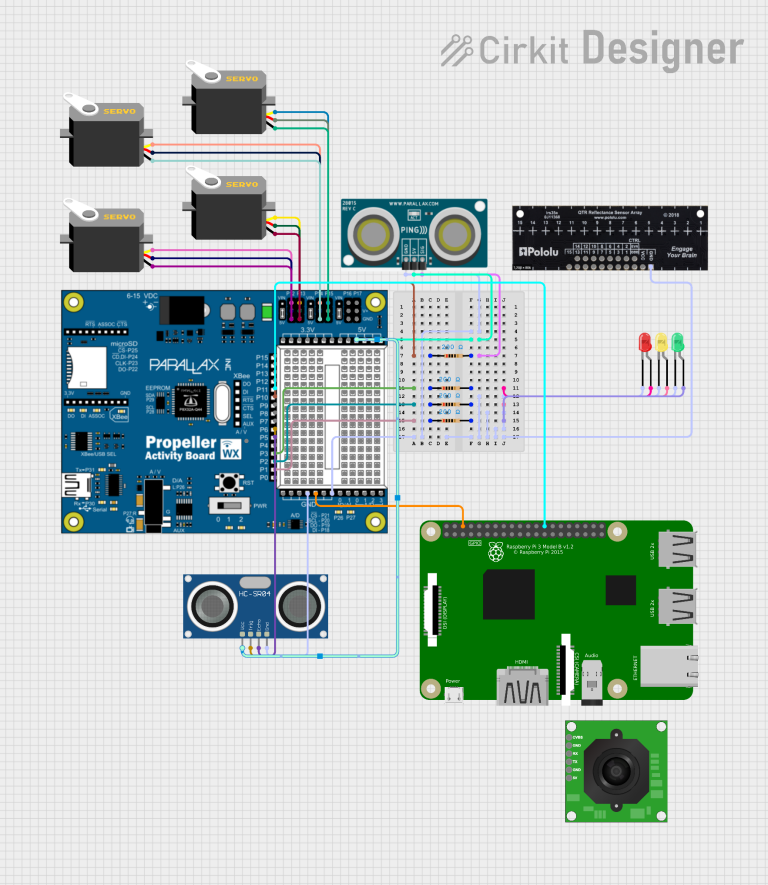

Figure 1. Circuit diagram of the Parallax Propeller Activity Board and the Raspberry Pi Model 3. The connection between the camera module and the raspberryPi is not shown due to the limitations of the Cirkit Designer software. The Propeller microcontroller has two ultrasonic sensors, three LEDs, a Pololu QTR sensor and four servo motors connected. The propeller is powered by a 9V battery pack. The Raspberry pi has the camera module attached to it and is powered by an external power source attached via the micro usb port.

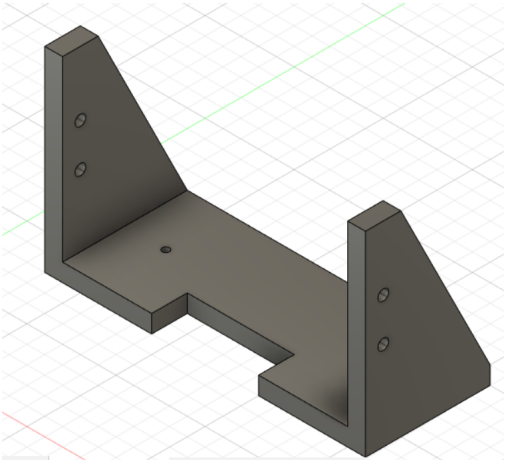

Figure 2. This is the second version of the QTR sensor mount.

Implementation

Parallax Propeller

Line tracking using QTR sensors

The read_line_sensor() function initializes the four pins and assigns the values to a vector array v. For efficiency, only 4 sensors are used to detect and follow the line instead of the full array of sensors. This vector v is returned at the end of the function. When the inner two sensors detect values greater than 1, indicating the black line, the bot will move forward. If the inner left sensor detects a value greater than 2, but the inner right sensor detects a value less than one, the bot will make a corrective motion to the right. If the inner right function detects a value greater than 2 and the inner left sensor detects a value less than one, the bot will make a corrective motion to the left. Only when all four sensors detect values greater than 2 will an intersection counter be incremented and the cog that monitors this value will blink the red LED to indicate it has reached an intersection.

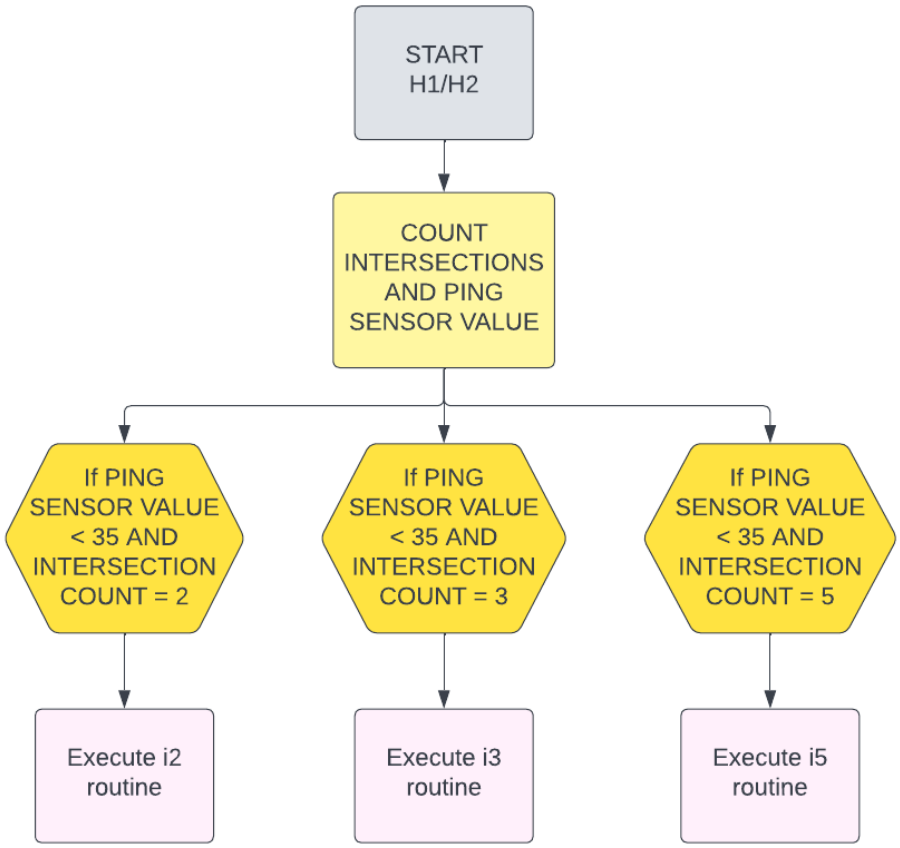

Figure 3. Navigation decision tree - Three routines are established based on the location of the obstacle. During this decision tree, the bot will also be checking for targets - in between intersections on the center lane, and at intersections for lanes A and B.

Navigation Routine

The Boe Bot starts at either H1 or H2 and tracks the curved black line until the first intersection or starting point, i0. At this stage, the bot only needs to track the line and identify this first intersection as a non active intersection (ie. not part of the grid). Without any further action, the bot can move forward to i1 where it begins the initial routine of checking for the obstacle location along the center lane.

There are friendlies and enemies planted arbitrarily along the center lane between intersections and the bot will use the ultrasonic sensor on its left to detect any targets and use the camera also mounted on the left to determine if the target is a friend or enemy. The bot will behave accordingly depending on the target's classification: hitting it if it's an enemy or lighting an LED if it's a friend. If an obstacle is detected the bot will first ensure that there are no targets on either side of the obstacle before making a 180 turn. At this point, the camera and ultrasonic sensor will be facing the other direction of the lane and begin to search for any targets on the other side that it missed earlier. The Boe-Bot will return to i1 to ensure it did not miss any targets along the center lane.

Once the Boe-Bot has cleared the center lane of any targets, it will take a right toward B1. At this stage of the search, the targets will now be located at intersections. Based on the location of the obstacle detected, the bot will do either the i2 routine, the i3 routine, or the i5 routine. To traverse from i1 to B1, the bot will complete a function used for single block movements, b1_countercheck(). If the obstacle was detected at i2, the i2 routine will be executed and the bot will run the function from B1 to B4, b1_b4_countercheck() as it will avoid turning on the second and third lane because of the presence of the obstacle and the direction of the road respectively. Then the bot will turn right on lane 4 and move towards the center lane. At this point the bot will search for targets on the center lane on the other side of the obstacle. When it returns to lane 4 it will continue its original loop and approach A4. At A4, the bot will turn right and repeat the same function that it used to run B1 to B4. If the bot has detected all targets before it reaches A1, it will stop. If not, the bot will make a full loop to B5, repeating the steps earlier from B1 to B4.

If the obstacle is on i3 and the i3 routine is run, it will follow the same path trajectory as i2. Except when it approaches lane 4 to search for targets in the center lane, it will only need to check lane 5 for any missing targets.

If the obstacle is on i5 and the i5 routine is run, it will follow the same path trajectory as i2 except, all center lane targets have already been detected at this stage and it will not need to check the center lane again when it traverses lane 4 and uses b4_a4_countercheck(). When it reaches A1, if all targets have been found it will stop. If there is still a remaining target, it will approach B1 and then to B5 using a function for five block movement, b1_b5_countercheck().

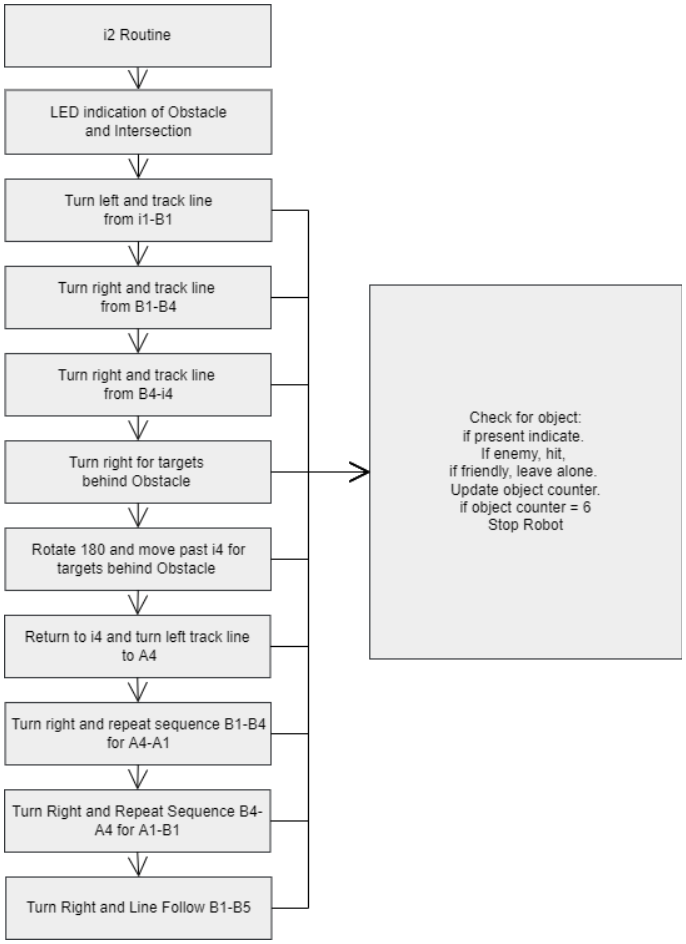

Figure 4. i2 Routine

Raspberry Pi

Camera

The camera module uses OpenCV for the ArUco libraries and the code for the Raspberry Pi is written in Python. It takes a video capture in grayscale. When the ID range is between 0, 10 a friend is detected and indicated and when the ID range is between 10 and 20 an enemy is detected and GPIO7 will be driven high to indicate to the propeller that an enemy is detected. After the propeller takes the appropriate action, the GPIO7 pin will be set to low again.

When the GPIO7 pin is set to high, a function detect_enemy() is run. To confirm the reading is not lagging and not reading a tag directly in front of it, the ultrasonic sensor is used to confirm that there is an object within 13cm. If an object is confirmed and it's been identified as an enemy, the back servo will lower to hit the object.

Appendix: Project Code

Propeller

```
/*
  Blank Simple Project.c
  http://learn.parallax.com/propeller-c-tutorials
*/
#include "simpletools.h"                      // Include simple tools
#include "abdrive.h"                          // abdrive library
#include "servo.h"
#include "ping.h"                             // Include ping header
#include "adcDCpropab.h"

float * read_line_sensor();
void  line_follow();
void detect_enemy();
void object_detection(void *par);
void intersection_led_blink(void *par);
void bot_forward(int speed_value);
void bot_back(int speed_value);
void navigate(void *par1);
void bot_stop();
void i2_routine();
void i3_routine();
void i5_routine();
void bot_180();
void b1_b4();
void b4_a4();
void b4_a4_for_i2();
void a1_b1();
void b1_b5();
int distance();
int object_detect();
volatile int count_intersect = 0;
volatile int count_object = 0;
int count_intersect_b1b4 = 0;
unsigned int stack1[40+25];
unsigned int stack2[40+25];
int object;
int status;
int cmDist;
float v[4];
//float v_l[4];
```

```
volatile int detection = 0;
volatile int intersect = 0;
volatile int dist;
int flag = 0;

int main()                                    // Main function
{

   cogstart((void*)navigate, NULL, stack2, sizeof(stack2));
   cogstart((void*)intersection_led_blink,NULL, stack1,sizeof(stack1));

}

// read line sensor values in terms of
voltage-------------------------------------------------
float * read_line_sensor()
{
      adc_init(21,20,19,18);
      //float v0,v1,v2,v3;
      v[0] = adc_volts(0);
      v[1] = adc_volts(1);
      v[2] = adc_volts(2);
      v[3] = adc_volts(3);
      //print("A/D2 = %f V%c\n", v0, CLREOL);     // Display volts
      //print("A/D3 = %f V%c\n", v1, CLREOL);
      //print("A/D3 = %f V%c\n", v2, CLREOL);
      //print("A/D3 = %f V%c\n", v3, CLREOL);
      return v;
}

//line
follower-----------------------------------------------------------------------
---------------
void  line_follow()
{
    //2  6 10 14
    float * v;
    v = read_line_sensor();
    //print("A/D3 = %f V%c\n", v[2], CLREOL);

    if (v[1] > 1 && v[2] > 1)
    {
```

```
        bot_forward(50);
    }
    else if (v[1]>2 && v[2]<1)
    {
       servo_set(12,1400); //slight_left
       servo_set(13,1370);
    }
    else if(v[2]>2 && v[1]<1)
    {
       servo_set(12,1430);  //slight_right
       servo_set(13,1400);

     }
}

int distance()
{
  low(6);
  pulse_out(6,10);
  volatile long techo = pulse_in(5,1);
  volatile int dist = techo / 58;
  return dist;
}



// robot forward
motion-------------------------------------------------------------------------
void bot_forward(int speed_value)
{
  int left_motor_speed, right_motor_speed;
  left_motor_speed = 1400 + speed_value;
  right_motor_speed = 1400 - speed_value;
  servo_set(12, left_motor_speed);
  servo_set(13,right_motor_speed);
}

// robot backwards motion
void bot_back(int speed_value)
{
  int left_motor_speed, right_motor_speed;
  left_motor_speed = 1400 - speed_value;
```

```
  right_motor_speed = 1400 + speed_value;
  servo_set(12, left_motor_speed);
  servo_set(13,right_motor_speed);
}

// turn the robot left
--------------------------------------------------------------------------------
void bot_left()
{
  float * v;
  v = read_line_sensor();
  while(v[2]>2)
  {
    float * v;
    v = read_line_sensor();
    //printf("%d \n", v[1]);
    servo_set(12,1400);
    servo_set(13,1350);
  }

  while(v[2]<2)
  {
    float * v;
    v = read_line_sensor();
    //printf("low= %d \n", v[1]);
    servo_set(12,1400);
    servo_set(13,1350);
  }
}

//turn the robot
right-------------------------------------------------------------------------
----------------
void bot_right()
{
  float * v;
  v = read_line_sensor();
  while(v[1]>2)
  {
    float * v;
    v = read_line_sensor();
    servo_set(12,1450);
```

```
    servo_set(13,1400);
  }
  while(v[1]<2)
  {
    float * v;
    v = read_line_sensor();
    servo_set(12,1450);
    servo_set(13,1400);
  }
}

// Turn the BOT by 180 degrees
void bot_180()
{
  float * v;
  v = read_line_sensor();
  while(v[3]<1)
  {
    float * v;
    v = read_line_sensor();
    servo_set(12,1430);
    servo_set(13,1430);
  }
  print("%f", cmDist);
  while(v[3]>1)
  {
    float * v;
    v = read_line_sensor();
    servo_set(12,1430);
    servo_set(13,1430);
  }
  print("%f", cmDist);

}

//stop the
robot---------------------------------------------------------------------------
---------------------------------
void bot_stop()
{
  servo_set(12,1400);
  servo_set(13,1400);
```

```
}

// Intersection LED Blink
Function--------------------------------------------------------------------
---
void intersection_led_blink(void *par)
{
   ////print("%d \n", intersect);

    while(1)
   {
        while(intersect)
        {
            high(2);
            pause(200);
            low(2);
            pause(200);
        }
        //intersect = 0;
    }

}


//obstacle LED blink
-------------------------------------------------------------------------------
--
void obstacle_led_blink()
{
    high(4);
    //pause(50);
}

void detect_enemy()
{
   set_direction(11,0);
   int distance = object_detect();
   if (distance < 13 && flag==0)
   {
     bot_stop();
     pause(500);
      if (input(11) == 1)
```

```
      {
       bot_back(30);
       pause(100);

       servo_angle(15,1800);
       flag = 1;

       //pause(2500);
       //bot_forward(50);
       //pause(200);

      }
    }
}

// Detecting object using Ultrasonic Sensor
int object_detect()
{
  low(6);
  pulse_out(6,10);
  volatile long tEcho = pulse_in(5,1);
  volatile int distance = tEcho/58;
  return distance;
  }


// B1 to B4 Routine
void b1_b4()
{
  bot_forward(30);
  pause(500);
  count_intersect = 0;
  while(1)
  {
  do{
    line_follow();
    detect_enemy();
   }while(v[0]<1 && v[3]<1);
  intersect = 1;
  flag = 0;
  servo_angle(15,900);
  count_intersect = count_intersect + 1;
```

```
  if (count_intersect == 3)
    {
      break;
    }
  bot_forward(30);
  pause(500);
  intersect = 0;
  }
}

//B4 to A4 via F4 and F3 for obstacle at i3
void b4_a4()
{
  do{
    line_follow();
    if (v[0]>2 && v[1]>2 && v[3]>2 && v[2]>2)
    {
      intersect = 1;
      break;
    }
    }while(1);
  bot_right();                ///Right from i4 towards i3
  intersect = 0;
  do{
    line_follow();
    detect_enemy();
    cmDist = ping_cm(10);
    }while(cmDist>4);
  flag = 0;
  servo_angle(15, 900);
  bot_180();
  do{
    line_follow();
    detect_enemy();
    }while(v[0]<1 && v[3]<1);    // Came back to i4
  intersect =1;
  flag = 0;
  servo_angle(15, 900);
  bot_forward(50);
  pause(500);
  intersect = 0;
  do{
```

```
    line_follow();
    detect_enemy();
    }while(v[0]<1 && v[3]<1);   // reached i5
  intersect = 1;
  flag=0;
  servo_angle(15, 900);
  bot_back(50);
  pause(600);
  bot_180();
  intersect = 0;
  do{
    line_follow();
    detect_enemy();
    }while(v[0]<1 && v[3]<1);   //reached i4 again
  intersect = 1;
  flag=0;
  servo_angle(15, 900);
  bot_left();
  do{
    line_follow();
    }while(v[0]<1 && v[3]<1);  // reached a4
  intersect = 1;
}



//B4 to A4 for obstacle at i2

void b4_a4_for_i2()
{
  do{
    line_follow();
    if (v[0]>2 && v[1]>2 && v[3]>2 && v[2]>2)
    {
      intersect = 1;
      break;
    }
    }while(1);
  bot_right();                ///Right from i4 towards i3
  intersect = 0;
  flag = 0;
  servo_angle(15,900);
```

```
do{
  line_follow();
  detect_enemy();
  }while(v[0]<1 && v[3]<1);   // beyond i4 towards i3
intersect = 1;
flag = 0;
servo_angle(15,900);
bot_forward(50);
pause(500);
intersect = 0;

do{
  line_follow();
  detect_enemy();
  cmDist = ping_cm(10);   //i3 to i2
  }while(cmDist>5);
flag = 0;
servo_angle(15,900);

bot_180();
do{
  line_follow();
  detect_enemy();
  }while(v[0]<1 && v[3]<1);   // Came back to i3
intersect = 1;
flag = 0;
servo_angle(15,900);

bot_forward(50);
pause(500);
intersect = 0;
do{
  line_follow();
  detect_enemy();
  }while(v[0]<1 && v[3]<1);   // reached i4
intersect = 1;
flag=0;
servo_angle(15, 900);
bot_forward(50);
pause(500);
intersect = 0;
```

```
  do{
    line_follow();
    detect_enemy();
    }while(v[0]<1 && v[3]<1);   // reached i5
  intersect = 1;
  flag = 0;
  servo_angle(15, 900);

  bot_back(50);
  pause(600);
  bot_180();
  intersect = 0;
  do{
    line_follow();
    detect_enemy();
    }while(v[0]<1 && v[3]<1);   //reached i4 again
  intersect = 1;
  flag=0;
  servo_angle(15, 900);
  bot_left();
  intersect = 0;
  do{
    line_follow();
    }while(v[0]<1 && v[3]<1);  // reached a4
  intersect = 1;
}


void a1_b1()
{
  count_intersect = 0;
  while(1)
  {
  do{
    line_follow();
   }while(v[0]<1 && v[3]<1);
  intersect = 1;
  count_intersect = count_intersect + 1;
  if (count_intersect == 2)
    {
      break;
```

```
    }
  bot_forward(50);
  pause(500);
  intersect = 0;
  }
}

void b1_b5()
{
  count_intersect = 0;
  flag=0;
  servo_angle(15, 900);
  while(1)
  {
  do{
    detect_enemy();
    line_follow();
   }while(v[0]<1 && v[3]<1);
  intersect = 1;
  flag = 0;
  servo_angle(15, 900);
  count_intersect = count_intersect + 1;
  if (count_intersect == 4)
    {
      break;
    }
  bot_forward(50);
  pause(750);
  intersect = 0;
  }
}

void navigate(void *par1)
{
  servo_angle(14,900);
  servo_angle(15,900);
  int cmDist = ping_cm(10);
  int dist = distance();
  float * v;
  v = read_line_sensor();
  while(1)
   {
```

```
    do{
      line_follow();
      float * v;
      v = read_line_sensor();
      detect_enemy();
      if (v[0]>1  &&  v[1]>1 &&  v[2]>1  && v[3]>1)
      {
          servo_angle(15,900);
          intersect = 1;
          flag = 0;
          break;
      }
      }while(1);
    //intersect = 0;
    int cmDist = ping_cm(10);
    count_intersect = count_intersect + 1;
    float * v;
    v = read_line_sensor();

    if (cmDist < 35 && v[0]>2 && v[3]>2)
    {
      obstacle_led_blink();
      intersect  = 1;
      break;
    }
    bot_forward(50);
    pause(500);
    intersect=0;
  }
intersect = 0;
if (count_intersect == 5)
{
  count_intersect = 0;
  i5_routine();
}
if (count_intersect == 3)
{
  count_intersect = 0;
  i3_routine();
}
if (count_intersect == 2)
{
```

```
    count_intersect = 0;
    i2_routine();
  }

}

void i3_routine()
{
  bot_forward(50);
  pause(500);
  flag = 0;
  servo_angle(15, 900);
  do{
    line_follow();
    detect_enemy();
    float * v;
    v = read_line_sensor();       ////////REPEAT
    cmDist = ping_cm(10);

    if (cmDist < 5)  //// check for obstacle
    {
      break;
    }
    }while(1);

  bot_180();                //////rotate 180 when obstacle at i3 intersection
  flag = 0;
  servo_angle(15, 900);
  do{
    line_follow();
    detect_enemy();
    }while(v[0]<1 && v[3]<1);
  intersect = 1;
  flag=0;
  servo_angle(15, 900);
  bot_forward(50);
  pause(500);
  //count_intersect = count_intersect + 1;
  do{
    line_follow();
    detect_enemy();
    }while(v[0]<1 && v[3]<1);
```

```
  intersect = 1;
  bot_right();                  //////Reached i1 intersection
  intersect = 0;
  flag = 0;
  servo_angle(15, 900);
  do{
    line_follow();
    }while(v[0]<1 && v[3]<1);
  intersect = 1;
  bot_right();                   /////Reached B1 and took right
  intersect = 0;
  detect_enemy();
  flag=0;
  servo_angle(15, 900);
  b1_b4();                       ////Reached B4
  bot_right();
  intersect = 0;
  b4_a4();
  bot_right();
  intersect = 0;
  b1_b4();
  bot_right();
  intersect = 0;
  a1_b1();
  bot_right();
  b1_b5();
  bot_stop();
}

void i5_routine()
{
  bot_forward(50);
  pause(500);

  do{
    line_follow();
    detect_enemy();
    float * v;
    v = read_line_sensor();       ////////REPEAT
    cmDist = ping_cm(10);

    if (cmDist < 4)  //// check for obstacle
```

```
  {
    flag = 0;
    servo_angle(15,900);
    break;
  }
  }while(1);

bot_180();
count_intersect = 0;
while(1)
{
do{
  line_follow();
  detect_enemy();
 }while(v[0]<1 && v[3]<1);
intersect = 1;
flag = 0;
servo_angle(15,900);
count_intersect = count_intersect + 1;
if (count_intersect == 4)  /// go till i1
  {
    break;
  }
bot_forward(50);
pause(500);
intersect = 0;
}

bot_right();
do{
  line_follow();
 }while(v[0]<1 && v[3]<1);
intersect = 1;
bot_right();
intersect = 0;
detect_enemy();
flag=0;
servo_angle(15, 900);
b1_b4();
bot_right();
intersect =0;
a1_b1();
```

```
  bot_right();
  b1_b4();
  bot_right();
  intersect = 0;
  a1_b1();
  bot_right();
  b1_b5();
  bot_stop();
  pause(5000);
}

void i2_routine()
{
  bot_forward(50);
  pause(500);
  flag = 0;
  servo_angle(15, 900);
  do{
    line_follow();
    detect_enemy();
    float * v;
    v = read_line_sensor();       ////////REPEAT
    cmDist = ping_cm(10);

    if (cmDist < 5)  //// check for obstacle
    {
      break;
    }
    }while(1);

  bot_180();                //////rotate 180 when obstacle at i2 intersection
  flag = 0;
  servo_angle(15, 900);
  do{
    line_follow();
    detect_enemy();
    }while(v[0]<1 && v[3]<1);
  flag=0;
  servo_angle(15, 900);

  //count_intersect = count_intersect + 1;
  bot_right();
```

```
  do{
    line_follow();
    }while(v[0]<1 && v[3]<1);
  bot_right();                    /////Reached B1 and took right
  detect_enemy();
  flag=0;
  servo_angle(15, 900);
  b1_b4();                        ////Reached B4
  bot_right();
  intersect = 0;
  b4_a4_for_i2();
  bot_right();
  intersect = 0;
  detect_enemy();
  flag = 0;
  servo_angle(15, 900);
  b1_b4();
  bot_right();
  intersect = 0;
  a1_b1();
  bot_right();
  b1_b5();
  bot_stop();
}
```

Raspberry Pi

```
#!/usr/bin/env python3

#Import only if not previously imported

import cv2

import RPi.GPIO as GPIO

from time import sleep

from ultrasonic import *



myPin=7
```

```
myPin2=8

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

GPIO.setup(myPin,GPIO.OUT)

GPIO.setup(myPin2,GPIO.OUT)

# In VideoCapture object either Pass address of your Video file

# Or If the input is the camera, pass 0 instead of the video file

cap = cv2.VideoCapture(-1)

cap.set(3,480)

cap.set(4,320)

if cap.isOpened() == False:

    print("Error in opening video stream or file")

while(cap.isOpened()):

    ret, frame = cap.read()

    #dist = distance()

    #print(dist)

    #if (dist < 13):

        #GPIO.output(myPin2, GPIO.HIGH)



    if ret:
```

```
frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

aruco_dict = cv2.aruco.Dictionary_get(cv2.aruco.DICT_6X6_50)

parameters = cv2.aruco.DetectorParameters_create()

corners, ids, rejected = cv2.aruco.detectMarkers(

    frame, aruco_dict, parameters=parameters)



if ids in range(0,10) and corners:

    print("Friend")

    #cv2.aruco.drawDetectedMarkers(frame,corners, ids)

# Display the resulting frame

elif ids in range(10,20) and corners:

    print("Enemy")

    GPIO.output(myPin, GPIO.HIGH)

    state = GPIO.input(myPin)

    print(state)

    print("HIGH")



    cv2.aruco.drawDetectedMarkers(frame,corners,ids)

else:

    print("No TAG")
```

```
            cv2.imshow('Frame',frame)

            # Press esc to exit

            if cv2.waitKey(20) & 0xFF == 27:

                break

            GPIO.output(myPin, GPIO.LOW)

            state = GPIO.input(myPin)

            print(state,"low")

            print("LOW")

        else:

            break

cap.release()

cv2.destroyAllWindows()
```