Autonomous Enemy Detection Bot

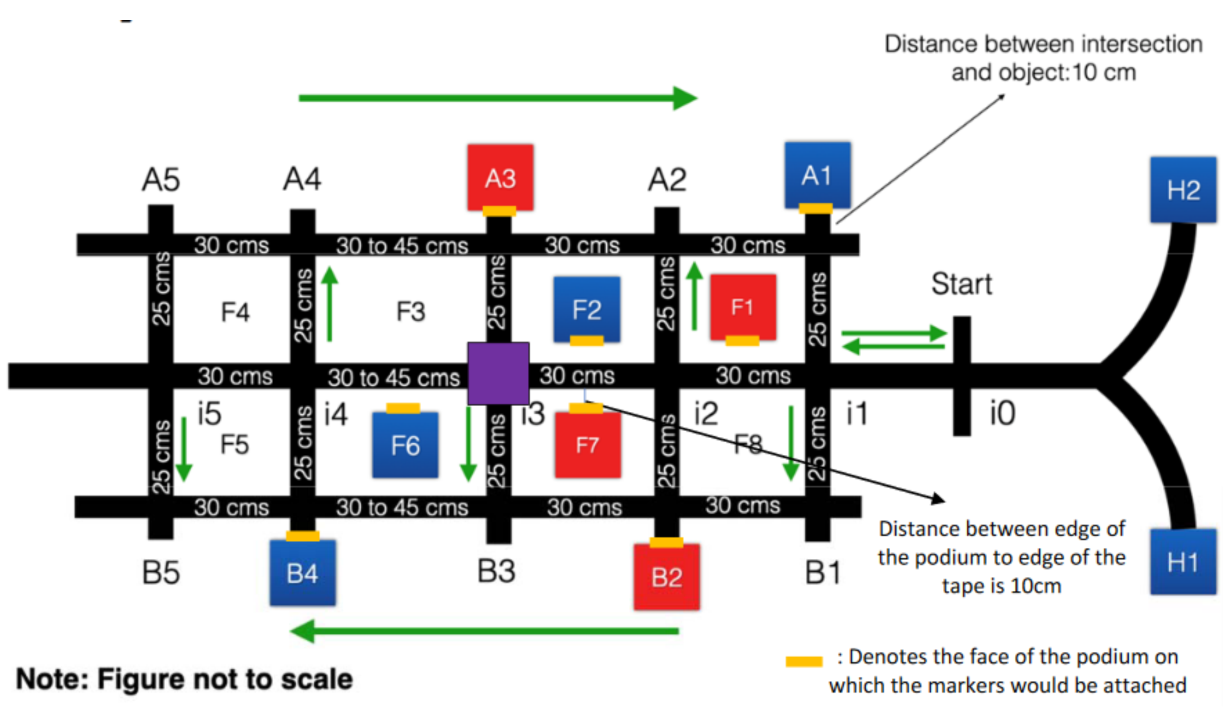
By-
Avirat Varma - N17296397
Rishi Kavi - N15231052
Pavan Chowdary Cherukuri - N10938396

Table of Contents

INTRODUCTION

An autonomous robot is to be designed that will drive around the streets of Manhattan to find enemy intruders (the enemies) among us who are troubling the civilians (the friendlies) in the city. The robot must lookout for the enemies and eliminate them by scouring through the streets while following all the traffic rules and should avoid obstacles.

OBJECTIVES



Distance between intersection and object:10 cm
A5
A4
A3
A2
A1
H2
30 cms
30 to 45 cms
30 cms
30 cms
25 cms
F4
F3
F2
F1
Start
i5
i4
i3
i2
i1
i0
F5
F6
F7
F8
B5
B4
B3
B2
B1
H1
Distance between edge of the podium to edge of the tape is 10cm
Note: Figure not to scale
: Denotes the face of the podium on which the markers would be attached

- The robot must use 2 microcontrollers: 1st - Raspberry Pi or similar and 2nd - Arduino Uno/Propeller Activity Board.
- The robot will start at either one of the safe houses.
- The robot must always follow the path (black tape).
- The robot must reach i0 (start position) and should NOT stop at any time during its full run.

- The robot must move towards the friendlies/enemy's locations to reach them while avoiding obstacles on its path. It should provide an indication of reaching each location by some means (e.g., LED, piezoelectric buzzer, etc.). Note: Reaching locations would mean stopping in front of the location (i.e., the corresponding intersection for the objects placed at A1-A5, B1-B5 and the corresponding street segment for objects placed on the sides of the middle lane) and providing different and clear indications for friendlies and enemies.
- The robot should knock off the enemies from their car (or podium) by a mechanism attached to the robot. It should NOT hurt the friendlies.
- The friendlies/enemies locations will not be revealed for hard coding of the solution, instead it must be found by means of a camera mounted on the robot. We will be provided with Aruco tags (fiducial marker) by default, (Tag Ids corresponding to friendlies would be in the range of 0-9; Tag Ids corresponding to enemies would be in the range of 10-19).
- We will be assigned only those locations that we can reach without breaking any traffic rules.
- The friendlies/enemies are represented by cylinders and would be mounted on a platform (representing car or podium). The Aruco tag will be stuck on one face of the platform.
- The size of the default tags will be provided will be 1in×1in

METHODOLOGY

The behavioral architecture of the bot is as follows:

- The bot must traverse through the grid by laterally aligning itself at the center of the a line (a black line of 2.5cm width in this case).
- Starting from the home position, a curved track takes the bot to the first intersection labeled *i0*. Indicating this intersection without slowing down or stopping with a yellow LED.
- The delivery bot continues further to intersection *i1*. At this point the bot performs the following tasks at the same time using multiple cores at its disposal:

    - Indicate intersection detection (yellow LED).
    - Ping center ultrasonic sensor to determine the location of the obstacle (traffic congestion). Based on the ping duration, the intersection of the congestion is determined.
- Move straight until the obstacle and process all the possible objects on the left, take a u-turn and return to *i1* while processing all the objects on the left.
- Turn right from the main street at *i1*, to traverse on street B. The bot then moves through this street till the street at *B4*, detecting and indicating objects, LED for friendly, knocking for enemies.
- Taking a right turn at *B4*, the bot moves straight to location *i4*. Depending on where the obstacle is, it takes a right here and processes the remainder of the possible locations on the main street, returning to *i4* and moving onto *A4*.
- The bot then moves down street A, detecting and indication objects till A1 and then moves to the main street.
- Returning to intersection *i1*, the bot moves up via street B to finally reach and detect an object at *B5* and indicate if the case exists.
- A final push to move out of the grid, and stop

LED Indications are as follows:

- Yellow LED - Intersection detected
- Red LED - Enemy
- Green LED - Friendly

Line Following

The reflectance sensor array gives a value of 0 for minimal reflectance (white surface) and 2500 for maximum reflectance (black surface). The algorithm aims to sense the line using a comparison of the values from all sensors. Using this, at all hierarchies of the logic (mentioned above), the bot must be laterally center aligned on the black line.

The IR sensor array was meticulously selected to ensure that the IR LEDs are equidistant at 0.95mm to avoid the array to bounce between minimum and maximum detection values i.e the IR LED is not on the edge of the tape.

Tag Detection

The Aruco Tag Detection was done through the Pi Camera attached to the Raspberry Pi 4. The library “aruco” and its functions from cv2 were used for the tag detection. Once the tag was detected , its ID was used to categorize it as an enemy or friendly. If it was an enemy, the motion of the robot (controlled by Propeller) was paused, red LED was lit and the enemy was knocked down using a servo motor interfaced with the Raspberry Pi itself. If it was a friendly, green LED was lit. A single digital channel between the Raspberry Pi and the Propeller was used to communicate the pause. When the RPi makes this line high, the Propeller board pauses the robot motion for a while. The Propeller constantly monitors this signal during its motion to stop for any enemies.

Computation time

Since our algorithm enables multiple cores of the Propeller microcontroller, multiple services were executed without hindering the main algorithm. Thereby, allowing the bot to detect, indicate and ping the ultrasonic sensors without slowing down the bot or stopping. Since, our line following algorithm is efficient, the adjustments to the control of the bot are done at a fast rate, thereby eliminating the need to add a PID controller to refine the maneuvers of the bot. The fast processing speed of the Propeller microcontroller makes all executions and their outputs quick.

CIRCUIT DESIGN

List of components are as follows:

1. 1 HC-SR04 Ultrasonic Sensors
2. 2 Continuous Servo motors
3. 1 Standard Servo motor
4. 8-InfraRed Sensor Array
5. Parallax Propeller Activity Board
6. Raspberry Pi 4
7. Pi Camera
8. 2 - 6V(1.5Vx4) Battery packs
9. 1 Indicator LEDs
10. Power Bank

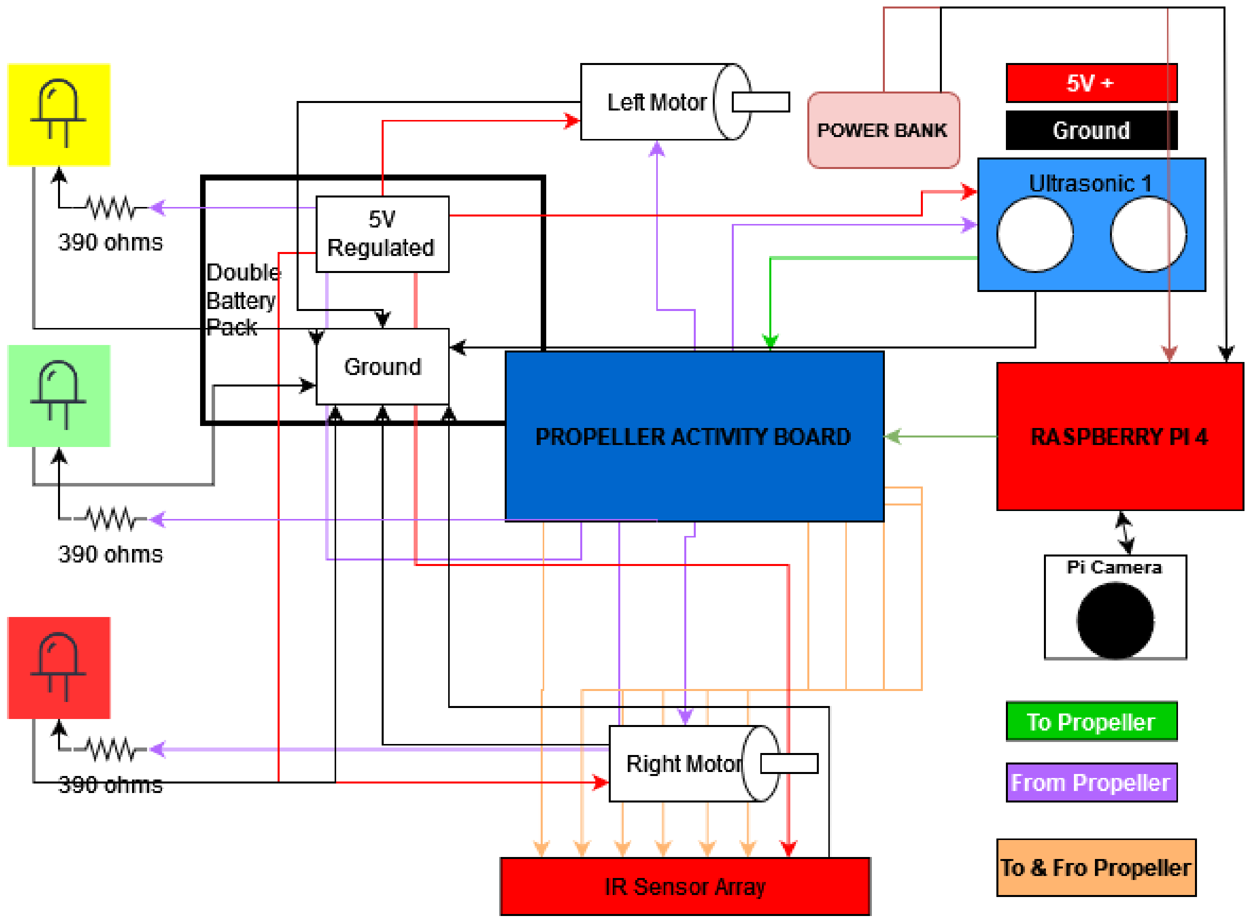


Figure 1: Circuit Design

MECHANICAL DESIGN

All the 3 Dimensional CAD models made as seen in Figure x were modeled in AutoDesk Fusion 360 software. This design was 3D printed using PLA 18 material which is lightweight and has considerable strength. The thickness and lengths of the design were optimized to provide the required strength and support.

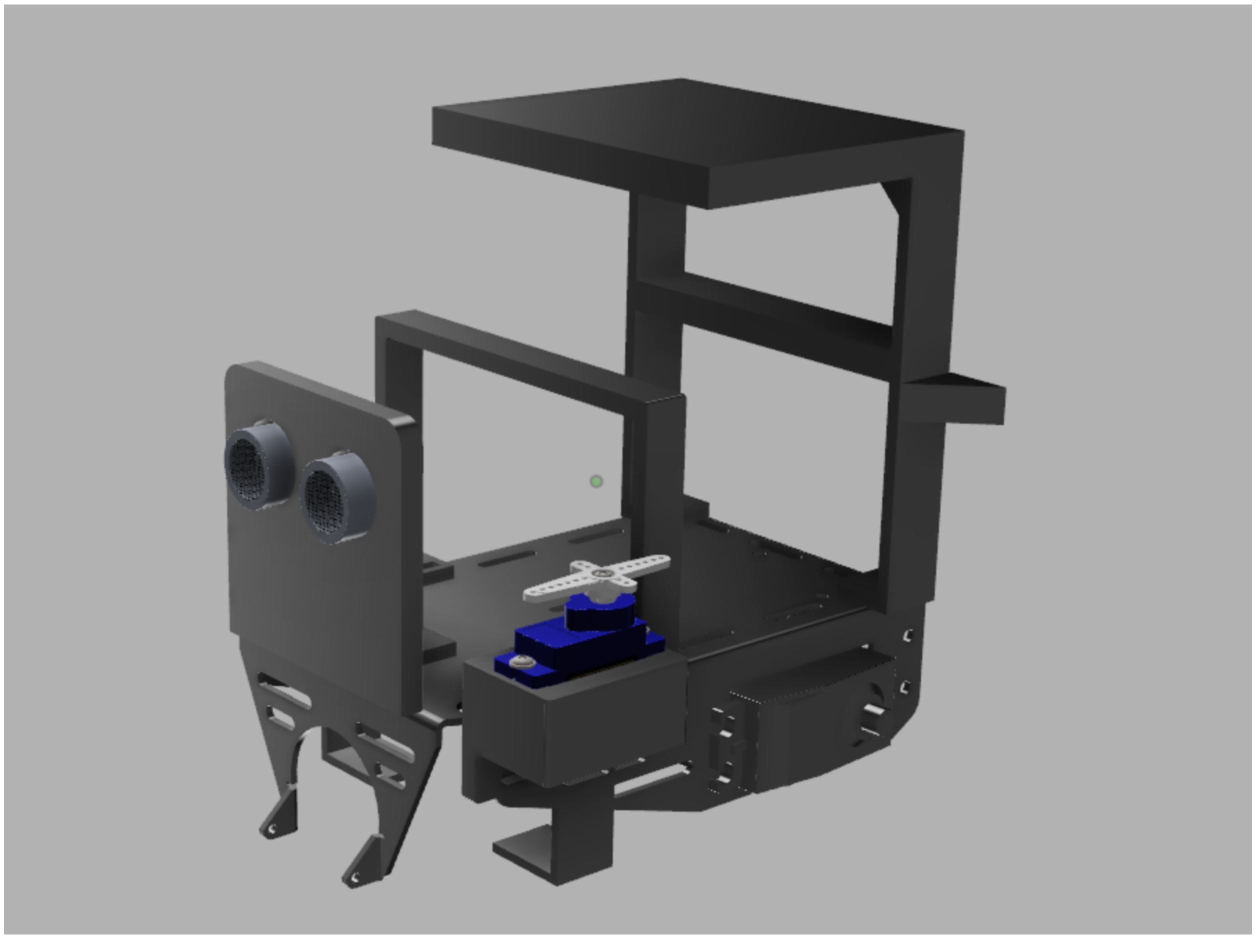


Figure 2: 3D CAD Assembly

- The Ultrasonic Sensor Housing was designed keeping in mind - aesthetic appeal and accurate positioning for the HC-SR04 ultrasonic sensor. Since there is only one ultrasonic sensor being used, the design was made to be minimal.
- A mount and a pedestal were made to hold the propeller board at an elevated level, leaving the chassis of the boe-bot to accommodate the power bank and 9 volt battery. The pedestal also provides second level elevation for the raspberry pi which makes sure no additional space around the bot is compromised thereby providing the required clearance for maneuvering around the obstacle.
- A servo holder was made to hold the standard sg-90 servo on the left for knocking. The servo holder is made to hold the servo at an offset on the left side, so that the knocker (attached to sg-90) doesn't interfere with any parts of the bot.

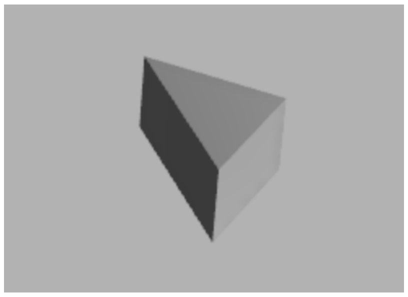


Figure 3: Angled Camera Mount

- A camera mount is made in the form of a wedge to provide the camera 45 degree angle with the vertical for early detection of the tags (friends/enemies). This early detection facilitates the standard servo to knock the enemies synchronously. A camera holder was also made so that the angle with horizontal can be varied to change the camera's field of view as required by the user.
- Two IR sensor holders were made for the Pololu QTR-8RC IR sensor array, which holds the sensor array 4mm from the ground and within the length of the bot's chassis.

PSEUDO CODE

```
Define all IR Sensor Pins
Define al Ultrasonics Trigger and Echo Pins
Define Left & Right motor pins
Define LED indicator pins
Define values for Push, Right and Left turn

Initialise SensorValues array
Initialize Ultrasonic Ping values
Initialize Line Following index value and intersection counter (inter)

Function prototypes:
    ● Intersection detection();
    ● Line detection();
    ● Line follower();
    ● Follow till object();
    ● Center ultrasonic ping();
    ● Left ultrasonic ping();
    ● Push to align bot();
```

- Shortest push();
- Left Turn();
- Right Turn();
- Object indication(); //cog 3
- Intersection indication(); //cog 1
- Obstacle indication(); //cog 2
- Brake();
- Short push();
- Completion indication();

```
Initialize 3 unsigned int stacks for cogs 1,2 and 3
Initialize 3 static volatile int variables for cogs 1,2 and 3

int main(){
while(1){
intersection detection();
if Check for line;
{
if line is centered
{robot moves straight;}
else if line is on left
{robot moves left;}
else if line is on right
{robot moves right;}
}
Check if i1 is reached
{
Start intersection indication cog1;
push();
Center ultrasonic();
if check for obstacle at i2
{
intno=2;
spush();
Follow till object();
uturn();
pause(1000);
Line follower();
spush();
Right Turn();
}
else if check for obstacle at i3
{
intno=3;
spush();
Line follower();
spush();
Follow till object();
uturn();
pause(1000);
Line follower();
spush();
Line follower();
spush();
Right Turn();
}
else if obstacle at i5
{
intno=5;
spush();
Line follower();
spush();
Line follower();
spush();
Line follower();
spush();
Follow till object();
uturn();
pause(1000);
Line follower();
spush();
Line follower();
spush();
Line follower();
spush();
Line follower();
spush();
Right Turn();

}
short push();
follower();
//Reached B1
if rpi=1
{pause;}
short push();
Right turn();
for(to move through B1 to B4)
{
Follow line;
Align ultrasonic;
if rpi=1
{pause;}
}
```

```
//Reached B4
Right turn();
Follow Line;
//Reached i4
if obstacle not at i5
{
Right turn();
If obstacle at i2
{Line follower;}
Follow till object();
uturn();
Line Follower();
if obstacle at i2
{Line follower;}
Right turn();
}
Line Follower();
//Reached A4
Short push();
Right turn();
if rpi=1
{pause;}
for(to move through A4 to A1)
{
Follow line;
Align ultrasonic;
if rpi=1
{pause;}
//Reached A1
Shortest push();
Right turn();
Push();
pause(25);
Push();
follower();
//Reached B1
push();
Right turn();
for(final run from B1 to B5)
{
follower();
Short push();
}
//Reached B5
Left ultrasonic();
if rpi=1
{pause;}
party();
break;
}
else if(bot is at i0)
{
Start intersection indication cog1;
Increment intersection counter;
push();
}
else if(special case to ignore street merge) //special intersection ignorance case
{if(l>r)
{for(int x=0;x<=23;x++)
{
pulse_out(rm,400);
pause(5);
}
f=1;}
else
{
for(int x=0;x<=23;x++)
{pulse_out(lm,2300);
pause(5);}
f=1;}
c++;
}
else if(c==0)
{
c++;
pause(10);
}}}

//-----------function definitions-----------------------------
int inter() //returns threshold sensor value for intersection detection
{
Define sensor pins;
for(loop through all sensor pins)
{
Set direction output for all pins;
}
Define input time array;
for(loop through all sensor pins)
{
Set pin i, high;
pause(1);
Set direction input for all pins;
Store rc_time of pin i to input time array i;
```

```
}
Initialize sum of all reflectances to 0;
for (loop through all sensor pins)
{
add time of each led incrementally in all indices of time array;
}
return summation value;
}

int linloc() // returns the location of the black line over the sensor array
{
Define sensor pins;
for(loop through all sensor pins)
{
Set direction output for all pins;
}
Define input time array;
for(loop through all sensor pins)
{
Set pin i, high;
pause(1);
Set direction input for all pins;
Store rc_time of pin i to input time array i;
}
Define line location variable and temporary variables;
for (loop through all sensor pins)
{
if (time of sensor i is greater than temp. variable)
{
Temp. variable = time of that sensor[i];
Line location = index of that sensor;
}
}
return index;
}

void follower()  // follows black line until an intersection is detected
{
initialize summation variable=0;
sum=inter();
while(define reflectance summation range for line)
{
if rpi=1
{pause;}
int loc=linloc();
if (index is on center sensor) //if true, robot moves straight
{
```

```
Bot goes straight;
}
else if (index is on left sensor) //if true, robot moves left
{
Bot turns left
}
else if (index is on right sensor) //if true, robot moves right
{
Bot turns right;
}
sum=inter();
}
Start intersection indication cog1;
}

void followobs()  // follows black line until a distance from the object
{
check center ultrasonic();
while(distance<3cms)
{
if rpi=1
{pause;}
int loc=linloc();
if (index is on center sensor) //if true, robot moves straight
{
Bot goes straight;
}
else if (index is on left sensor) //if true, robot moves left
{
Bot turns left
}
else if (index is on right sensor) //if true, robot moves right
{
Bot turns right;
}
check center ultrasonic();
}
Start intersection indication cog1;
}

//------------------ultrasonic----------------
int cl() // returns distance of nearest object in front of left sensor
{
Short brakes to motors;
Ping left ultrasonic;
Store duration of ping;
Calculate distance based on duration;
```

```
Return distance;
}

int cc() // returns distance of nearest object in front of center sensor
{
Short brakes to motors;
Ping center ultrasonic;
Store duration of ping;
Calculate distance based on duration;
Return distance;
}

void push() // a linear push
{for(int x=0;x<=p1;x++)
{Bot moves straight;}}

void puss() // a very short linear push
{for(int x=0;x<=p3;x++)
{Bot moves straight;}}

void L90() // turn 90 degrees left
{for(int x=0;x<=left;x++)
{Point rotation left;}}

void R90() // turn 90 degrees right
{for(int x=0;x<=right;x++)
{Point rotation right;}}

void brake() // single brake pulse
{Stop motor;}

void object(void *parc) //led blink for object indication
{Blink red LED with 500ms pause;
Stop cog3;}

void isec(void *para) // led blink for intersection indication
{Blink yellow LED with 500ms pause;
Stop cog1;}

void blink(void *parb) // led blink for obstacle location indication
{for(int pav=0; pav<intersection number; pav++)
{Blink yellow blue with 500ms pause;}
Stop cog2;
}

void party() // random blink sequence at the end
{Blink yellow and red led alternatively 4 times;}
```

```
void spush() // a short linear push
{for(int x=0;x<=p2;x++)
{Bot moves straight;}}
```

BILL OF MATERIALS

Component	Cost/Unit	Quantity	Cost
Boe-Bot Chassis	$70	1	$70
Parallax Continuous Servo Motor	$15	2	$30
Standard Servo Motor	$5	1	$5
Propeller Activity Board	$105	1	$105
Raspberry Pi 4	$120	1	$120
Pi Camera	$25	1	$25
HC-SR04 Ultrasonic Sensor	$2	1	$2
Pololu QTR-8RC IR Sensor Array	$15	1	$15
Custom Designed Sensor Holder	$0	1	$0
Battery Pack	$7	2	$14
Power Bank	$10	1	$10
Duracell Battery	$1.375	8	$11
LED	$0.2	3	$0.6
Switch	$0	1	$0
Total Cost			**$407.6**

RESULTS

1. The bot accurately centers itself on the line.
2. The bot can successfully detect and indicate intersections.
3. The bot can successfully detect and indicate the obstacle using the central ultrasonic sensor when at i1.
4. The bot can successfully detect and indicate the objects as friendly or enemy using the Pi Camera mounted on the left side of the bot.
5. The bot successfully knocks down all the enemies.
6. The bot successfully gives indications for all events.
7. The bot doesn't have to stop anywhere during the process, by use of multiple cores.
8. The bot successfully stops after servicing all intersections.
9. Changing batteries can vary the speed and turning calibrations of the robot.

APPENDIX - CODE

Propeller Code

```
#include <stdio.h>         // Recommended over iostream for saving space
#include <propeller.h>               // Propeller-specific functions
#include <simpletools.h>
#include <time.h>
#include "servo.h"// the motor pulse values are different for normal operation and pushes
#define ut1 12 //left ultrasonic trigger
#define ue1 13 //left ultrasonic echo
#define ut2 10 // center ultrasonic tigger
#define ue2 11 //center ultrasonic echo
#define lm 14 //left motor signal line
#define rm 15 //right motor signal line
#define rpi 13// signal from rpi
#define p1 52 //push 1 push
#define p2 23 //push2 for short push
#define p3 20 //push 3 puss (this puss is only used when reaching A1, i have changed the values to put it
after uturn initial value 44)
#define p77 12
#define left 16
```

```
#define right 18
#define r1 10
#define r2 12
#define l1 7
#define uturnv 110
#define reversev 53
#define reversevshort 25
#define pidelay 2000 // delay when signal from rpi
#define rishi 17 // was 19 earlier at 18:13
#define fslow 10  // was 75 earlier
#define rslow 15  // 15 og // 8
int inter();
int linloc();
void follower();
void followobs();
int cc();
int cl();
void push();
void puss();
void L90();
void R90();
void SR90();
void uturn();
void reverse();
void reverseshort();
void object(void *parc);
void isec(void *para);
void brake();
void spush();
void party();
unsigned int stack1[40 + 25]; // Stack vars for other cog
unsigned int stack2[40 + 25];
unsigned int stack3[40 + 25];
static volatile int cog, cog2, cog3, intno;

int main()
{
 int haha=1;
 int c=0;
 int l=0;
 int r=0;
 int f=0;
 int dc;
 int dl;
 while(1)
 {
       int sum=inter();
       if (sum < 17000 && sum > 2500)
       {
       int loc=linloc();
       if (loc == 2 || loc == 3) //if true, robot moves straight
```

```
        {
        pulse_out(lm,1390);
        pulse_out(rm,1300);
        pause(rslow);
        }
        else if (loc == 0 || loc == 1) //if true, robot moves left
        {
        pulse_out(rm,1300);
        pause(rslow);
        l++;
        }
        else if (loc == 4 || loc == 5) //if true, robot moves right
        {
        pulse_out(lm,1390);
        pause(rslow);
        r++;
        }
        }
        else if(c==3)
        {
        cog=cogstart(&isec, NULL, stack2, sizeof(stack2));
//////////////////////

        spush();
        dc=cc();
        printf("%d \n",dc);
        if(dc<45)
        {
        intno=2;
        spush();
        followobs();
        uturn();
        reverseshort();
        follower();
        puss();
        R90();
        }
        else if(dc>45 && dc<70)
        {
        intno=3;
        spush();
        follower();
        spush();
        followobs();
        uturn();
        reverseshort();
        pause(1000);
        follower();
        spush();
        follower();
        spush();
R90();

}
else if (dc>70)
{
intno=5;
spush();
follower();
spush();
follower();
spush();
follower();
spush();
followobs();
uturn();
reverseshort();
pause(1000);
follower();
spush();
follower();
spush();
follower();
spush();
follower();
spush();
R90();

}

follower();
puss();
puss();
SR90();
pause(1500);
if(input(rpi))
{
pause(pidelay);
}
for(int bint=0;bint<3;bint++)
{
spush();
follower();
spush();
if(input(rpi))
{
pause(pidelay);
}
pause(100);

}
R90();
```

```
spush();
follower();
spush();
//i4
if(intno!=5)
{
L90();
spush();
follower();
spush();
uturn();
follower();
spush();
pause(50);

//i4
pause(50);////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

if(intno==2)
{
spush();
follower();
}
puss();
followobs();
//obstacle reached
uturn();
//uturn done
if(intno==2)
{
follower();
spush();
}
follower();
spush();
//i4 again
R90();
spush();
}
follower();
//A4
puss();
puss();
SR90();
if(input(rpi))
{
pause(pidelay);
}
for(int aint=0;aint<3;aint++)
{
spush();
```

```
follower();
spush();
if(input(rpi))
{
pause(pidelay);
}
pause(100);


}
//A1
spush();
R90();
spush();
pause(25);
follower();
spush();
follower();
//B1
spush();
R90();
for(int aint=0;aint<4;aint++)
{
spush();
follower();
spush();

}
//B5
if(input(rpi))
{
pause(pidelay);
}
party();
break;

}
else if(c==2)
{
cog=cogstart(&isec, NULL, stack2, sizeof(stack2));
c++;
spush();
}
else if(c==1 && f==0) //special intersection ignorance case
{
printf("c=1\n");
if(l>r)
{
for(int x=0;x<=rishi;x++)
{
pulse_out(rm,400);
```

```
            pause(rslow); //not sure if this should be slow or change it back to 9
            }
            f=1;
            }
            else
            {
            for(int x=0;x<=rishi;x++)
            {
            pulse_out(lm,2300);
            pause(rslow);
            }
            f=1;
            }
            c++;
            }


            else if(c==0)
            {
            c++;
            pause(10);
            }
  }
}




//------------functions-------------------
int inter()
{
  int pins[6]={8,7,6,5,4,3};
  for(int i=0;i<6;i++)
  {
            set_direction(pins[i],1);
  }
  int times[6];
  for(int i=0;i<6;i++)
  {
            high(pins[i]);
            pause(1);
            set_direction(pins[i],0);
            times[i]=rc_time(pins[i],1);
  }
  int s = 0;
  for (int i = 0; i < 6; i++)
  {
            s += times[i];
  }
  return s;
}
```

```
int linloc()
{
  int pins[6]={8,7,6,5,4,3};
  for(int i=0;i<6;i++)
  {
            set_direction(pins[i],1);
  }
  int times[6];
  for(int i=0;i<6;i++)
  {
            high(pins[i]);
            pause(1);
            set_direction(pins[i],0);
            times[i]=rc_time(pins[i],1);
  }
  int bleh = 0;
  int maxind;
  for (int i = 0; i < 6; i++)
  {
            if (times[i] > bleh)
            {
            bleh = times[i];

            maxind = i;
            }
  }
  return maxind;
}

void follower()
{
  int sum=0;
  sum=inter();
  while(sum < 17500 && sum > 2000)
  {
            if(input(rpi))
            {
            pause(pidelay);
            }
            int loc=linloc();
            if (loc == 2 || loc == 3) //if true, robot moves straight
            {
            pulse_out(lm,1390);
            pulse_out(rm,1300);
            pause(fslow);
            }
            else if (loc == 0 || loc == 1) //if true, robot moves left
            {
```

```
            pulse_out(rm,1300);
            pause(fslow);
            }
            else if (loc == 4 || loc == 5) //if true, robot moves right
            {
            pulse_out(lm,1390);
            pause(fslow);
            }
            sum=inter();
            }
            high(0);
            pause(500);
            low(0);
}
void followobs()
{
  int cdist=0;
  cdist=cc();
  while(cdist>2)
  {
            if(input(rpi))
            {
            pause(pidelay);
            }
            int loc=linloc();
            if (loc == 2 || loc == 3) //if true, robot moves straight
            {
            pulse_out(lm,1390);
            pulse_out(rm,1300);
            pause(fslow);
            }
            else if (loc == 0 || loc == 1) //if true, robot moves left
            {
            pulse_out(rm,1300);
            pause(fslow);
            }
            else if (loc == 4 || loc == 5) //if true, robot moves right
            {
            pulse_out(lm,1390);
            pause(fslow);
            }
            cdist=cc();
            }

}
//------------------ultrasonic----------------
int cl()
{
            brake();
            low(ut1);
            pause(2);
		high(ut1);
		pause(10);
		low(ut1);
		int duration1 = pulse_in(ue1, 1);
		int dl = (duration1 * 0.0343) / 2;
		return dl;
		}
int cc()
{		low(ut2);
		pause(2);
		high(ut2);
		pause(10);
		low(ut2);
		int duration2 = pulse_in(ue2, 1);
		int dc = (duration2 * 0.0343) / 2;
		return dc;
		}
void push()
{
		for(int x=0;x<=p1;x++)
		{
		pulse_out(rm,1290);
		pulse_out(lm,1450);
		pause(10);
		}
		}
void puss()
{
		for(int x=0;x<=p3;x++)
		{
		pulse_out(rm,400);
		pulse_out(lm,2300);
		pause(5);
		}
		}
void spuss()
{
		for(int x=0;x<=p77;x++)
		{
		pulse_out(rm,400);
		pulse_out(lm,2300);
		pause(5);
		}
		}
void L90()
{		for(int x=0;x<=left;x++)
		{
		pulse_out(rm,900);
		pulse_out(lm,900);
		pause(30);
		}
```

```
}

void R90()
{          for(int x=0;x<=right;x++)
           {
           pulse_out(rm,1600);
           pulse_out(lm,1600);
           pause(30);
           }
}
void SR90()
{
           for(int x=0;x<=55;x++)
           {
           pulse_out(rm,1380);
           pulse_out(lm,1380);
           pause(40);
           }

}
void uturn()
{          for(int x=0;x<=uturnv;x++)
           {
           pulse_out(rm,1380);
           pulse_out(lm,1380);
           pause(30);
           }
}
void reverse()
{
  for(int x=0;x<=reversev;x++)
           {
           pulse_out(lm,1300);
           pulse_out(rm,1390);
           pause(10);
           }
}
void reverseshort()
{
  for(int x=0;x<=reversevshort;x++)
           {
           pulse_out(lm,1300);
           pulse_out(rm,1390);
           pause(10);
           }
}
void brake()
{
   pulse_out(rm,1350);
   pulse_out(lm,1350);
 }
```

```
void object(void *parc)
{
 pause(25);
 high(2);
 pause(500);
 low(2);
 cogstop(cog3);
}
void isec(void *para)
{
high(0);
pause(500);
low(0);
cogstop(cog);
}

void party()
{
 high(2);
 pause(500);
 low(2);
 high(1);
 pause(500);
 low(1);
 high(2);
 pause(500);
 low(2);
 high(1);
 pause(500);
 low(1);
 high(2);
 pause(500);
 low(2);
 high(1);
 pause(500);
 low(1);
}

void spush()
{
        for(int x=0;x<=p2;x++)
        {
        pulse_out(rm,1280);
        pulse_out(lm,1450);
        pause(10);
        }
        }
```

Raspberry Pi Code

```
import cv2
import cv2.aruco as aruco
import RPi.GPIO as GPIO
import time
import numpy as np
from gpiozero import Servo
import pigpio
from time import sleep
pi = pigpio.pi()
servo=18 #pin 12
LedPinE=4  #pin7
LedPinF=17 #pin 11
commpin=27 #pin 13
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(servo,GPIO.OUT)
GPIO.setup(LedPinE,GPIO.OUT)
GPIO.setup(LedPinF,GPIO.OUT)
GPIO.setup(commpin, GPIO.OUT)
#servo = Servo(ServoPin)

cap = cv2.VideoCapture(0)
delay = 1
count0=1
count1=1
count2=1
count3=1
count4=1
count5=1
count6=1
count7=1
count8=1
count9=1

count10=1
count11=1
count12=1
count13=1
count14=1
count15=1
count16=1
count17=1
count18=1
count19=1

def Aruco(img, markersize = 6, total = 250, draw = True):
    key = getattr(aruco, f'DICT_{markersize}X{markersize}_{total}')
    #grey_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    Dictionary = aruco.Dictionary_get(key)
    Para = aruco.DetectorParameters_create()
    bbox, ids,_ = aruco.detectMarkers(img, Dictionary, parameters = Para)
#    print(ids)

    if draw:
            aruco.drawDetectedMarkers(img, bbox)
    return bbox, ids

#def ServoMove(ServoAngle):
#  sleep(1)
#  PulseWidth = 0
#  PulseWidth=ServoAngle*9.72+500 #OnTime for PWM
#  PulseWidth=PulseWidth/1000000
#  for indx in range(0,2):
#          GPIO.output(ServoPin, 1)
#          sleep(PulseWidth)
#          GPIO.output(ServoPin, 0)
#          sleep(0.02)

pwm = pigpio.pi()
pwm.set_mode(servo, pigpio.OUTPUT)

pwm.set_PWM_frequency( servo, 50 )

def roi(img):
    height = img.shape[0]
    width = img.shape[1]
    grey_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ROI = np.array([[(120,height),(120,180),(500,180),(500,height)]], dtype=np.int32)
    black = np.zeros_like(grey_img)
    roi = cv2.fillPoly(black, ROI, 255)
    roi_img = cv2.bitwise_and(grey_img,roi)
    return roi_img
#sleep(0.5)
#servo.max()
GPIO.output(LedPinE,GPIO.LOW)
GPIO.output(LedPinF,GPIO.LOW)
#servo.max()
while True:
  GPIO.output(LedPinE,GPIO.LOW)
  ret, img = cap.read()
  grey_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  bbox, ids = Aruco(grey_img)
  if cv2.waitKey(1) == 113:
          break
  cv2.imshow('Halal4Lyf',grey_img)
  id=int(ids or 69)
  print(id)
  if id==10 and count10<2:
          GPIO.output(commpin, GPIO.HIGH)
```

```
        sleep(0.5)
        GPIO.output(commpin, GPIO.LOW)
        GPIO.output(LedPinE,GPIO.HIGH)
        pwm.set_servo_pulsewidth( servo, 500 ) ;
        time.sleep( 3 )
        GPIO.output(LedPinE,GPIO.LOW)
        pwm.set_servo_pulsewidth( servo, 2500 ) ;
        time.sleep( 3 )
        pwm.set_PWM_dutycycle(servo, 0)
        pwm.set_PWM_frequency( servo, 0 )
        sleep(0.2)
        count10 = 2
    elif id==11 and count11<2:
        GPIO.output(commpin, GPIO.HIGH)
        sleep(0.5)
        GPIO.output(commpin, GPIO.LOW)
        GPIO.output(LedPinE,GPIO.HIGH)
        pwm.set_servo_pulsewidth( servo, 500 ) ;
        time.sleep( 3 )
        GPIO.output(LedPinE,GPIO.LOW)
        pwm.set_servo_pulsewidth( servo, 2500 ) ;
        time.sleep( 3 )
        pwm.set_PWM_dutycycle(servo, 0)
        pwm.set_PWM_frequency( servo, 0 )
        sleep(0.2)
        count11 = 2
    elif id==12 and count12<2:
        GPIO.output(commpin, GPIO.HIGH)
        sleep(0.5)
        GPIO.output(commpin, GPIO.LOW)
        GPIO.output(LedPinE,GPIO.HIGH)
        pwm.set_servo_pulsewidth( servo, 500 ) ;
        time.sleep( 3 )
        GPIO.output(LedPinE,GPIO.LOW)
        pwm.set_servo_pulsewidth( servo, 2500 ) ;
        time.sleep( 3 )
        pwm.set_PWM_dutycycle(servo, 0)
        pwm.set_PWM_frequency( servo, 0 )
        sleep(0.2)
        count12 = 2
    elif id==13 and count13<2:
        GPIO.output(commpin, GPIO.HIGH)
        sleep(0.5)
        GPIO.output(commpin, GPIO.LOW)
        GPIO.output(LedPinE,GPIO.HIGH)
        pwm.set_servo_pulsewidth( servo, 500 ) ;
        time.sleep( 3 )
        GPIO.output(LedPinE,GPIO.LOW)
        pwm.set_servo_pulsewidth( servo, 2500 ) ;
        time.sleep( 3 )
        pwm.set_PWM_dutycycle(servo, 0)
        pwm.set_PWM_frequency( servo, 0 )
        sleep(0.2)
        count13 = 2
    elif id==14 and count14<2:
        GPIO.output(commpin, GPIO.HIGH)
        sleep(0.5)
        GPIO.output(commpin, GPIO.LOW)
        GPIO.output(LedPinE,GPIO.HIGH)
        pwm.set_servo_pulsewidth( servo, 500 ) ;
        time.sleep( 3 )
        GPIO.output(LedPinE,GPIO.LOW)
        pwm.set_servo_pulsewidth( servo, 2500 ) ;
        time.sleep( 3 )
        pwm.set_PWM_dutycycle(servo, 0)
        pwm.set_PWM_frequency( servo, 0 )
        sleep(0.2)
        count14 = 2
    elif id==15 and count15<2:
        GPIO.output(commpin, GPIO.HIGH)
        sleep(0.5)
        GPIO.output(commpin, GPIO.LOW)
        GPIO.output(LedPinE,GPIO.HIGH)
        pwm.set_servo_pulsewidth( servo, 500 ) ;
        time.sleep( 3 )
        GPIO.output(LedPinE,GPIO.LOW)
        pwm.set_servo_pulsewidth( servo, 2500 ) ;
        time.sleep( 3 )
        pwm.set_PWM_dutycycle(servo, 0)
        pwm.set_PWM_frequency( servo, 0 )
        sleep(0.2)
        count15 = 2
    elif id==16 and count16<2:
        GPIO.output(commpin, GPIO.HIGH)
        sleep(0.5)
        GPIO.output(commpin, GPIO.LOW)
        GPIO.output(LedPinE,GPIO.HIGH)
        pwm.set_servo_pulsewidth( servo, 500 ) ;
        time.sleep( 3 )
        GPIO.output(LedPinE,GPIO.LOW)
        pwm.set_servo_pulsewidth( servo, 2500 ) ;
        time.sleep( 3 )
        pwm.set_PWM_dutycycle(servo, 0)
        pwm.set_PWM_frequency( servo, 0 )
        sleep(0.2)
        count16 = 2
    elif id==17 and count17<2:
        GPIO.output(commpin, GPIO.HIGH)
        sleep(0.5)
        GPIO.output(commpin, GPIO.LOW)
        GPIO.output(LedPinE,GPIO.HIGH)
        pwm.set_servo_pulsewidth( servo, 500 ) ;
```

```
            time.sleep( 3 )
            GPIO.output(LedPinE,GPIO.LOW)
            pwm.set_servo_pulsewidth( servo, 2500 ) ;
            time.sleep( 3 )
            pwm.set_PWM_dutycycle(servo, 0)
            pwm.set_PWM_frequency( servo, 0 )
            sleep(0.2)
            count17 = 2
 elif id==18 and count18<2:
            GPIO.output(commpin, GPIO.HIGH)
            sleep(0.5)
            GPIO.output(commpin, GPIO.LOW)
            GPIO.output(LedPinE,GPIO.HIGH)
            pwm.set_servo_pulsewidth( servo, 500 ) ;
            time.sleep( 3 )
            GPIO.output(LedPinE,GPIO.LOW)
            pwm.set_servo_pulsewidth( servo, 2500 ) ;
            time.sleep( 3 )
            pwm.set_PWM_dutycycle(servo, 0)
            pwm.set_PWM_frequency( servo, 0 )
            sleep(0.2)
            count18 = 2
 elif id==19 and count19<2:
            GPIO.output(commpin, GPIO.HIGH)
            sleep(0.5)
            GPIO.output(commpin, GPIO.LOW)
            GPIO.output(LedPinE,GPIO.HIGH)
            pwm.set_servo_pulsewidth( servo, 500 ) ;
            time.sleep( 3 )
            GPIO.output(LedPinE,GPIO.LOW)
            pwm.set_servo_pulsewidth( servo, 2500 ) ;
            time.sleep( 3 )
            pwm.set_PWM_dutycycle(servo, 0)
            pwm.set_PWM_frequency( servo, 0 )
            sleep(0.2)
            count19 = 2
############################################ from 0 to 9
 elif id==0 and count0<2:
            GPIO.output(commpin, GPIO.HIGH)
            sleep(0.5)
            GPIO.output(commpin, GPIO.LOW)
            GPIO.output(LedPinF,GPIO.HIGH)
            start = time.time()
            while time.time() - start < delay:
            pass
            GPIO.output(LedPinF,GPIO.LOW)
            count0 = 2
 elif id==1 and count1<2:
            GPIO.output(commpin, GPIO.HIGH)
            sleep(0.5)
            GPIO.output(commpin, GPIO.LOW)
```

```
        GPIO.output(LedPinF,GPIO.HIGH)
        start = time.time()
        while time.time() - start < delay:
        pass
        GPIO.output(LedPinF,GPIO.LOW)
        count1 = 2
elif id==2 and count2<2:
        GPIO.output(commpin, GPIO.HIGH)
        sleep(0.5)
        GPIO.output(commpin, GPIO.LOW)
        GPIO.output(LedPinF,GPIO.HIGH)
        start = time.time()
        while time.time() - start < delay:
        pass
        GPIO.output(LedPinF,GPIO.LOW)
        count2 = 2
elif id==3 and count3<2:
        GPIO.output(commpin, GPIO.HIGH)
        sleep(0.5)
        GPIO.output(commpin, GPIO.LOW)
        GPIO.output(LedPinF,GPIO.HIGH)
        start = time.time()
        while time.time() - start < delay:
        pass
        GPIO.output(LedPinF,GPIO.LOW)
        count3 = 2
elif id==4 and count4<2:
        GPIO.output(commpin, GPIO.HIGH)
        sleep(0.5)
        GPIO.output(commpin, GPIO.LOW)
        GPIO.output(LedPinF,GPIO.HIGH)
        start = time.time()
        while time.time() - start < delay:
        pass
        GPIO.output(LedPinF,GPIO.LOW)
        count4 = 2
elif id==5 and count5<2:
        GPIO.output(commpin, GPIO.HIGH)
        sleep(0.5)
        GPIO.output(commpin, GPIO.LOW)
        GPIO.output(LedPinF,GPIO.HIGH)
        start = time.time()
        while time.time() - start < delay:
        pass
        GPIO.output(LedPinF,GPIO.LOW)
        count5 = 2
elif id==6 and count6<2:
        GPIO.output(commpin, GPIO.HIGH)
        sleep(0.5)
        GPIO.output(commpin, GPIO.LOW)
        GPIO.output(LedPinF,GPIO.HIGH)
```

```
          start = time.time()
          while time.time() - start < delay:
          pass
          GPIO.output(LedPinF,GPIO.LOW)
          count6 = 2
elif id==7 and count7<2:
          GPIO.output(commpin, GPIO.HIGH)
          sleep(0.5)
          GPIO.output(commpin, GPIO.LOW)
          GPIO.output(LedPinF,GPIO.HIGH)
          start = time.time()
          while time.time() - start < delay:
          pass
          GPIO.output(LedPinF,GPIO.LOW)
          count7 = 2
elif id==8 and count8<2:
          GPIO.output(commpin, GPIO.HIGH)
          sleep(0.5)
          GPIO.output(commpin, GPIO.LOW)
          GPIO.output(LedPinF,GPIO.HIGH)
          start = time.time()
          while time.time() - start < delay:
          pass
          GPIO.output(LedPinF,GPIO.LOW)
          count8 = 2
elif id==9 and count9<2:
          GPIO.output(commpin, GPIO.HIGH)
          sleep(0.5)
          GPIO.output(commpin, GPIO.LOW)
          GPIO.output(LedPinF,GPIO.HIGH)
          start = time.time()
          while time.time() - start < delay:
          pass
          GPIO.output(LedPinF,GPIO.LOW)
          count9 = 2
```