Autonomous Test Kit Delivery Bot

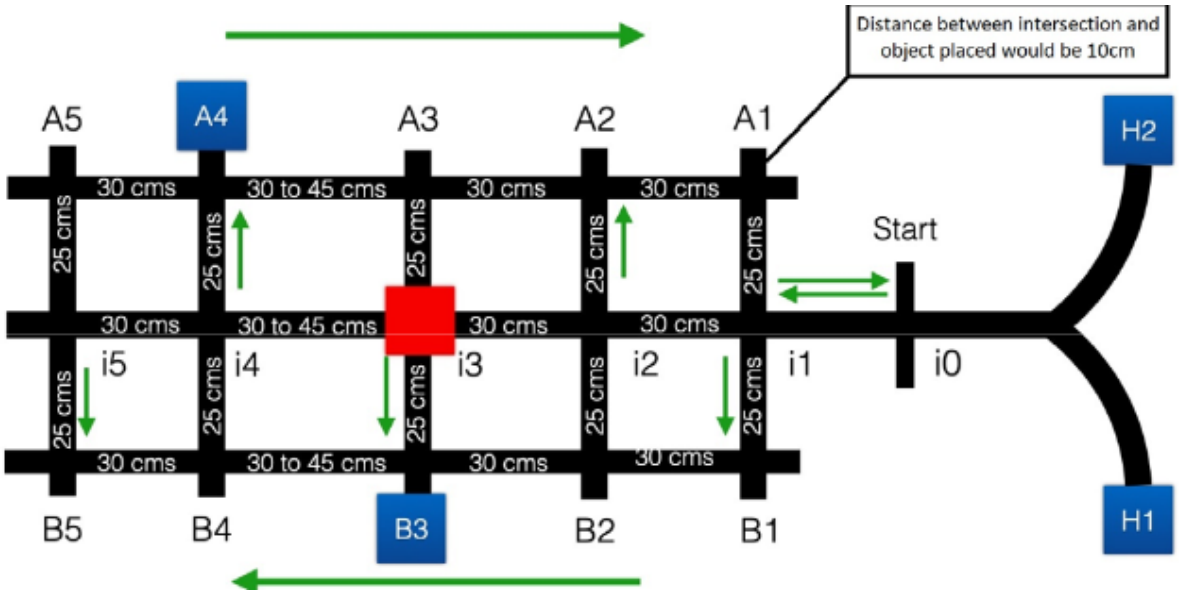
By-
Avirat Varma - N17296397
Rishi Kavi - N15231052
Pavan Chowdary Cherukuri - N10938396

Table of Contents

INTRODUCTION

The pandemic has disrupted normal life. Given these unusual circumstances, a need for autonomous, contact less and fast delivery arises. The project aims to achieve these needs by developing and presenting an autonomous line following bot tailored to serve the city-grid road network of Manhattan. The bot can start at a home position, traverse through the circuit using line following algorithms, following Manhattan traffic laws to complete its objective of reaching/detecting drop-off locations.

OBJECTIVES



- The robot will be placed at either one of the home locations.
- The robot must always follow the path (black tape), emulating the roads of NYC.
- The robot must reach intersection i0 (start position) and should NOT stop at any time during its full run.
- The robot detects intersections and provides an indication of the detection using a LED, on the go without stopping.
- Upon detecting the obstacle on its path, the robot must avoid it.

- The robot must move towards the desired locations to reach them; and should provide an indication of reaching each location on the go by some means (e.g., LED, piezoelectric buzzer, etc.); Reaching locations essentially would mean reaching the corresponding intersection of the location.
- The robot should reach two locations assigned (example here- A3 and B5) by lawfully navigating around the grid.
- The bot at the end, pushes out of the grid and indicates objective completion by a sequence of alternating blinking LEDs.

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.
        - A blue LED will blink the number of times of the intersection number. For example, if the obstacle is at *i3*, the LED will blink 3 times.
    - Ping the left ultrasonic sensor to detect the presence of an object at *B1*.
- Turn left on the street at *i1*, to traverse on street B. The bot then moves through this street till the street at *B4*, detecting and indicating objects (red LED), without stopping.
- Taking a right turn at *B4*, the bot moves straight to location *A4*.

- The bot then moves down street A, detecting and indication objects till A1, without stopping 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, stop and give a special indication of completion.

LED Indications are as follows:

- Yellow LED - Intersection detected
- Red LED - Object Detected
- Blue LED - Intersection at which obstacle was detected.

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.

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. 2 HC-SR04 Ultrasonic Sensors
2. 2 Continuous Servo motors
3. 8-InfraRed Sensor Array
4. Parallax Propeller Activity Board
5. 2 - 6V(1.5Vx4) Battery packs
6. 3 Indicator LEDs

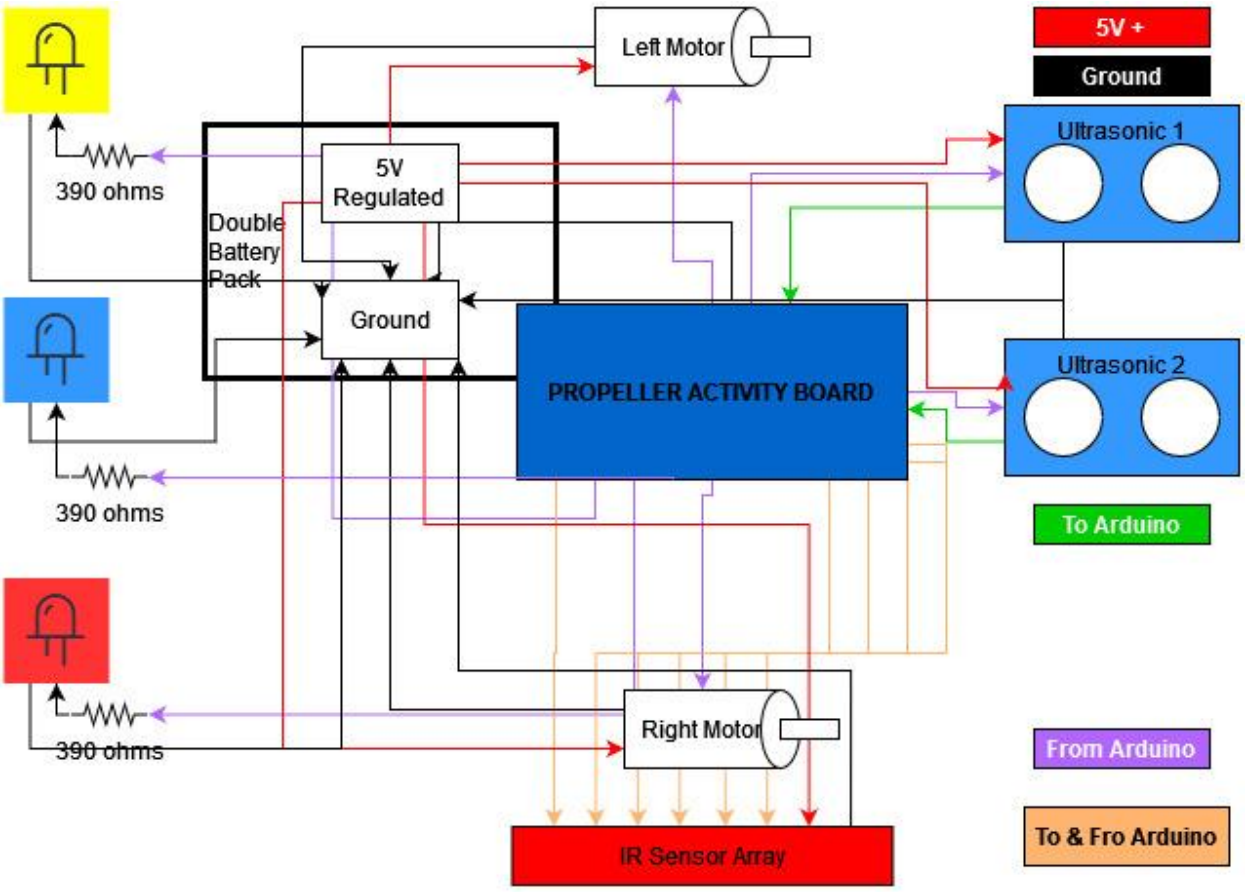


Figure 1: Circuit Design

MECHANICAL DESIGN

The 3 Dimensional CAD model for the Sensor Housing, as seen in the Figure 2 and Figure 3 below was modeled in AutoDesk Fusion 360 software. This design was 3D printed using PLA 18 material which is lightweight and has considerable strength.

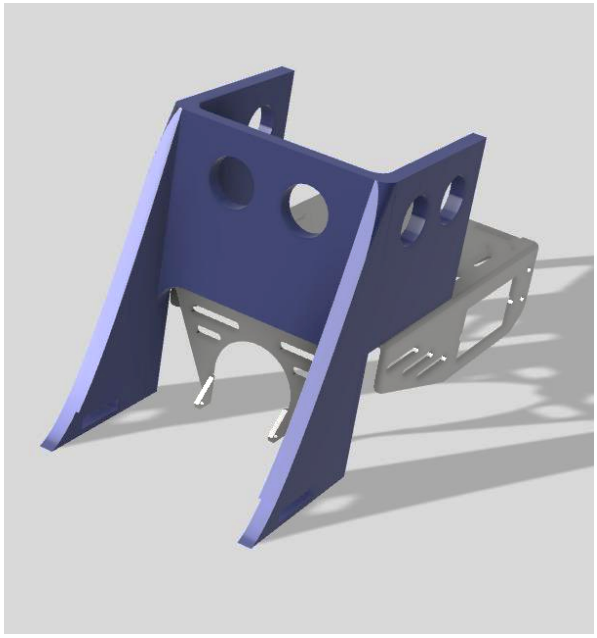


Figure 2: Isometric View

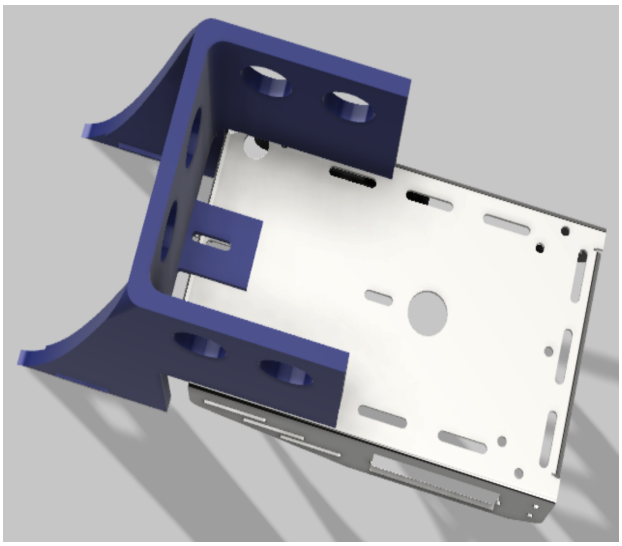


Figure 3: Top View

The Sensor Housing was aesthetically designed keeping in mind - aesthetic appeal and accurate positioning for the 2 HC-SR04 ultrasonic sensors and the IR sensor array in a combined housing design. The thickness and lengths of the design were optimized to provide the required strength and support. A groove is provided as shown in Figure 3 to mount onto the Boe-Bot Chassis. Two slots of thickness 4mm were provided at the bottom to hold the Infrared sensor array at an optimal height of 3mm from the ground.

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();
- 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;
else if check for obstacle at i3
intno=3;
else
intno=5;
Start obstacle indication cog2;
short push();
Left ultrasonic();
Left turn();
pause(25);
follower();
//Reached B1
if object at B1
{Start object indication cog3;}
short push();
push();
Right turn();
for(to move through B1 to B4)
{
Follow line;
Align ultrasonic;
if(object present at Bi)
{
Start object indication cog3;
}
}
//Reached B4
Right turn();
Follow Line;
//Reached A4
Center ultrasonic();
if object at A4
{Start object indication cog3;}
Short push();
Right turn();
for(to move through A4 to A1)
{
Follow line;
Align ultrasonic;
if(object present at Ai)
{Start object indication cog3;}
}
```

```
//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 object at B5
{Start object indication cog3;}
for(to move outside the grid)
{
Bot moves straight;
}
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)
{
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;
}
//------------------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
Propeller Activity Board	$105	1	$105
HC-SR04 Ultrasonic Sensor	$2	2	$4
Pololu QTR-8RC IR Sensor Array	$15	1	$15
Custom Designed Sensor Holder	$0	1	$0
Battery Pack	$7	2	$14
Duracell Battery	$1.375	8	$11
LED	$0.2	3	$0.6
Switch	$0	1	$0
Total Cost			**$249.6**

DEMONSTRATION

Google Drive Link:

Demo

The above is a demonstration video.

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 using the two ultrasonic sensors at each possible location.
5. The bot successfully stops after servicing all intersections.
6. Changing batteries can vary the speed and turning calibrations of the robot.
7. The bot successfully gives indications for results 2,3 & 4 using 3 LEDs
8. The bot doesn't have to stop anywhere during the process, by use of multiple cores.

APPENDIX - 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"

#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 p1 45 //push 1
#define p2 14 //push2 for short push
#define p3 44 //push 3
#define left 14
#define right 17
int inter();
int linloc();
void follower();
int cc();
int cl();
void push();
void puss();
void L90();
void R90();
```

```
void object(void *parc);
void isec(void *para);
void blink(void *parb);
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 c=0;
int l=0;
int r=0;
int f=0;
int dc;
int dl;
while(1)
{
int sum=inter();
if (sum < 18000 && sum > 2500)
{
int loc=linloc();
if (loc == 2 || loc == 3) //if true, robot moves straight
{
pulse_out(lm,2300);
pulse_out(rm,400);
pause(5);
}
else if (loc == 0 || loc == 1) //if true, robot moves left
{
//pulse_out(lm,1350);
pulse_out(rm,400);
pause(5);
l++;
}
else if (loc == 4 || loc == 5) //if true, robot moves right
{
pulse_out(lm,2300);
//pulse_out(rm,1350);
pause(5);
r++;
}
}
else if(c==3)
{
cog=cogstart(&isec, NULL, stack2, sizeof(stack2));
//////////////////////
push();
```

```
dc=cc();
if(dc<45)
intno=2;
else if(dc>45 && dc<70)
intno=3;
else
intno=5;
cog2=cogstart(&blink, NULL, stack1, sizeof(stack1));
spush();
dl = cl();
L90();
pause(25);
//spush();
follower();
//B1
if(dl<40)
{
cog3=cogstart(&object, NULL, stack3, sizeof(stack3));
}
spush();
push();
R90();
for(int bint=0;bint<3;bint++)
{
spush();
follower();
spush();
pause(100);
dl=cl();
if(dl<20)
{
cog3=cogstart(&object, NULL, stack3, sizeof(stack3));
}
}
//B4
spush();
spush();
R90();
push();
follower();
pause(50);
push();
follower();
push();
//A4
pause(25);
dc=cc();
if(dc<20)
{
cog3=cogstart(&object, NULL, stack3, sizeof(stack3));
}
```

```
spush();
R90();
for(int aint=0;aint<3;aint++)
{
spush();
follower();
spush();
pause(100);
brake();
dl=cl();
if(dl<20)
{
cog3=cogstart(&object, NULL, stack3, sizeof(stack3));
}
}
//A1
puss();
R90();
push();
pause(25);
follower();
push();
follower();
//B1
push();
R90();
for(int lastrun=0;lastrun<4;lastrun++)
{
follower();
spush();
}
//B5
dl=cl();
if(dl<15)
{
cog3=cogstart(&object, NULL, stack3, sizeof(stack3));
}
pause(400);
for(int x=0;x<=90;x++)
{
pulse_out(rm,400);
pulse_out(lm,2300);
pause(5);
}
party();
break;
}
else if(c==2)
{
cog=cogstart(&isec, NULL, stack2, sizeof(stack2));
c++;
```

```
push();
}
else if(c==1 && f==0) //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);
}
}
}



//------------functions--------------------
int inter()//returns threshold for intersection detection
{
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()// returns the location of the black line over the sensor array
{
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()// follows black line until an intersection is detected
{
int sum=0;
sum=inter();
while(sum < 18500 && sum > 2500)
{
int loc=linloc();
if (loc == 2 || loc == 3) //if true, robot moves straight
{
pulse_out(lm,2300);
pulse_out(rm,400);
pause(5);
}
```

```
else if (loc == 0 || loc == 1) //if true, robot moves left
{
//pulse_out(lm,1350);
pulse_out(rm,400);
pause(5);
}
else if (loc == 4 || loc == 5) //if true, robot moves right
{
pulse_out(lm,2300);
//pulse_out(rm,1350);
pause(5);
}
sum=inter();
}
cog=cogstart(&isec, NULL, stack2, sizeof(stack2));
}
//------------------ultrasonic----------------
int cl() // returns distance of nearest object in front of left sensor
{
brake();
//push();
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() // returns distance of nearest object in front of center sensor
{ 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() // a linear push
{
for(int x=0;x<=p1;x++)
{
pulse_out(rm,400);
pulse_out(lm,2300);
pause(5);
}
}
void puss() // a linear push
{
```

```
for(int x=0;x<=p3;x++)
{
pulse_out(rm,400);
pulse_out(lm,2300);
pause(5);
}
}
void L90() // turn 90 degrees left
{ for(int x=0;x<=left;x++)
{
pulse_out(rm,900);
pulse_out(lm,900);
pause(30);
}
}

void R90() // turn 90 degrees right
{ for(int x=0;x<=right;x++)
{
pulse_out(rm,2000);
pulse_out(lm,2000);
pause(30);
}
}
void brake() // single brake pulse
{
pulse_out(rm,1350);
pulse_out(lm,1350);
}
void object(void *parc) //led blink for object indication
{
//pulse_out(rm,1350);
//pulse_out(lm,1350);
pause(25);
high(2);
pause(500);
low(2);
cogstop(cog3);
}
void isec(void *para) // led blink for intersection indication
{
high(1);
pause(500);
low(1);
cogstop(cog);
}
void blink(void *parb) // led blink for obstacle location indication
{
for(int pav=0; pav<intno; pav++)
{
high(0);
```

```
pause(300);
low(0);
pause(300);
}
cogstop(cog2);
}
void party() // random blink sequence at the end
{
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() // a linear push
{
for(int x=0;x<=p2;x++)
{
pulse_out(rm,400);
pulse_out(lm,2300);
pause(5);
}
}
```