ROB-GY 6103: Advanced Mechatronics Propeller Project Report Team 25

KaiYuan Ma
Ifedayo Olusanya
Justin Ramlall

Table Of Contents

Introduction:

The goal of this project is to design an automated guided vehicle (AGV) that will move widgets from a random pickup location to a random dropoff location. This is accomplished with the use of a Parallax Propeller Activity Board WX. The Propeller board is a multicore microcontroller that allows us to control various devices simultaneously. To test our understanding of using the multicore, a list of objectives have been outlined for us to complete. A variety of devices will be used to help us achieve this, including infrared sensors, ultrasonic sensors, servos, light emitting diodes (LEDs), and a liquid crystal display (LCD). A complete circuit diagram will be included in this report detailing the electrical connections as well as a flowchart and commented code that detail the control logic. Through testing we have confirmed that the AGV completes all of the designated tasks.

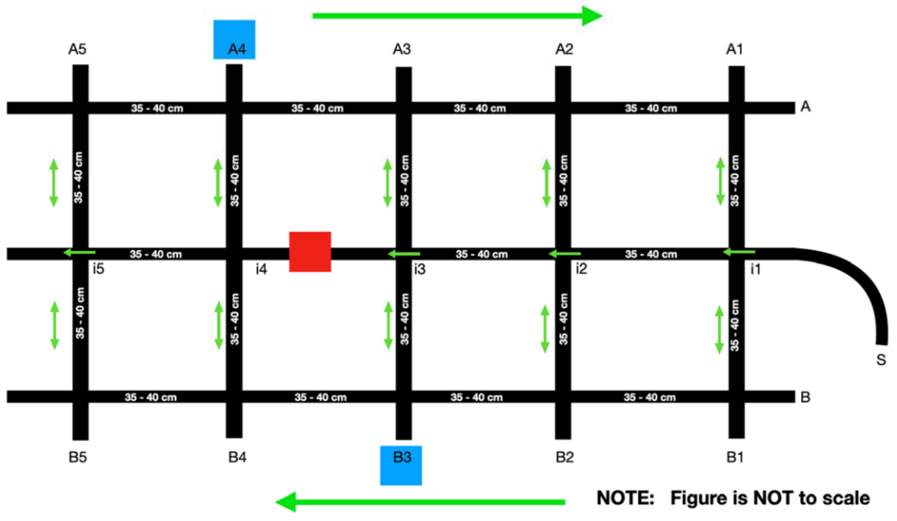

Figure 1: A diagram of the factory that the AGV has been designed to traverse. Lane S is the main traversal lane, while Lane A and Lane B are the pickup and dropoff lanes, respectively. Image courtesy of the ROB-GY 6103 instructional team.

Objectives:

In order to determine the hardware that will be used to create the AGV, we first need to understand the tasks for the AGV. The objectives designated for the AGV are as follows:

1. The AGVwill start at location S
2. The AGV will travel down the middle lane

3. The AGV will enter Lane A, show a decrease in speed, and stop at the randomly selected pickup location for 2 seconds
4. The AGV will then continue down to Lane B, where it will again show a decrease in speed, and then stop at the randomly selected dropoff location
5. The robot will display the distance it has driven from the pickup to drop off locations (in centimeters)

There are additional objectives that will be completed by the AGV as it moves along its path. These objectives are as follow:

1. The AGV will be following the black lines on the factory floor always
2. The AGV will indicate whenever it detects an intersection
3. The AGV will indicate when there is an obstacle blocking its path in the middle lane, and then travel along the lines to avoid it and continue on its path
4. The AGV will indicate when a dynamic obstacle is placed in front of it in either pickup or drop off lane, and stop moving until the dynamic obstacle has been removed

These objectives have all been demonstrated by the AGV in testing. Based on these objectives, we will be using the chassis and servos used in the Boe-Bot Kit from Parallax along with an infrared sensor array, two ultrasonic sensors, a LCD screen, and 2 LEDs.

System Design:

The system consists of a mechanical, electrical, and propeller code working in tandem to complete the objectives. The mechanical design will detail the components that were chosen for the AGV. The electrical design will detail the electrical connections for all of the components and the board and power supply. The propeller code will include a flowchart of the logic and a commented version of the code will be supplied in the report appendix.

Mechanical Design

Similar to the robot created for the Arduino Project, we are using the chassis provided in the Parallax Boe-Bot kit, along with the two Parallax Continuous Servos (Figure 2) and the wheel attachments. Our main microcontroller is the Parallax Propeller Activity Board WX (Figure 3) that will control all of our components. Also from Parallax, we are using the Parallax Ping))) Sensor (Figure 4) along with the Adafruit Industries LLC 3942 Ultrasonic Sensor (Figure 5) due to availability. For our indications, we will use two generic LEDs with a Parallax 2x16 Serial LCD with Piezo Speaker (Figure 6). To ensure proper line following, the Pololu QTR-8RC Reflectance Sensor Array (Figure 7) is used. All of these components are powered by two 9V batteries connected in series.

Figure 2: The continuous servo from Parallax. It is used to add motion to the AGV. Two of them are used with attached wheels.

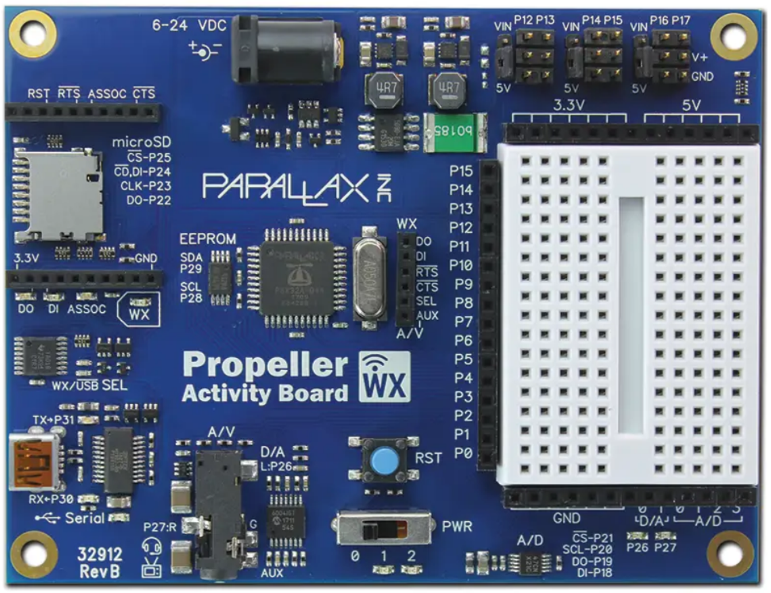

Figure 3: The Propeller Activity Board WX. This microcontroller controls all of our components. It has an attached breadboard for smaller circuits and 15 digital pins. It also has three headers to power motors and two onboard LEDs. It is powered by 18V DC.

Figure 4: The PING))) sensor from Parallax. It has 3 pins for 5V power supply, Ground, and SIG. SIG gets connected to a digital pin and is responsible for sending and receiving the ultrasonic signal.

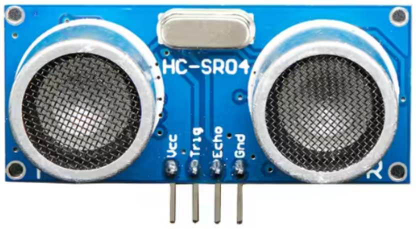

Figure 5: The Adafruit Industries LLC 3942 Ultrasonic Sensor. It has 4 pins for VCC power, Trigger, Echo, and Ground. Trigger and Echo send and receive the ultrasonic pulse. COnnecting them with a resistor allows this to function just like the PING))) sensor.

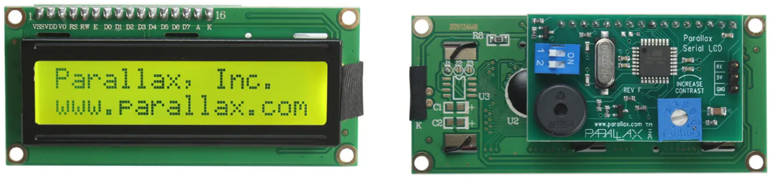

Figure 6: The Serial LCD from Parallax with two rows for displaying text. It has a rear header board that allows us to communicate with it using just three pins: 5V power supply, Ground, and RX for receiving serial data.

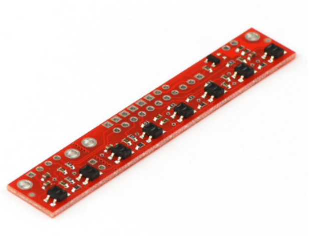

Figure 7: The Reflectance Sensor Array from Pololu. It has 8 Infrared Sensors. There is a pin for 5V power supply and Ground, and 8 pins for each infrared sensor.

Electrical Design

Below is the circuit diagram that details all of our electrical connections. Everything is powered directly from the board, which is powered with two 9V batteries in series. With just a single 9V battery, we did not see the AGV perform as expected, but adding a second battery in series showed an improvement in performance. The Pololu array is connected to digital pins P0 through P7. The Adafruit ultrasonic sensor is connected to P8 and the PING))) sensor is connected to P9. The two servos are connected to P16 and P17. As mentioned above, the Adafruit ultrasonic sensor has its trigger and echo pins connected to P8 with a 220 ohm resistor.

The two LEDs are connected to P14 for blue and P15 for red with the LCD connected to P12. Each LED has a 220 Ohm resistor connecting its positive end to the digital pin. Due to different voltage requirements, the PING))) sensor is connected to 3.3V supply, while all other components are connected to 5V supply. Each component has a connection to ground as well.

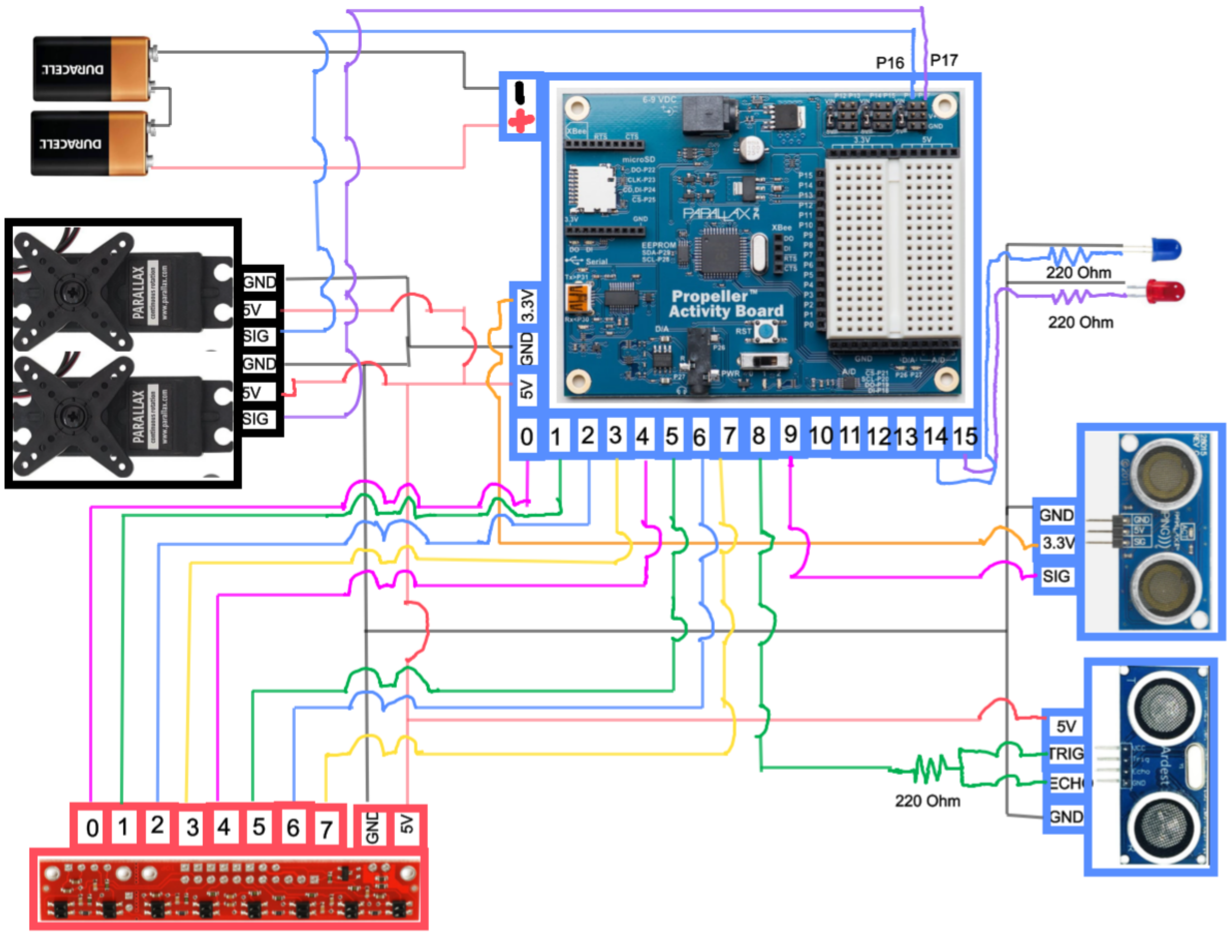

Figure 8: The circuit diagram for our AGV. All of the components and their electrical connections can be seen. The resistors connecting the LEDs and the trigger and echo pins of the Adafruit ultrasonic sensor are all labeled as 220 Ohm. The servos are directly powered by the Propeller board.

Propeller Code

The propeller code makes use of the multiple cogs on the board to complete tasks simultaneously. Below is a flowchart that details the logic for the code. There are five cogs that are used to complete these actions. The cogs include the main cog for the servos and LCD, a cog for the LEDs, a cog for the infrared sensor array, and a cog each for the two ultrasonic sensors. We start by calibrating the infrared array, and then go into line following. The AGV will move along the middle lane. If it detects an object, it will move to avoid the object. When the counter for the intersections in the middle lane reaches 5 (the designated number of intersections for this map), then the AGV will go into the pickup lane. Here, the ultrasonic sensors work to determine if there is either a pickup location on the left or if there is a dynamic object in front of it. If there is a pickup location, it will stop at the location for 2 seconds. If there is a dynamic object, it will stop until the object is moved, checking every 5ms if the object is still there. Once the counter for the pickup lane indicates that we are at the end of the lane, we then move towards the drop off lane. Similar to the pickup lane, we check for drop off locations and for dynamic obstacles. Once we stop at a drop off location, the LCD screen reads the distance that was traveled, which is calculated by counting the intersections from pickup to drop off and multiplying by 40cm, the code is done. Also, throughout the traversal, the blue LED will blink for an intersection and the red LED will blink for object indication.

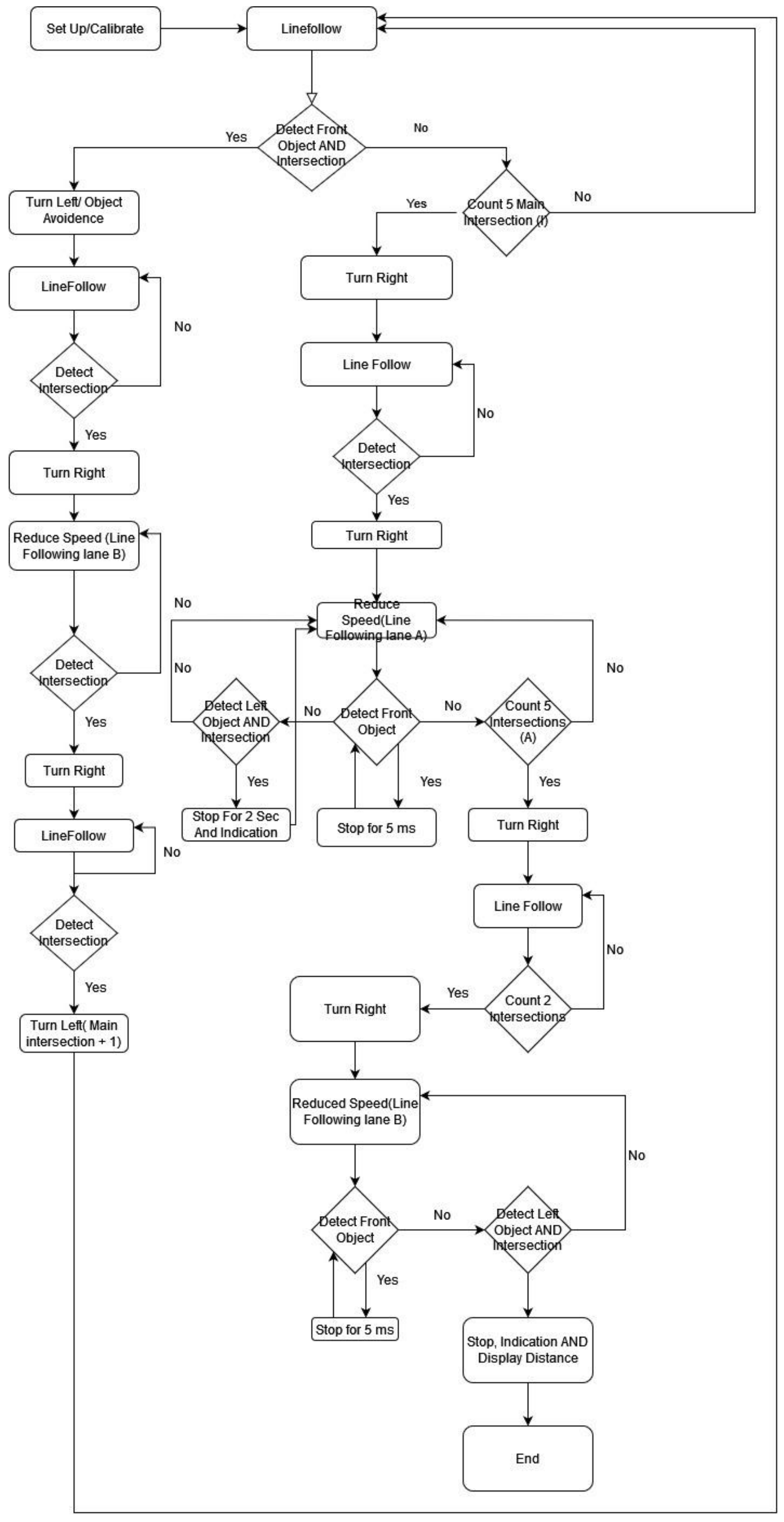
Set Up/Calibrate
Linefollow
Detect Front Object AND Intersection
Yes
No
Turn Left/ Object Avoidence
Count 5 Main Intersection (I)
Yes
No
LineFollow
Turn Right
No
Detect Intersection
Line Follow
Yes
No
Turn Right
Detect Intersection
Yes
Reduce Speed (Line Following lane B)
Turn Right
No
Reduce Speed(Line Following lane A)
Detect Intersection
No
Detect Left Object AND Intersection
No
Detect Front Object
No
Count 5 Intersections (A)
No
Yes
Yes
Yes
Yes
Turn Right
Stop For 2 Sec And Indication
Stop for 5 ms
Turn Right
LineFollow
No
Detect Intersection
Line Follow
No
Yes
Yes
Turn Left(Main intersection + 1)
Turn Right
Count 2 Intersections
Reduced Speed(Line Following lane B)
No
Detect Front Object
No
Detect Left Object AND Intersection
Yes
Stop for 5 ms
Stop, Indication AND Display Distance
End

Figure 9: This flowchart details the logic used for the propeller code. It includes instructions for how the AGV travels as well as what to do when it meets intersections and obstacles, as well as when to switch what lane it is traveling in.

System Testing:

In order to test complete functionality of the AGV, we took it step by step. First, we coded the AGV to traverse the entire play map. This was done using counters, creating functions for the servos to turn left and right, and a function for the infrared array to read the lines. Once the AGV was line following correctly, the next thing that we needed to do was add the ultrasonic sensors for object detection. This was simple, and mostly just using if statements. When there is an obstacle in the middle lane, a predetermined route to go around it is run. When the AGV returns to the middle lane, it again detects if a second obstacle is there in case it needs to be avoided as well. For the pickup and drop off locations, if the ultrasonic sensor on the side of the robot detects an object and the infrared array detects an intersection, the AGV will stop. For the dynamic obstacles in lanes A and B, when the front ultrasonic sensor detects an object then the AGV will stop, check every 5ms if the object is still there, and when the object is removed then the AGV will continue. Once we confirmed that all of this is functioning correctly, then the last part of our testing was just to include the indications for objects and intersections as well as the indication for the distance traveled at the end. This was the simplest part and once it was functionally, then the AGV was complete. From there we just conducted more testing to fine tune some variables and ensure the AGV was in its best operating condition.

Conclusion:

The goal of this project was to understand the use of a multicore microcontroller in order to traverse the map presented in Figure 1. Once all of the objectives were laid out, we picked out what we believed to be the best components available now to assemble our AGV. Once it was assembled, we began the testing process to program it. After this testing process was completed, we continued fine tuning the AGV until it was working as desired. The system design and testing process resulted in a robust AGV that allows us to complete all of the designated tasks.

Appendix:

```
#include "simpletools.h" // Include simpletools
#include "servo.h" // Include servo library
#include "ping.h" // Include Ultrasonic Sensor
serial *lcd; // Connect to LCD
void forward(void); //forward declaration of function to move robot forward at fastest speed
void forward_slow(void);//forward declaration of function to move robot forward at slower speed
void stopmotors(void); //forward declaration of function to stop robot
void turnright(void); //forward declaration of function to turn robot
void turnleft(void); //forward declaration of function to turn robot left
void spinleft(void); //forward declaration of function to spin robot left
void spinright(void); //forward declaration of function to spin robot right
void linefollow(void);//forward declaration of function to enable robot line follow at fastest speed
void linefollow_slow(void); //forward declaration of function to enable robot line follow at slower
speed
void calibrate(void *par1); //forward declaration of function to calibrate the Pulolo reflectance
sensor
void pingfromcog(void *par2); //forward declaration of function to read obstacle in front of robot
void pingfromcog2(void *par3);//forward declaration of function to read obstacle at the left of
robot
void led(void *par4); //forward declaration of function to blink the different LEDs
static volatile int blink; //variable that controlls the led we blink
volatile int cmdist; // distance reading for front ultrasonic sensor
volatile int cmdist2;// distance reading for left ultrasonic sensor
volatile int position, a; //position helps tell us where the black line is during line follow, while a
helps us control when to calibrate and when to read position values
volatile int intersection; //this helps us determine when we are at an intersection
volatile float scaled_sensor_val[8]; //stores the final reading of sensor values
unsigned int stack4[40 + 25]; // controls the first cog
unsigned int stack1[40 + 95]; // controls the second cog
unsigned int stack2[40 + 25]; //controls the third cog
unsigned int stack3[40 + 25]; //controls the 4th cog
int distance; //variable to determine total distance travelled between dropoff and pick up
int dx; // variable to control a counter
int movecount; // variable to control the number of loops the go into during motor actuation
int maininter; // counter for the center lane
int mainobstacle; // obstacle variable for the main lane
int binter; // variable for b lane during delivery
int bcounter; // counter for b lane during delivery
int acounter; // counter for a lane during delivery
int ainter; // variable for a lane
int obstacleb; // variable for b lane during obstacle avoidance
const int ON = 22; // turn on LCD
const int CLR = 12; // clear LCD screen
volatile int sensor_mean; // intersection variable
int main() // main function
{
a = 1; // begin calibration
cogstart((void*)calibrate, NULL, stack1, sizeof(stack1));// start up cog 1 and run calibrate
function
pause(2000); // wait 2 seconds
movecount = 30;
spinright(); // spin ring
movecount = 60;
```

```
spinleft(); // spin left
movecount = 30;
spinright(); // spin right
stopmotors(); //stop motors
a=2; // stop calibration and just keep reading position value
//initial declaration of counters and other variables
maininter =0;
binter = 0;
ainter = 0;
mainobstacle =0;
int bcounter =0;
int acounter =0;
int obstacleb =0;
dx =0;
distance =0;
//start ultrasonic cogs (2 and 3)
cogstart((void*)pingfromcog, NULL, stack2, sizeof(stack2)); // start second cog
cogstart((void*)pingfromcog2, NULL, stack3, sizeof(stack3)); // start third cog
//start led cog
blink =0; // variable to turn off all leds
cogstart((void*)led, NULL, stack4, sizeof(stack4)); // start cog 4 and run led function
lcd = serial_open(12, 12, 0, 9600); // start lcd
writeChar(lcd, ON); // ON
writeChar(lcd, CLR); //Clear screen
pause(5); // wait for 0.005 seconds
while(1){
//line follow function on the center lane. if an obstacle is seen in front of robot, go into the B
obstacle avoidance line following sequence
if(sensor_mean <300&&maininter >=0&&binter ==0 && ainter ==0){
do{
if(cmdist==1 && maininter <4){ // if we arent at the end of main lane and an obstacle is seen
linefollow(); // follow the line
blink = 3; // blink both Leds when an obstacle is seen
mainobstacle = 1; // indicate that an obstacle is seen
binter =1; // enter obstacle avoidance sequence
}
else{
linefollow();
}
}while(sensor_mean <300 && binter ==0);
}
// avoid obstacle by going into the b lane then go back into the center lane. when robot gets
back to center lane check i an obstcale is in front
// if an obstacle is in front, go back to avoid obstcale and return back to center lane
```

```
else if(sensor_mean <300&&maininter >=0&&binter ==1&& ainter ==0&&(obstacleb == 1
||obstacleb == 2||obstacleb ==3||obstacleb ==4)){
do{
if (cmdist ==1 && obstacleb == 2){
movecount = 20;
blink = 3; // blink both Leds when an obstacle is seen
stopmotors(); //stop motors
}
else if(obstacleb ==2){
linefollow_slow(); //line follow slowly
}
else if(cmdist ==0 && obstacleb ==4){
binter =0;
obstacleb = 0;
maininter +=1;
mainobstacle =0;
}
else if(cmdist ==1 &&obstacleb ==4){
movecount = 20;
stopmotors();
pause(50);
spinleft();
pause(170);
obstacleb = 1;
maininter +=1;
}
else{ linefollow();
}
}while(sensor_mean <300 && binter ==1);
}
//sequence to control line follow process while on the pickup lane. if an obstacle is seen at a
pick up location, pause for 2seconds to pickup
else if(sensor_mean <300 && maininter >=0 && binter == 0 && ainter ==1 &&(acounter>=0)){
do{
if(acounter ==0){
linefollow();
}
else if (acounter>=1 && cmdist ==1){
movecount = 20;
blink = 3; // blink both Leds when an obstacle is seen
stopmotors();
}
else if(acounter >=1){
linefollow_slow();
}
}while(sensor_mean <300 && binter ==0 && ainter ==1);
}
//sequence to control line follow process while robot is in delivery sequence
//check for obstacle at the drop off points. if an obstacle is seen, stop robot and display distance
travelled
else if(sensor_mean <300 && binter == 2){
do{
if(bcounter >=3 && cmdist ==1){
movecount = 20;
blink =3; // blink both Leds when an obstacle is seen
stopmotors();
}
else if(bcounter>=3){
linefollow_slow();
}
else{
linefollow();
}
}while(sensor_mean <300 && binter ==2);
}
// Sequence to control behaviour while at intersections
else if(sensor_mean>300)
{
if(mainobstacle ==1 &&(obstacleb==0)){// turn left so the robot approaches the b lane once an
obstacle is seen at the front of the robot
maininter +=1;
obstacleb +=1;
movecount = 20;
blink = 1; // blink red Led when an obstacle is seen
turnleft();
pause(500);
}
else if(mainobstacle ==1 && (obstacleb ==1||obstacleb==2)){// turn rght and enter the b lane
then at the next intersection turn right and approach the center lane
obstacleb +=1;
movecount = 20;
blink =1; // blink red Led when an obstacle is seen
turnright();
pause(500);
}
else if(maininter <4 && obstacleb ==3){// if robot is still in obstacle avoidance and it's not at the
last center lane intersection, turn robot right to enter the center lane
movecount =20;
```

```
blink =1; // blink red Led when an obstacle is seen
forward();
pause(10);
spinleft();
pause(200);
obstacleb +=1;
}
else if(maininter ==4 && obstacleb ==3){ // if robot is still in obstacle avoidance and it's at the
last intersection of the center lane,move forward and approach the a lane
movecount =20;
blink = 1; // blink red Led when an obstacle is seen
forward();
binter =0;
mainobstacle =0;
ainter +=1;
maininter +=1;
obstacleb = 0;
}
else if(maininter ==4 && obstacleb ==0){ // if robot is not in obstacle avoidance, turn left at the
last intersection of center lane
ainter +=1;
binter =0;
obstacleb = 0;
mainobstacle =0;
movecount = 20;
maininter +=1;
blink = 1; // blink red Led when an obstacle is seen
turnright();
pause(500);
}
else if(ainter ==1 && acounter ==0 && cmdist ==1){// if obstacle present in first pickup location,
pause for 2sec and turn right
dx = (4-acounter);
acounter +=1;
movecount = 20;
blink = 1; // blink red Led when an obstacle is seen
blink = 2; // blink blue Led when an obstacle is seen
stopmotors();
pause(2000);
turnright();
pause(500);
}
else if(ainter ==1 && acounter ==0){ // if no obstacle present in first pickup location, turn right
acounter +=1;
```

```
movecount = 20;
blink = 1; // blink red Led when an obstacle is seen
turnright();
pause(500);
}
else if(ainter ==1 && (acounter==1 || acounter ==2 || acounter ==3)){//check each intersection
on the a lane, if there is an obstacle pause for 2 sec and move forward
blink =1; // blink red Led when an obstacle is seen
movecount = 20;
forward_slow();
pause(100);
if(cmdist2 ==1){
dx =(4-acounter);
acounter +=1;
movecount = 20;
blink = 2; // blink blue Led when an obstacle is seen
stopmotors();
pause(2000);
forward_slow();
}
else
{
acounter +=1;
movecount = 20;
forward_slow();
}
}
else if(ainter ==1 && acounter==4){ // if at the last intersection of a lane, check if there is an
obstacle. if there is an obstacle pause for 2secs and turn right. if no obstacle turn right
blink = 1; // blink red Led when an obstacle is seen
movecount = 20;
forward_slow();
pause(100);
if(cmdist2 ==1)
{
dx =(4-acounter);
binter =2;
acounter +=1;
bcounter +=1;
movecount = 20;
blink = 2; // blink blue Led when an obstacle is seen
stopmotors();
pause(2000);
spinright();
```

```
pause(200);
}
else{
binter =2;
acounter +=1;
bcounter +=1;
movecount = 20;
spinright();
pause(200);
}
}
else if(binter ==2 && (bcounter==1)){// move forward until you are approaching the b lane
bcounter +=1;
movecount = 20;
blink = 1; // blink red Led when an obstacle is seen
forward();
}
else if(binter ==2 && (bcounter==2) && cmdist ==1){ // when at the b lane check if there is a
drop off at first point. if there is an obstacle just pause
distance = (dx+bcounter)*40;
bcounter +=1;
movecount = 20;
blink = 1; // blink red Led when an obstacle is seen
while(1){
dprint(lcd, "distance = %d ", distance); //print distance travelled to lcd
stopmotors();
writeChar(lcd, CLR);
blink = 2; // blink blue Led when an obstacle is seen
}
}
else if(binter ==2 && (bcounter==2)){// if there is no obstacle at first drop off point, turn right
bcounter +=1;
movecount = 20;
blink = 1; // blink red Led when an obstacle is seen
turnright();
pause(500);
}
else if(binter ==2 && (bcounter>=3 && bcounter<6)){ // at ever other drop off location. if there is
no drop off point, move forward. Else pause.
blink =1; // blink red Led when an obstacle is seen
movecount = 20;
forward_slow();
pause(100);
if(cmdist2 ==1){
```

```
distance = (dx+bcounter)*40;
bcounter +=1;
movecount = 20;
dprint(lcd, "distance = %d ", distance); //print distance travelled to lcd
while(1){
stopmotors();
blink = 2; // blink blue Led when an obstacle is seen
}
}
else{
bcounter +=1;
movecount = 20;
forward_slow();
}
}
else if(binter ==2 && (bcounter==6)){ //when at the last intersection of b lane, stop.
blink =1; // blink red Led when an obstacle is seen
movecount = 20;
forward_slow();
pause(50);
if(cmdist2 ==1){
distance = (dx+bcounter)*40;
bcounter +=1;
movecount = 20;
dprint(lcd, "distance = %d ", distance); //print distance travelled to lcd
while(1){
stopmotors();
blink = 2; // blink blue Led when an obstacle is seen
}
}
else{
bcounter +=1;
movecount = 20;
stopmotors();
pause(5000);
}
}
else{ // ever other situation, just blink and add one to the main intersection
maininter +=1;
movecount = 20;
blink = 1; // blink red Led when an obstacle is seen
forward();
}
}
```

```
}
}
//function to calibrate ultrasonic sensor and read position of blackline
void calibrate(void *par1){
// local variables to hold sensor reading
int cal_ini_sensor_val[8];
int cal_sensor_val[8];
int max_cal[8];
int min_cal[8];
int cal_value;
int sensor_ini_val[8];
int sensor_val[8];
int sensor_scale_sum;
// charge all IR sensors of the pulolo, get the RC times and initialize the first reading of each
sensor to those values
set_directions(7, 0, 0b11111111); // P7...P0 -> output
set_outputs(7, 0, 0b11111111);
pause(10);
for(int i=0;i<8;i++){
cal_ini_sensor_val[i] = rc_time(i,1);
max_cal[i] = cal_ini_sensor_val[i];
min_cal[i]= cal_ini_sensor_val[i];
}
while(a==1)
{ // calibrate sequence
// while turning the robot left and right,read rc time of each sensor multiple times and get their
max and min values
set_directions(7, 0, 0b11111111); // P7...P0 -> output
set_outputs(7, 0, 0b11111111);
pause(10);
for(int i=0;i<8;i++){
cal_sensor_val[i] = rc_time(i,1);
}
for(int i=0;i<8;i++){
if(min_cal[i] >cal_sensor_val[i]){
min_cal[i]= cal_sensor_val[i];
}
else if (max_cal[i]<cal_sensor_val[i]){
max_cal[i] = cal_sensor_val[i];
}
}
}
while(a==2){
```

```
// after calibration keep taking rc time, scale each senor values to a value between 0 and 1000,
use the weighted function to determine the position
set_directions(7, 0, 0b11111111); // P7...P0 -> output
set_outputs(7, 0, 0b11111111);
pause(10);
for(int i=0;i<8;i++){
sensor_ini_val[i] = rc_time(i,1);
sensor_val[i] = sensor_ini_val[i];
if(sensor_val[i]< min_cal[i]){
sensor_val[i] = min_cal[i];
}
else if (sensor_val[i] >max_cal[i]){
sensor_val[i] = max_cal[i];
}
}
sensor_scale_sum = 0;
for(int i=0;i<8;i++){
//conduct an interpolation to enable scaling down. Similar to the map function
float A = (float)(max_cal[i] -sensor_val[i]);
float B =(float) (max_cal[i] - min_cal[i]);
float C = (A/B);
scaled_sensor_val[i] = 1000 -(1000*(C));
sensor_scale_sum += scaled_sensor_val[i];
}
sensor_mean = (int)(sensor_scale_sum/8);
float D = 0*scaled_sensor_val[0]+
1000*scaled_sensor_val[1]+2000*scaled_sensor_val[2]+3000*scaled_sensor_val[3]+4000*scal
ed_sensor_val[4]+5000*scaled_sensor_val[5]+6000*scaled_sensor_val[6]+7000*scaled_sensor
_val[7];
float E = scaled_sensor_val[0]+
scaled_sensor_val[1]+scaled_sensor_val[2]+scaled_sensor_val[3]+scaled_sensor_val[4]+scale
d_sensor_val[5]+scaled_sensor_val[6]+scaled_sensor_val[7];
position = (int)(D/E); // weighted average
}
}
// line follow and adjust position of robot based on blackline reading
void linefollow(void){
if (position < 2800) {
do {
movecount = 1;
turnright();
} while (position < 2800);
} else if (position > 4200)
do {
```

```
movecount = 1;
turnleft();
} while (position > 4200);
movecount = 2;
forward();
}
// move robot forward by turning both motor in opposite directions at about 80% of the full speed
void forward(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,80);
servo_speed(17,-80);
pause(5);
}
}
// stop robot by stopping both motors
void stopmotors(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,0);
servo_speed(17,0);
pause(20);
}
}
// turn robot right by stopping right motor and turning left motor forward
void turnright(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,95);
servo_speed(17,0);
pause(20);
}
}
// turn robot right by stopping left motor and turning right motor forward
void turnleft(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,0);
servo_speed(17,-95);
pause(20);
}
}
// spin robot right by turning both motors forward
void spinright(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,95);
servo_speed(17,95);
pause(20);
```

```
}
}
// spin robot left by turning both motors backwards
void spinleft(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,-95);
servo_speed(17,-95);
pause(20);
}
}
// read distance with front untra sonic sensor then convert it to an obstacle true or false.
void pingfromcog(void *par2) {
int a;
while(1)
{
a =ping_cm(8);
if(a>30
){
cmdist = 0;}
else{
cmdist = 1;
}
}
}
// read distance with left ultrasonic sensor then convert it to an obstacle true or false.
void pingfromcog2(void *par3) {
int a;
while(1)
{
a =ping_cm(9);
if(a>30){
cmdist2 = 0;}
else{
cmdist2 = 1;
}
}
}
// same at forward function but at about 40% of full speed
void forward_slow(void) {
for (int i = 0; i < movecount; i++) {
servo_speed(16,40);
servo_speed(17,-40);
pause(5);
}
```

```
}
// same as linefollow function but at the slower speed
void linefollow_slow(void){
if (position < 2800) {
do {
movecount = 1;
turnright();
} while (position < 2800);
} else if (position > 4200)
do {
movecount = 1;
turnleft();
} while (position > 4200);
movecount = 2;
forward_slow();
}
// Led function running on cog 4
void led(void *par4){
while(1){
//whenever blink is 0 turn off both leds
while(blink ==0){
low(14);
low(15);
pause(10);
}
while(blink ==1){ //when blink is one turn on red led, wait for 0.2 seconds and turn it off
high(14);
pause(200);
blink =0;
}
while(blink ==2){ //when blink is two turn on blue led, wait for 0.2 seconds and turn it off
high(15);
pause(200);
blink =0;
}
while(blink ==3){ // when blink is three turn on both leds, wait for 0.2 seconds and turn it off
high(14);
high(15);
pause(200);
blink =0;
}
}
}
```