

*All work in the report is done by the listed authors, with the exception of properly cited material.*

Final Report

Yujin Cho 20060040

John Xia 20067226

Professor­­ Brian Frank

April 5, 2019

ELEC 299

# Executive Summary

The robot competition project is a simulation for an autonomous robotic warehouse. Three robot prototypes work together to collect objects from three walls and navigate back to the drop-off point to unload the objects. Prototyping the robot involves hardware and software skills developed in the first half of the course ELEC 299. Throughout the project, processes have been designed, tested, and iterated numerous times to improve robot performance and overcome challenges. This has led to overall success defined by the functional specifications that were initially outlined in the Proposal Report. The team worked effectively together.

Table of Contents

[Executive Summary i](#_Toc5362232)

[Proposed Specifications 1](#_Toc5362233)

[Original Pseudocode 1](#_Toc5362234)

[Modifications to Pseudocode 1](#_Toc5362235)

[Hardware Schematic 2](#_Toc5362236)

[Performance Analysis 2](#_Toc5362237)

[Team Evaluation 3](#_Toc5362238)

[Appendix A: Final Code 4](#_Toc5362239)

# List of Figures

[Figure 1: Flowchart for the robot's decision making 1](#_Toc5370020)

[Figure 2: Schematic diagram of hardware connections 2](#_Toc5370021)

## Proposed Specifications

The initial functional requirements were to accurately navigate the playing field to collect the objects while avoiding collisions. To do so, it was proposed to have a hardcoded set of instructions (specific to each starting position) that could be loaded into an array that the robot would read and execute. The basic steps to follow were to first localize the robot to get in front of the object, then lower and close the gripper servos to grab the object, then turn around and follow the gridlines back to the goal bin where the object would drop.

## Original Pseudocode

The original pseudocode was structured to have two while loops, the first one would check for the IR beacon signal and the second one would read though all the elements within the instruction array and call the appropriate function. The specifics of each of these functions were not discussed in the proposal as code completed for labs 1-3 were expected to be directly implemented. The original instruction set selection method depending on starting point was designed to copy the selected array directly into a useable one. For navigation, the robot relied on encoder counts for total distance travelled and the line-tracker sensors to maintain a straight course. The basic flow for the software is shown in Figure 1.

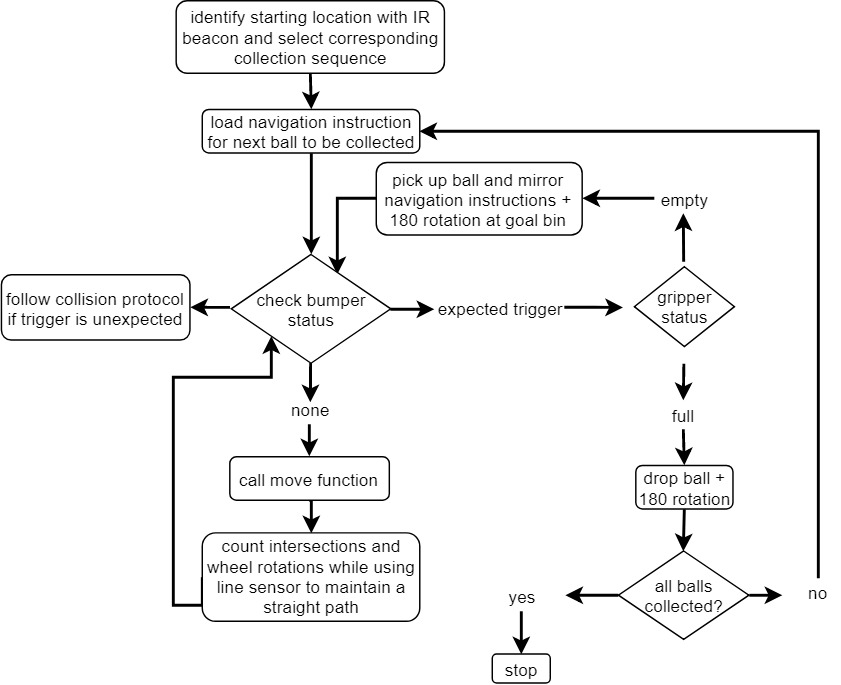


Figure : Flowchart for the robot's decision making

## Modifications to Pseudocode

Depending purely on the encoders for navigation was risky due to calibration limitations and performance variance of the robot. Therefore, navigation became dependant on a combination of the encoders and line-tracker sensors. As reading the encoder value would return a high or low, depending on the latency of the digital reads within the while loop, transitions between values could be missed and the encoder would miss a wheel count increment. Therefore, to reduce the delay for the encoder read, a while loop without the line-tracker sensors being read was decided as additional lines of code would prove to make the encoder reads less precise. Ultimately the encoder was used as a delay for the line-tracker sensors since with the encoder would delay the line-tracker sensors when activated such as to not count additional intersections or stop turning prior to the desired position.

As maintaining the robots to a strict schedule was the preferred method for avoiding collisions, a grip sensor was irrelevant as the robot was to have a single attempt at collecting the dice. The bumper status was only checked during the forward function and would exit the function if activated. There was no collision avoidance code within the final copy of the code except for a full minute delay built in to allow for the robots to cross paths with larger windows to avoid collisions.

The forward() function begins by reading each individual line-tracker sensor then reading appropriate if statements to either count an intersection, exit the function, or correct its own path. If the sensor reads an intersection, the encoder() function is called and delays the program until a threshold of transitions are reached. This allows for an intersection to be counted once and not repeated based on line-tracker thresholds. An alternative was to hardcode a time-delay, however that would change based on the speed of the robot which would be less precise than the encoders.

Left(), right(), spin(), spinnyR(), and spinnyL() are all turning functions that incorporate an initial encoder delay as to skip over a specific number of lines to turn the required number of degrees.

## Hardware Schematic

Figure 2 below shows the updated schematic diagram.

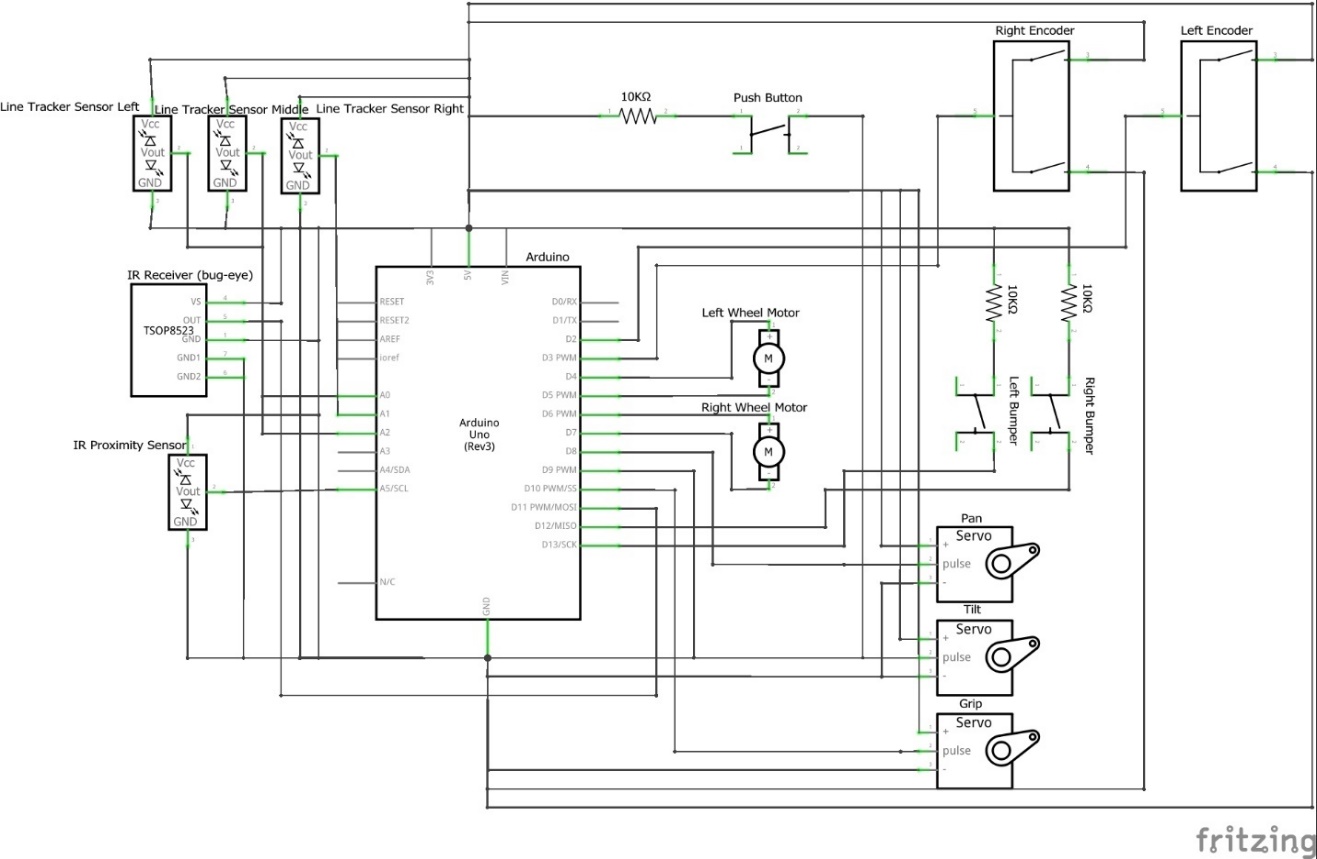


Figure : Schematic diagram of hardware connections

The original hardware schematic was amended to include the three different line tracking sensors instead of having one symbol represent all of them, and the force-gripping sensor was removed because it was deemed unnecessary for the functioning of the robot. This gives a more accurate representation of how the hardware is organized.

## Performance Analysis

The driving and object retrieval functions of the robot worked well during the competition; however, the collision avoidance strategy could have been better. Following instructions from the predetermined arrays worked with high consistency throughout the testing phases and worked as expected during the competition. The problems that were encountered with the collision avoidance strategy could have been avoided had the teams allocated more time for testing the routes with all three robots at the same time. Due to unexpected hardware issues in the two other teams, the schedule for doing so was set back, and the testing room was very crowded during the last few days before the competition.

Having the robot start with the IR beacon was also a minor issue since the teams did not get much time to test with them following lab 5, when iterations were still being done for the navigation and retrieval systems. There was troubleshooting involved with the beacons during the ten minutes before competing, which was not ideal, but the issue was resolved in time.

## Team Evaluation

The original project plan was to initially write out functions that verify that the hardware works correctly, then adapt those functions and iterate on the design of the robot system to ultimately carry out an array of instructions that depend on the starting location. The hardware was all wired properly and tested with initial code by March 20th, which left ample time for more versions of the code to be drafted and refined. It was agreed to join the other two teams to test the collision avoidance strategy on Saturday March 30th, by which time the driving, pickUp and dropOff functions were finished and the instruction arrays had been populated. Issues arose with the other team’s robots which led to limited testing on Saturday, and the collision avoidance testing had to be postponed until the following Monday night, which gave us just two weekdays to finish up testing together and led to underwhelming results during the competition.

Teamwork can be evaluated on four metrics: attendance, efficiency, initiative, and quality. Attendance for both members was satisfactory to get a meeting’s goals accomplished. Efficiency can be attributed to how well team members stayed on task during meetings. Both team members were sufficiently focused on the tasks at hand, until frustration with the robot functionality made it necessary to take a break. Both team members showed initiative when volunteering to complete specific tasks, organizing meetings, and communicating with the other teams to test the collision avoidance strategy. The quality of the work that was completed as a team was reasonably good, since the hardware wiring and coding that was done was all functional and consistent throughout the testing period.

Overall, the team worked well together on the project even though it was short one member compared to other teams. One way to improve some of the work processes would be to create a to-do list and have an agenda for each of the meetings, so everything would be well-organized beforehand rather than figuring out what needs to be done during the meeting. Another area for improvement could be decision-making. The team agreed too quickly on one approach before finding out that there would be an issue somewhere. An evaluation matrix could have been used to weigh the pros and cons of one option over another, thinking everything through more thoroughly before carrying out an idea.

## Appendix A: Final Code

#include "QSerial.h"

#include <Servo.h>

#include <String.h>

QSerial myIRserial;

//pin definitions

#define Lencoder 2

#define Rencoder 3

#define Ldirection 4

#define Lspeed 5

#define Rspeed 6

#define Rdirection 7

#define sensor 11

#define Rbumper 12

#define Lbumper 13

#define rightSensor A1

#define middleSensor A2

#define leftSensor A0

//one wheelturn is 20

//one intersection is 4

//encoder count calibrations

#define backup 5

#define spinny 18

#define turn 10

#define intersection 4

#define wheelTurn 30

//motor speed calibration

#define Lvroom 170

#define Rvroom 162

#define MAX 225

#define rotate 10

//line sensor calibration

#define Lthresh 910

#define Mthresh 910

#define Rthresh 910

//declare variables for encoder function

int prev\_L = 0;

int prev\_R = 0;

int next\_L = 0;

int next\_R = 0;

Servo grip, tilt, pan;

//array for number of intersections to drive forwards

int mag[3][54] = {{3,0,2,0,0,1,0,3,0,

0,1,0,4,0,0,3,0,1,0,0,

0,1,0,1,0,5,0,0,4,0,1,0,1,0,

0,4,0,2,0,0,1,0,4,0,

0,5,0,2,0,0,1,0,5,0},

{2,0,3,0,0,2,0,2,0,

0,2,0,3,0,0,2,0,2,0,0,

0,6,0,0,6,0,

0,5,0,1,0,1,0,0,1,0,5,0,

0,5,0,1,0,1,0,0,1,0,5,0,0,0,0,0},

{1,0,4,0,0,3,0,1,0,

0,3,0,2,0,0,1,0,3,0,0,

0,1,0,1,0,5,0,0,4,0,1,0,1,0,

0,5,0,2,0,0,1,0,5,0,

0,4,0,2,0,0,1,0,4,0}};

//instruction array

char act[3][54] = {{'f','l','f','p','s','f','r','f','d',

's','f','r','f','p','s','f','l','f','d','t',

's','f','l','f','r','f','p','s','f','l','f','r','f','d',

's','f','l','f','p','s','f','r','f','d',

's','f','l','f','p','s','f','r','f','d'},

{'f','l','f','p','s','f','r','f','d',

's','f','r','f','p','s','f','l','f','d','t',

's','f','p','s','f','d',

's','f','l','f','r','f','p','q','f','r','f','d',

's','f','r','f','l','f','p','w','f','l','f','d','x','x'},

{'f','l','f','p','s','f','r','f','d',

's','f','r','f','p','s','f','l','f','d','t',

's','f','r','f','l','f','p','s','f','r','f','l','f','d',

's','f','r','f','p','s','f','l','f','d',

's','f','r','f','p','s','f','l','f','d'}};

void setup() {

prev\_L = digitalRead(Lencoder);

prev\_R = digitalRead(Rencoder);

grip.attach(10);

tilt.attach(9);

pan.attach(8);

grip.write(40);

tilt.write(165);

pan.write(70);

pinMode(Ldirection,OUTPUT);

pinMode(Lspeed,OUTPUT);

pinMode(Rdirection,OUTPUT);

pinMode(Rspeed,OUTPUT);

pinMode(Lencoder,INPUT);

pinMode(Rencoder,INPUT);

pinMode(Lbumper,INPUT);

pinMode(Rbumper,INPUT);

pinMode(sensor, INPUT);

myIRserial.attach(sensor, -1);

Serial.begin (9600);

}

int val = -1;

void loop()

{

int location = getlocation();

for ( int i=0;i<54;i++)

{

char alpha = act[char(val)][i];

if (alpha=='f')

forward(mag[char(val)][i]);

else if(alpha=='l')

left();

else if(alpha=='r')

right();

else if(alpha=='p')

pickup();

else if(alpha=='d')

dropoff();

else if(alpha=='s')

spin();

else if(alpha=='q')

spinnyL();

else if(alpha=='w')

spinnyR();

else if(alpha =='t')

delay(60000);

}

}

void forward(int magnitude)

{

int count = 0;

digitalWrite(Ldirection,HIGH);

digitalWrite(Rdirection,HIGH);

while(count < magnitude)

{

int L = analogRead(leftSensor);

int M = analogRead(middleSensor);

int R = analogRead(rightSensor);

analogWrite(Lspeed,Lvroom+25);

analogWrite(Rspeed,Rvroom+25);

if(L > Lthresh && M > Mthresh && R > Rthresh)

{

encoder(intersection);

count=count+1;

}

if(digitalRead(Lbumper) == 0 && digitalRead(Rbumper)==0)

{

break;

}

else if(L > Lthresh && M < Mthresh && R < Rthresh)

analogWrite(Lspeed,0);

else if(L < Lthresh && M < Mthresh && R > Rthresh)

analogWrite(Rspeed,0);

}

analogWrite(Lspeed,0);

analogWrite(Rspeed,0);

}

void left()

{

digitalWrite(Rdirection,HIGH);

analogWrite(Rspeed,Rvroom);

digitalWrite(Ldirection,LOW);

analogWrite(Lspeed,Lvroom/2);

encoder(turn);

while(true)

{

int L = analogRead(leftSensor);

int M = analogRead(middleSensor);

int R = analogRead(rightSensor);

if(L < Lthresh && M > Mthresh && R < Rthresh)

break;

}

analogWrite(Rspeed,0);

analogWrite(Lspeed,0);

}

void right()

{

digitalWrite(Ldirection,HIGH);

digitalWrite(Rdirection,LOW);

analogWrite(Rspeed,Rvroom/2);

analogWrite(Lspeed,Lvroom);

encoder(turn);

while(true)

{

int L = analogRead(leftSensor);

int M = analogRead(middleSensor);

int R = analogRead(rightSensor);

if(L < Lthresh && M > Mthresh && R < Rthresh)

break;

}

analogWrite(Lspeed,0);

analogWrite(Rspeed,0);

}

void spin()

{

digitalWrite(Ldirection,HIGH);

digitalWrite(Rdirection,LOW);

analogWrite(Lspeed,Lvroom-30);

analogWrite(Rspeed,Rvroom-22);

encoder(spinny);

while(true)

{

int L = analogRead(leftSensor);

int M = analogRead(middleSensor);

int R = analogRead(rightSensor);

if(L < Lthresh && M < Mthresh && R > Rthresh)

break;

}

analogWrite(Lspeed,0);

analogWrite(Rspeed,0);

}

void encoder(int turns)

{

int wheelCountL = 0;

int wheelCountR = 0;

prev\_L = digitalRead(Lencoder);

prev\_R = digitalRead(Rencoder);

while (wheelCountL < turns && wheelCountR < turns)

{

next\_L = digitalRead(Lencoder);

next\_R = digitalRead(Rencoder);

if (prev\_L != next\_L)

wheelCountL++;

if (prev\_R != next\_R)

wheelCountR++;

prev\_L=next\_L;

prev\_R=next\_R;

}

}

void pickup()

{

digitalWrite(Ldirection,LOW);

digitalWrite(Rdirection,LOW);

analogWrite(Lspeed,Lvroom-70);

analogWrite(Rspeed,Rvroom-62);

encoder(backup);

analogWrite(Lspeed,0);

analogWrite(Rspeed,0);

delay(800);

tilt.write(78);

pan.write(70);

delay(500);

grip.write(120);

delay(500);

tilt.write(220);

digitalWrite(Ldirection,LOW);

digitalWrite(Rdirection,LOW);

analogWrite(Lspeed,Lvroom);

analogWrite(Rspeed,Rvroom);

encoder(10);

analogWrite(Lspeed,0);

analogWrite(Rspeed,0);

}

void dropoff()

{

tilt.write(220);

delay(300);

tilt.write(60);

delay(50);

grip.write(40);

delay(500);

tilt.write(145);

digitalWrite(Ldirection,LOW);

digitalWrite(Rdirection,LOW);

analogWrite(Lspeed,Lvroom);

analogWrite(Rspeed,Rvroom);

encoder(6);

analogWrite(Lspeed,0);

analogWrite(Rspeed,0);

}

void spinnyL()

{

digitalWrite(Ldirection,HIGH);

digitalWrite(Rdirection,LOW);

analogWrite(Lspeed,Lvroom);

analogWrite(Rspeed,Rvroom);

encoder(5);

while(true)

{

int L = analogRead(leftSensor);

int M = analogRead(middleSensor);

int R = analogRead(rightSensor);

if(L < Lthresh && M < Mthresh && R > Rthresh)

break;

}

analogWrite(Lspeed,0);

analogWrite(Rspeed,0);

}

void spinnyR()

{

digitalWrite(Ldirection,HIGH);

digitalWrite(Rdirection,LOW);

analogWrite(Lspeed,Lvroom);

analogWrite(Rspeed,Rvroom);

encoder(30);

while(true)

{

int L = analogRead(leftSensor);

int M = analogRead(middleSensor);

int R = analogRead(rightSensor);

if(L < Lthresh && M < Mthresh && R > Rthresh)

break;

}

analogWrite(Lspeed,0);

analogWrite(Rspeed,0);

}

int getlocation()

{

while(1)

{

int IRinput = myIRserial.receive(200);

if (IRinput == 48)

{

val = 0;

return val;

}

if (IRinput == 49)

{

val = 1;

return val;

}

if (IRinput == 50)

{

val = 2;

return val;

}

}

}