ECE 375 Final Project

SUNY Oswego

Electrical and Computer Engineering Department

Team: Lake Show

Abedallah Abedrabbah

Justin Ross

Zack Salvador

Spring 2017

Dr. Manseur

May 9, 2017

Paper compiled and edited by Justin Ross

1 Abstract

Written by Zack Salvador

This project tasked students with designing and programming a robot to navigate a given course by implementing things learned during the course of ECE375. The robot had to navigate the path while staying within the lines and be able to detect an object obstructing the path, proceeding accordingly. In the case of this project, the course can have different paths: a short path, and a long path. One of these two paths will be blocked with an object, as mentioned above. After successfully navigating one of the two paths, the robot will then enter a parking area. There will be three different spots in this area, with only one of them ever being vacant. The robot must detect which spot is open, and be able to park within the confines of this spot. How successful the robot is in completing these tasks will be determined by a set scoring list, which shows how many points will be subtracted from the final score for each possible error during the test runs. The scoring will be discussed later on in this report.

2 Introduction

Written by Zack Salvador

This project is given near the end of the ECE375, Microprocessor Applications, course in order to test and display the knowledge that its students have obtained through the semester. The project tests the knowledge of various components that work with the Dragon Board, such as motors, encoders, line sensors, and various others. It also tests how familiar the students are with the numerous ports and components within the board, as well as testing the students' basic knowledge of coding in assembly language and C. This project also demonstrates how a successful team can be ran, by having the individual groups decide on not only what times to meet and work on the project, but also to divide the different needs of the project amongst the group so that it can be successfully completed within the allotted time frame.

The specific task of this project was to have groups of students design, program, and run a track designed by the professor. The robot will start in a consistent starting point, and will have to navigate one of two paths, using line sensors to keep it moving between the black lines of the laid out track. It will run either a short path or a long path, one of which will always be blocked with an obstacle. The robot will have to make use of a ping sensor utilized for object detection and avoidance, in order to continue down the unblocked path. Once the robot has started on the unblocked path, it will continue to navigate the rest of the course until it reaches the parking area. Once in the parking area, the robot must utilize the ping sensor again to detect which one of the three parking spots in the area is open. Once the open spot is detected the robot will use the line sensors to park completely within the confines of the parking space. The robot's performance will be graded on several different aspects

during its demonstration runs. Points will be deducted for occurrences such as the robot not being able to make it to the parking zone or having a wheel cross the line. Points can also be deducted for the robot leaving the track, colliding with an obstacle, exceeding the parking space, or any other thing that would coincide with the robot not performing the desired task.

Previous work from this course will be beneficial in terms of working with some of the components on the robot. Over the past few weeks the groups in the course worked on several teaching labs that helped to familiarize them with components that will be needed to complete the desired tasks. The students completed teaching labs that introduced them to servo motors and encoders, ping sensors, and line sensors. Each of these components are crucial when it comes to completing the course. The servos are what causes the robot to move forward, backward, left, or right. Hooked up to these motors are encoders, which use pulses to keep track of how far the wheels have turned. This is very crucial when it comes to distance and turn control. The line sensors work in unison with the servos and encoders in order to keep the robot within the lines of the track. The final major component that was gone over in a teaching lab was the ping sensor, which will tell the robot when there is an obstacle in the path, and how far away that obstacle is.

All of the knowledge obtained throughout the duration of this course, as well as the teaching labs and exercises will help in the completion of these desired tasks. With this knowledge, the group will be able to design and program the robot to successfully run the course, meaning it will stay within the lines of the track, decide which path to follow by detecting an object in either path. It will use these components to reach the parking area without incident, and detect which parking spot is available and park completely within the open spot.

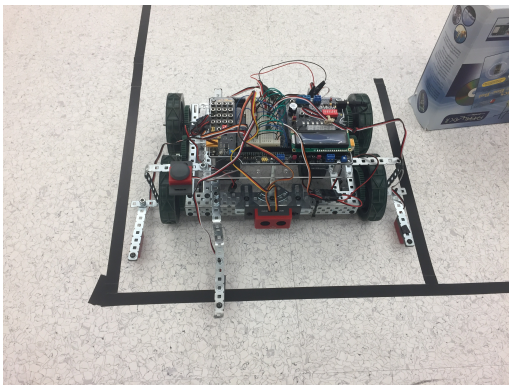

Figure 1: The final robot

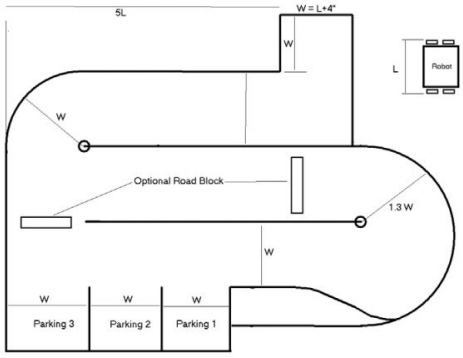

Figure 2: The track

3 Project Description

Written by Abedallah Abedrabbah

When approaching for a solution, a simple design was taken using a minimal amount of hardware possible which is described above above. Also an approach to setting up the ports

of the hardware to categorized in different ports. The servos put in the Port P, Encoder and ping sensor in Port T, and the line sensors that connected to the Analog to Digital Converter Ports. Also, having using only one encoder kept our code less complex and to avoid the arithmetic to find the difference and setting up two encoders.

The basic task difficulty that was present was figuring out the code and checking how the hardware would react to it. The solution was to slow implement each hardware of the design in the code. The first was the ping sensors which was to check if there was an object present in front of the robot. Next the servos were implemented along with the encoders which was used to tell robot to go forward, turn, and backup. The final piece of hardware was the three line sensors that were used to check if the robot was moving off track and have it realign. The solution in the code was that the encoder would measure the distance traveled to determine the moment the robot parked. During this the ping and line sensors are constantly updating and checking the values, if there is a line crossing, or if there is an object in front of it.

The code was designed so that the robot would run a small distance and then allow the line sensors to check if the robot was moving off track. For the ping sensors the robot, would send pulse and check if there was an object in front of it. The ping sensor was set that if the robot has traveled less that the given distance in the code. A variable in the code is set to one and would enable the long park method or if not the short park. There was two different park methods that was used which were long and short park long would activate when the first object was detected while the short path would activate when an object is found when the distance is higher than the given distance.

3.1 Difficulties

The difficulties we experienced were wiring issues. This caused a variety of issues including a breadboard melting, ping sensors being unable to detect objects, and the line sensor being altered that they can not read correctly. To solve the wiring issues, the board was reorganized such that hardware wires were sectioned close to their respective ports to prevent any wiring from coming loose, which could lead to the melting of the breadboard. Which happened to Mr. Salvador's Dragonboard. Hardware malfunctions were also experienced when running the project code. When coming to the ping sensor, the time of checking was adjusted so that the sensors and check more often and also fixing turns so the that robot would align straight enough to detect an object. For the line sensors the arms were adjusted and set up before running the robot and its code. The charge of the batter also lead to some problems. The battery variation difficulty made that every time the robot ran it must have a decently charged battery. If not, the encoder would not count correctly on the track where can lead other issues or make the robot completely workable.

4 Conclusions

Written by Zack Salvador

Upon completion of the demonstrations, it was clear that the solution this group found was a rather useful one. The robot was able to complete the desired task during the three trial runs, despite some difficulties during the process. The design of our robot along with the implementation of the various components and code allowed the robot to traverse the track while being able to utilize object detection and successfully park in an open spot.

The robot's ability to move around the track and go on the correct path was exceptional. The robot rarely had cases where it went off of the track or failed to detect an obstacle during the development and testing phases. It didn't leave the track or collide with an object once during the demonstration phase of the project.

The biggest issues with the project were apparent during the parking phase. The short path made it easy to initiate the parking phase, as it would begin when it detected the first object in the parking area and not in the track. But during the long path there was really no set way to initiate the parking methods. The best and most useful idea was to keep a count variable that would increment every time a certain method ran, and to use trial and error to see how far a particular value would get the robot on the track. This trial and error with the count variable would be used to see about when the robot would enter the parking area going about the long path. Since this count was not consistent whatsoever, it was extremely difficult to make the robot's parking ability on the long path repeatable. There would be a very easy solution to this problem that will be mentioned later.

The parking method itself was not difficult for this project, it was just knowing exactly when to initiate the method, particularly on the long path. As previously mentioned, the parking method on the short path could easily be initiated by knowing whether the first object detected was on the track or in a parking spot. This part of the parking presented no issue. The long path presented difficulties, though. There was no set way to determine when the parking method should begin. A very simple and logical solution to this issue would be to modify the track so that the parking area is easily detectable. The best way to do this would be to put a piece of tape, other than black, at the entrance to each ends of the parking area. This makes sense as most autonomous vehicles are either signaled to park remotely, or have some sort of signal that lets it know when it is supposed to park. This issue, however, was able to be overcome by using a count variable. Although this solution did not provide perfectly repeatable results, it was easy to tweak areas of the code after test runs to reach the final goal.

This project was very useful in testing how much of the course material the students were able to retain, as it implemented several of the topics covered during the course. Perhaps more important than the demonstration of the knowledge obtained was the experience it was able to provide students in terms of working in groups to achieve a common goal. This project modeled what engineers today often to on big projects in a work environment. Students had to schedule work times and plan what needed to be completed when, and also share ideas

on what is the best possible way to go about solving the particular problem at home. This project not only taught the groups how to implement the lessons learned throughout the duration of this course, but also provided valuable experience that can be carried over into a professional work environment.

5 Hardware Components

Written by Justin Ross

- 1x Dragon12-Light Project Development Board
- 1x VEX NiMH 7.2v 3000mah Battery Pack
- 1x Custom Robot Chassis built using VEX components
- 1x Tamiya to Barrel Plug Power Cable
- 1x VEX Bumper Switch
- 2x VEX Servo Motors
- 2x VEX H-Bridges
- 2x VEX Optical Shaft Encoders
- 1x VEX Ping Sensor
- 3x VEX Line Trackers

6 References

Written by Justin Ross

[1] *Programming the DRAGON12-Plus-USB in C and Assembly Language Using CodeWarrior.*

[2] VEX Robotics. 2 wire motor 393. Available at `https://content.vexrobotics.com/docs/instructions/276-2177-inst-0712.pdf`.

[3] VEX Robotics. Line tracker. Available at `https://content.vexrobotics.com/docs/instructions/276-2154-Line-Tracker-Instr-0312.pdf`.

[4] VEX Robotics. Optical shaft encoder (2-pack). Available at `https://content.vexrobotics.com/docs/instructions/276-2156-instr-0312.pdf`.

7 Code

Code Commented by Justin Ross

```
//Abedallah Abedrabbah, Justin Ross, and Zack Salvador
// ECE375 Final Project
// This program allows the robot to move on a pre-determined track, there are
//2 different places for obstacles, if it senses any, it will turn out of the
//way. At the end of the track, it needs to find a free parking spot
//I certify that this program works according to specification
#include <hidef.h>      /* common defines and macros */
#include <mc9s12dg256.h>     /* derivative information */
#pragma LINK_INFO DERIVATIVE "mc9s12dg256b"
#include "main_asm.h" /* interface to the assembly module */
//---------------------------------------------------------------------//
//Declare Methods
void Update (void); //Updates the line sensor values
void stop(void); //Stops the motors
void PulseOut(void); //Sends out a pulse from the Ping Sensor
void getDist(void); //Gets the distance the ping traveled
void Sensor(void); //Concatenates the previous two methods
void PAC_init(void); //Initializes the Pulse Accumulator System
void move_forward(int n); //Has the motors move forward
void move_back(int n); //Has the motors move backward
void turn_right(int n); //Has the motors spin right
void turn_left(int n); //Has the motors spin left
void shortPark(void); //Begins the logic for the shirt path
void longPark(void); //Begins the logic for the long path
void finished (void); //Stops the program
void basiclineSensor(void); //Checks line sensor values when parking
void lineSensor(void); //Checks line sensor values any other time
//---------------------------------------------------------------------//
//Initialize Variables
int mid,left,right; //The variables for the line sensors
int totalcount =0; //The count used to keep track of the distance
//that the robot has traveled
int sonarCount =0; //Keeps track of how many objects have been
//detected
int firstcount =0; //Keeps track of how many times the middle line
//sensor detected black tape
int pcount; //A Pulse accumulator tracker
int parkCount; //A variable used to break out of the program
int first = 0; //Keeps track of the move forward calls in the
//Long park method
//---------------------------------------------------------------------//
void interrupt 9 handler1(){
  HILOtimes1();       //Interrupt Handler for the Ping Sensor
}
//---------------------------------------------------------------------//
```

```
//Updates the Line Sensors
void Update (void){
  mid = ad0conv(2);     //Reads the value on the right line sensor
  mid = mid >>1; //shifts the value right 1 bit, makes it easier
   //to compare values
  left = ad0conv(1);    //Reads the value on the left line sensor
  left = left >>1;
  right = ad0conv(6);   //Reads the value on the right line sensor
  right = right >>1;
}
//-----------------------------------------------------------------------//
//Stops the Motors
void stop(void){
   set_servo76(4500);   //Sets the servos to stop
   set_servo54(4500);
}
//-----------------------------------------------------------------------//
//Sends a Pulse Out
void PulseOut(void){
int time, flag;         //Initialize Local Variables
  DDRT |= 0x08;         // set Port 3 as Output
  PTT |= 0x08;
  TCTL2 = 0x40;         //set pin to toggle
  TIOS |= 0x08;         //Port 3 is now Output Compare
  TSCR1_TEN = 1;        //Enable Timer Counter
  time = TCNT;

  TFLG1 |= 0x08;        //Clears the Flag
  time += 23880;        //Pin is high for 20 micro seconds
  TC3 = time;     //Updates the count
  flag = TFLG1 & 0x08;
while(flag ==0){        //wait until the TFLG1 is thrown
  flag = TFLG1 & 0x08;
}

 TFLG1 |= 0x08;           //Clear the Flag
 time = time + 120;       //Wait 5 microseconds
 TC3 = time;
 flag = TFLG1 & 0x08;
while(flag ==0){         //Wait until the TFLG1 is thrown
  flag = TFLG1 & 0x08;
 }
}
//-----------------------------------------------------------------------//
//Gets the Distance of an object
void getDist(void){
 int hitime;   //Initialize local variables
 HILO1_init();                    //initialize the high low time
do{
 hitime = get_HI_time1();        //Check how long the ping was gone
```

```
if (hitime > 300 && hitime < 2000){ //Checks if an object is between 3 and 20 inches away.
 sonarCount++;   //Keep track of how many objects have been detected.
if(sonarCount >=1  && totalcount < 1000){ //Checks if the robot is before where the
//1st box would be placed
        turn_left(75); //Turns left about 90 degrees
}else if(sonarCount >= 1 &&totalcount>1000){//Checks if the robot is after where the
//1st box would be placed
        while(1){
        shortPark();               //Starts to park if the short path is open.
        }
}else{
        stop();     //if something unexpected happens, stop.
      }
    }
  }while(hitime > 300 && hitime < 2000);
}


//-------------------------------------------------------------------------//
//Concatenates the two sensor methods into one, easy to call method
void Sensor(void){
  PulseOut();   //Sends out a Pulse
  getDist();   //Checks if there's an object close by
}


//-------------------------------------------------------------------------//
//Initialized the Pulse Accumulator System
void PAC_init(void) {
  DDRT = DDRT & 0x7E;       // Pin 7 & 0 of port T is input for pulses
  PBCTL = 0x40;             // PAEN=1, PAMOD=0, PEDGE =0, No Interrupts
  PACTL = 0x40;             // PAEN=1, PAMOD=0, PEDGE =0, No Interrupts
}
//-------------------------------------------------------------------------//
//Initialized the Servos
void servo_init(void){
  servo54_init();          //Pin 5 of Port P is enabled to drive the servo
  servo76_init();           //Pin 7 of Port P is enabled to drive the servo
}
//-------------------------------------------------------------------------//
//Move Forward
void move_forward(int n){
  PACN32 = 0;             //makes sure the pulse counter is set to zero
  pcount=PACN32;
  while(pcount<n){        // for how ever many pulses the code calls for,
   set_servo76(4850);     //sets the left servo
   set_servo54(4850);     //sets the right servo
      pcount=PACN32;
    }
    stop();   //stops the motors
```

```
  }
//-------------------------------------------------------------------------//
//Move Backwards
void move_back(int n){
  PACN32 = 0;            //makes sure the pulse counter is set to zero
  pcount=PACN32;
while(pcount<n){         // move back for how ever many pulses the code calls for,
  set_servo76(3700);     //sets the left servo
  set_servo54(3700);     //sets the right servo
  pcount=PACN32;       //updates the pulse counter
  }
  stop();  //stops the motors.
}

//-------------------------------------------------------------------------//
//Turn Right
void turn_right(int n){
  PACN32 = 0;            //makes sure the pulse counter is set to zero
  pcount=PACN32;
while(pcount<n){         // for how ever many pulses the code calls for,
             // the servos will move at opposite rates causing a right turn
  set_servo76(5000);     //sets the left servo
  set_servo54(4000);     //sets the right servo
  pcount=PACN32;  //Update the count for the turn
  Update();      //Updates the Line Sensor values
   }
  stop();        //stops the motors
}

//-------------------------------------------------------------------------//
//Turn Left
void turn_left(int n){
  PACN32 = 0;         //makes sure the pulse counter is set to zero
  pcount=PACN32;
while(pcount<n){      // for how ever many pulses the code calls for,
      // the servos will move at opposite rates causing a left turn
  set_servo76(4000);  //sets the left servo
  set_servo54(5000);  //sets the right servo
    pcount=PACN32;   //Update the count for the turn
  Update();     //Updates the Line Sensor Values
   }
  stop();             //Stops the servos
}

//-------------------------------------------------------------------------//
//The Parking Logic if the Short Path is unblocked
void shortPark(void){
  int hitime;   //Declare the local variables
  move_back(8);   //Move back slightly
  turn_left(95);   //Turn left about 90 degrees
```

```
do{
  hitime = 0;   //Initialize the variables used in the method
  parkCount = 0;
  move_forward(130);  //Move forward to the next parking spot
  turn_right(65);   //Turn right to face the box
  PulseOut();   //Send out a pulse
  HILO1_init();       //initialize the high low time
  hitime = get_HI_time1(); //Check how long the ping was gone
if (hitime > 300 && hitime < 3200) { //Check if an object is between 3 and 32 inches away
  turn_left(65);      //If there's an object in the way, turn left and check another spot
}else{
 while(mid<400){
  basiclineSensor();  //If no object in the way, move forward until the
     //middle line sensor is above tape
 }
  parkCount = 1;   //change this variable to break out of the loop
  }
 }while(parkCount < 1);
  finished();   //end the program
}
//-------------------------------------------------------------------------//
//The Parking Logic if the Short Path is blocked
void longPark(void){
  first = 0;  //Initialize the Variables
do{
int hitime;   //declare and declare more variables
  parkCount = 0;

if(first ==0){ //When it's checking the first parking spot
  move_forward(60); //move forward a little
  first++; //make sure that the first movement isn't repeated
 }else{
  move_forward(130);//move to the next parking spot
  }
  turn_left(75); //face towards the parking spot
  PulseOut(); //send out a pulse
  HILO1_init(); //initialize the high low time
  hitime = get_HI_time1();//Check how long the ping was gone
if(hitime > 300 && hitime < 3200) { //Check if an object is between 3 and 32 inches away
  move_forward(5); //move forward slightly to stay within the lines
  turn_right(65); //turn out of the way to check for another parking spot
}else {
  while(mid<200){
  basiclineSensor();//If no object in the way, move forward until the middle
   //line sensor is above tape
  }
  parkCount = 1; //change this variable to break out of the loop
 }
}while(parkCount < 1);
  finished(); //end the program
```

```
}
}
//-------------------------------------------------------------------------//
//A Basic Line Sensor used for parking straight
void basiclineSensor(void){
  Update(); //Updates the values detected by each line sensor
if(mid < 490 && left < 200 && right <490){ //If all the line sensors are above
   //white tile go forward
  set_servo76(4825);        //sets the left servo
  set_servo54(4825);        //sets the right servo
  Update();      //Updates the values detected by each line sensor
}else if(mid < 490 && left > 200 && right <490){//If the left line sensor is above
//black, turn right
  turn_right(1);  //turns the rover right
}else if(mid < 490 && left < 200 && right >490){//If the right line sensor is above
//black, turn left
  turn_left(1); //turns the robot left
}else {
  stop(); //If something unexpected happens, it stops
   }
}
//-------------------------------------------------------------------------//
//Drives using the line sensor
void lineSensor(void){
int temp =0;     //Initialize the variables
  Update(); //Updates the values detected by each line sensor
if(mid < 490 && left < 200 && right <490){ //If all the line sensors are above
   // white tile go forward
  if(totalcount <2400){     //Checks if the robot is near the long parking condition
   Sensor();     //Checks if something is in the way
   totalcount++; //increments the count to keep track of where the robot is.
   set_servo76(4850);       //sets the left servo
   set_servo54(4850);       //sets the right servo
   }else{
   longPark();       //when it reaches the spot to begin parking, it parks.
  }
 }else if(mid > 200 && left < 200 && right <490){//If the middle line sensors are above black
   stop(); //stop momentarily,
   ms_delay(10);
    if(firstcount ==0){  //If the line is black the first time
   move_back(10);                    //back up slightly
   turn_right(60); //and turn right roughly 90 degrees
   firstcount++;         //keeps track of the number of times only the middle is black
    }else if (sonarCount == 0){
   finished();                              //first parking condition
}else if(sonarCount > 1 && firstcount ==1){ //if the first parking spot in the short path is c
   finished();                                //end the program
}
}else if(mid < 490 && left > 200 && right <490){//If the left line sensor is
//above black,turn right
```

```
   turn_right(1); //turns right
}else if(mid < 490 && left < 200 && right >490){//If the right line sensor is above
//black, turn left
   turn_left(1); //turns left
}else{
   stop(); //If something unexpected happens, it stops
    }
}
//-------------------------------------------------------------------------//
//Ends the program
void finished(void){
 while(1){
  stop(); //parks forever.
 }
}
//-------------------------------------------------------------------------//
//Runs the course
void main(void) {
ad0_enable(); //Initializes the Analog to Digital Converters
PAC_init(); //Initializes the Pulse Accumulator System
servo_init();           //Initializes the servos
SW_enable(); //Initializes the Switch that starts the rover
while(1){
if(SW5_down()){ //Waits until the switch is pressed to start
while(1){
 lineSensor(); //Runs the course
        }
}
  }
}
```