Final Report

# 

Team 203 PB&J

Pierce Simpson

Brandon Libjanic

Abigail Francis

Jonathan Hawkins

December 9, 2015

Table of Contents

[Overview 3](#_Toc437030824)

[Design Summary 3](#_Toc437030825)

[Hardware Design 3](#_Toc437030826)

[Hardware 3](#_Toc437030827)

[Additional Part description 8](#_Toc437030828)

[Pin Assignments and Pin Usage Descriptions 9](#_Toc437030829)

[Circuit Diagrams 11](#_Toc437030830)

[Software Design 16](#_Toc437030831)

[Overview 16](#_Toc437030832)

[Block Diagram 17](#_Toc437030833)

[Function Descriptions 18](#_Toc437030834)

[Testing Design 20](#_Toc437030835)

[Overview 20](#_Toc437030836)

[Tests 21](#_Toc437030837)

# Overview

### Design Summary

This robot follows a black line track, analyzes intersection nodes, and avoids obstacles that are in its path. The design consists of two independent motors that adjust speed and direction in order to follow the track. Eight infrared transmitters and receivers are mounted in a circular formation in order to detect the dark track. The circular formation allows for node analysis in order to determine the location of the robot on the track. When the robot detects an object on the front it will move off the track and then find the track again once the obstacle has been passed. Objects are detected using an ultrasonic sensor which determines the distance of an object in front of the robot. As that distance decreases the robot begins the sequence that will allow it to avoid the object.

# Hardware Design

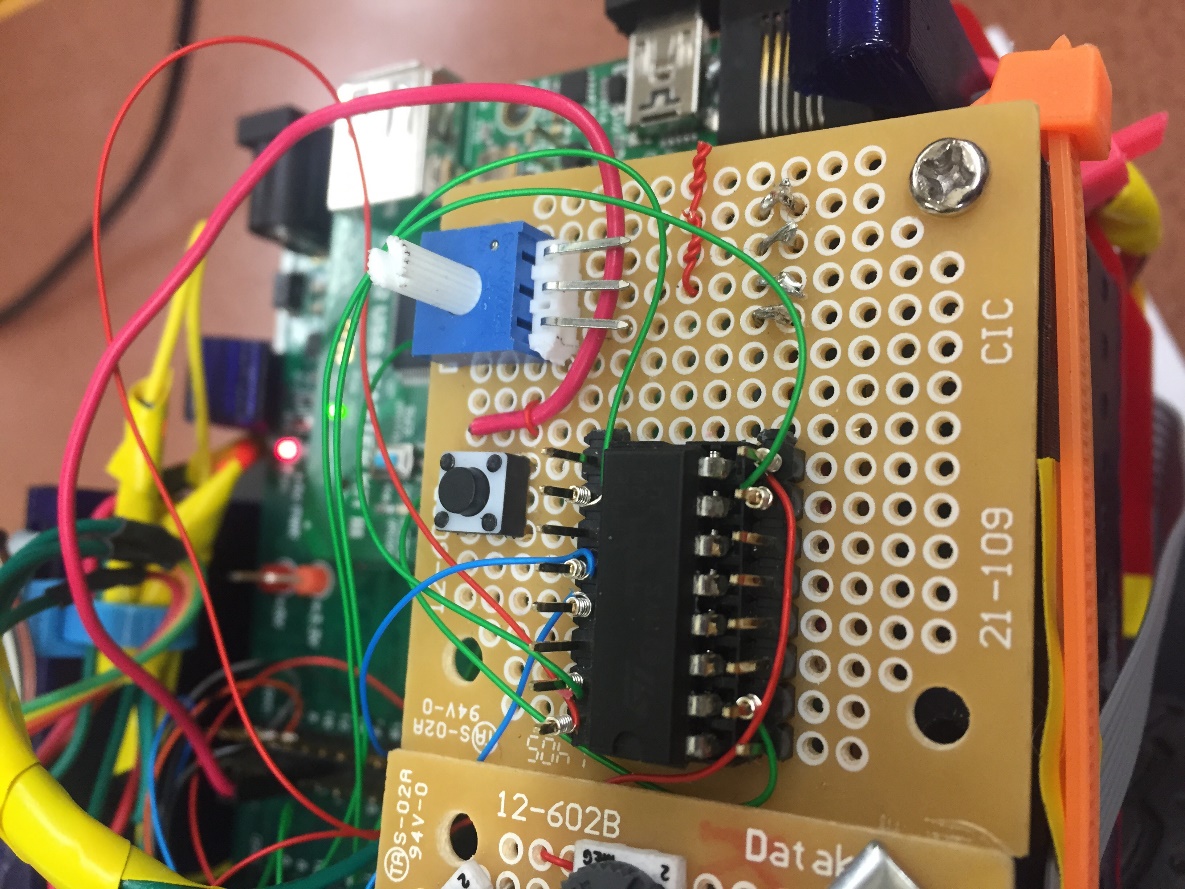
### Hardware

**Figure 1**



Ultrasonic sensor mounted to front of the robot

Figure 2



H-Bridge for Motor Control

Figures 3 and 4

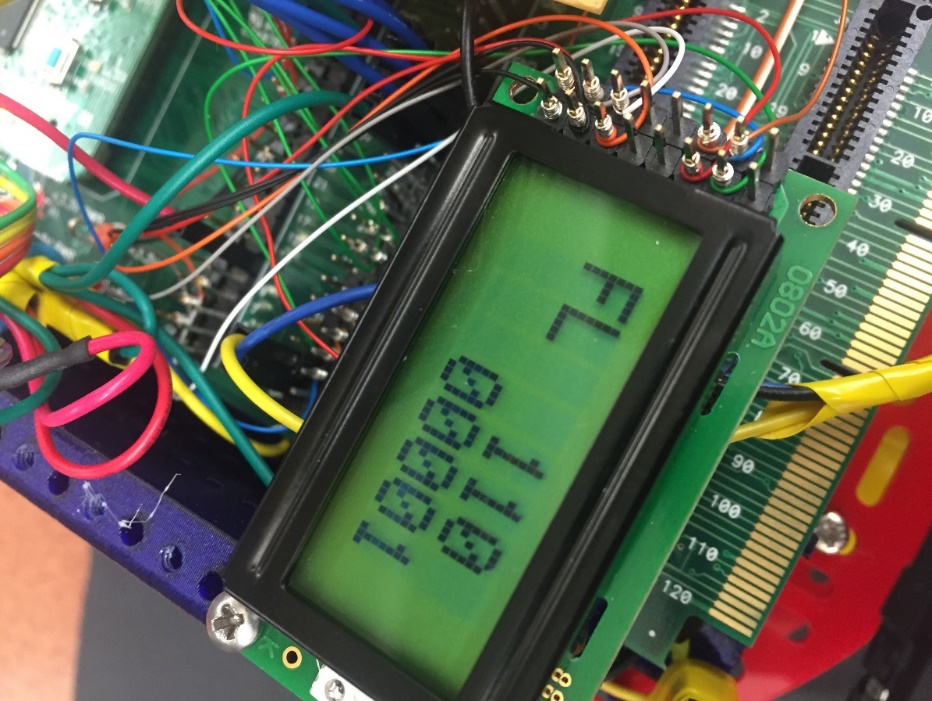
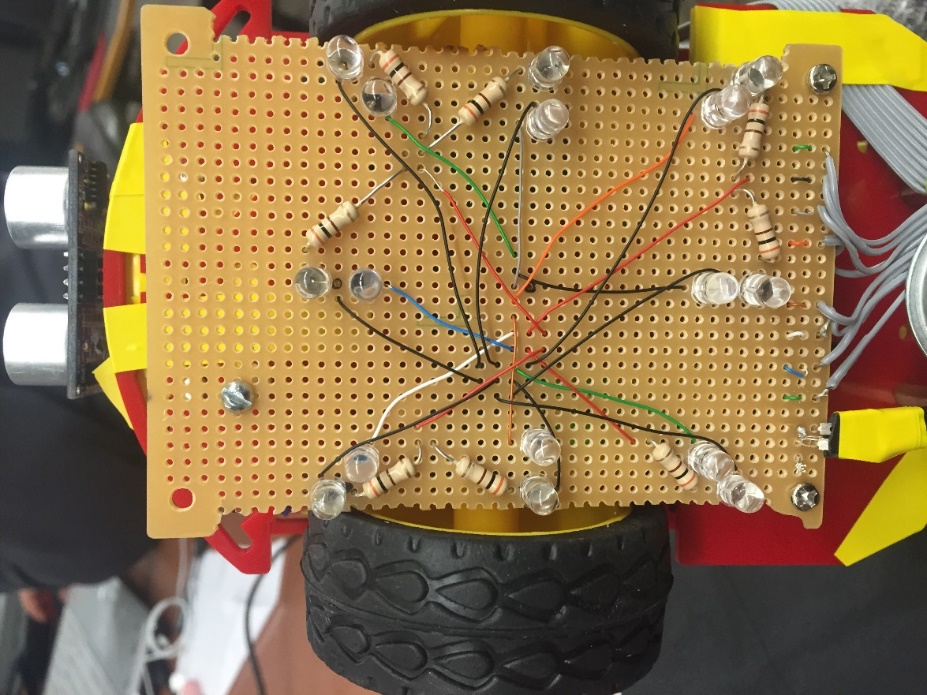


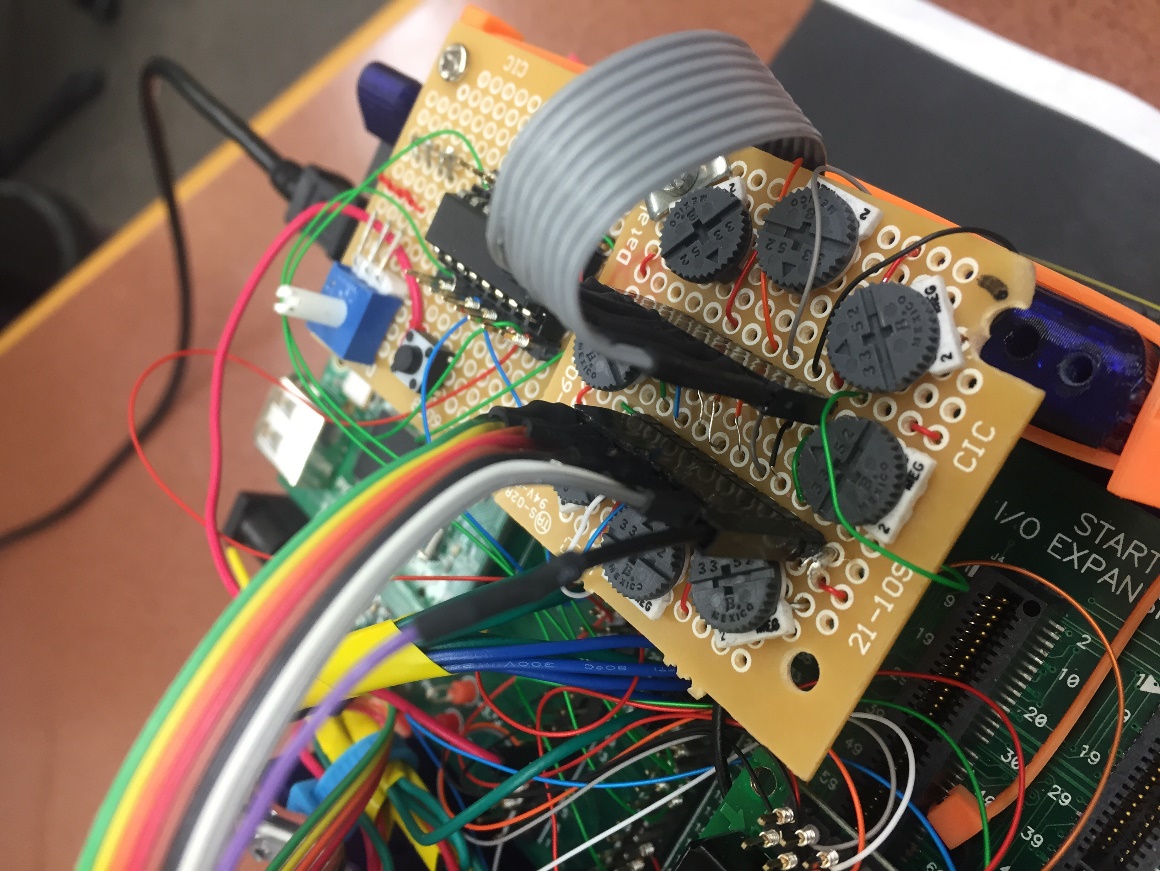
Figure 3 (mirrored): IR LED and sensor array board

Figure 4: LCD screen. The letters indicate what state the robot is in (i.e. “FL” = “Find line”) and the numbers show 1 for IR reflection and 0 for IR absorption.

Figure 3 is mirrored for explanatory purposes:

The top left IR LED corresponds to the top left numeric value on the LCD. The LCD numbers correspond to the LEDs in a clockwise from top left fashion.

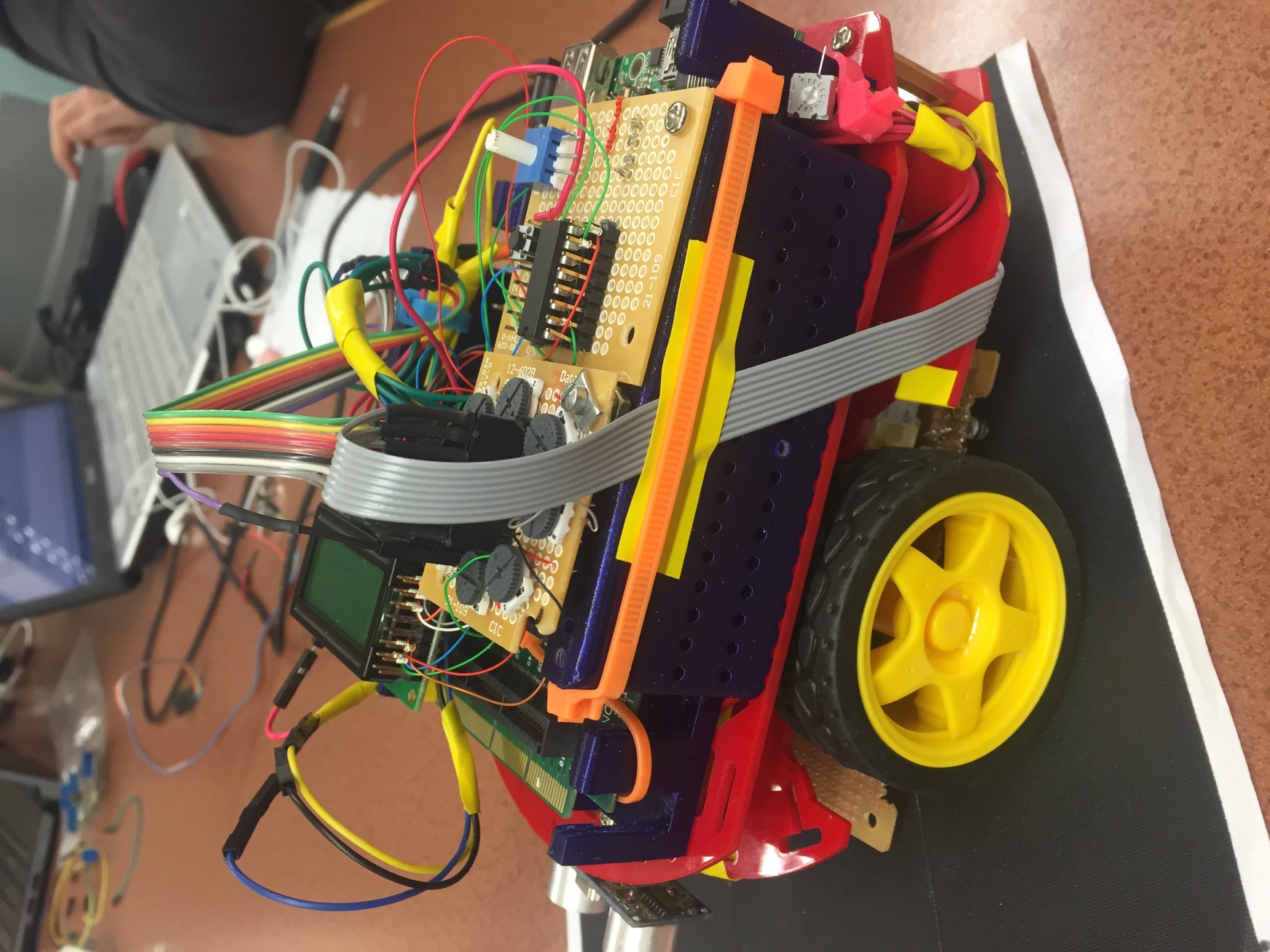
**Figure 5**



Potentiometer board with custom connector to IR board (left) and custom connector to PIC expansion board (right).

Potentiometer positions correspond to IR positions in Figure 3 for ease of adjustment.

**Figure 6**



The Coolest Robot Ever!!! Note potentiometer board mounted on top and ultrasonic sensor at left mounted to the front of the robot. IR board is the vector board mounted on the bottom.

### Additional Part description

Table 1

Parts added to lab

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Part Name** | **Part Number** | **Cost** | **Source** | **Manufacturer** | **Purpose** |
| 2M Potentiometer | A0014 | $1.00 | Elliot’s Electronics | China | To adjust sensitivity of the infrared arrays |
| Ultrasound Distance Sensor | HC-SR04 | $10.00 (2 pack) | Amazon | Sunfounder | Object distance detection |
| 300 Ω resistor | n/a | $0.00003 | Elliot’s Electronics | No one really knows | To regulate current flow through the IR LEDs |
| Ribbon cables | n/a | $0.28 /foot (est) | Provided by Mr. Hawkinses | China again? | For simplicity in connnections |

### Pin Assignments and Pin Usage Descriptions

**Figure 6**

PIC32MX470F512L Pinout

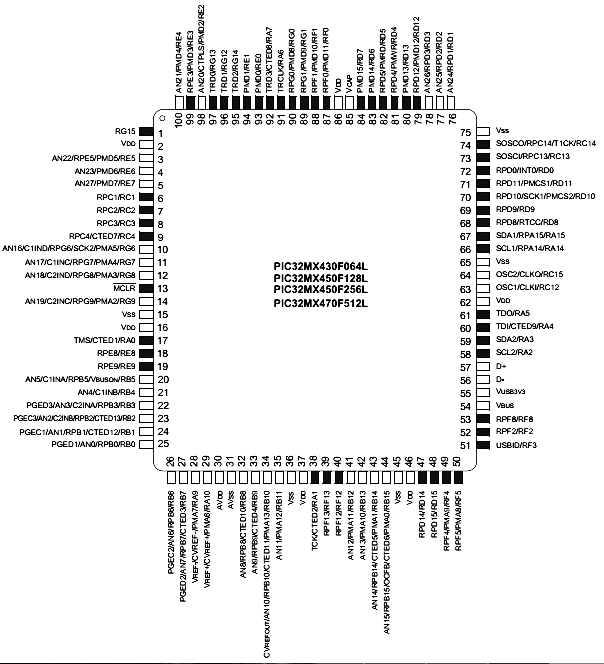


Table 2

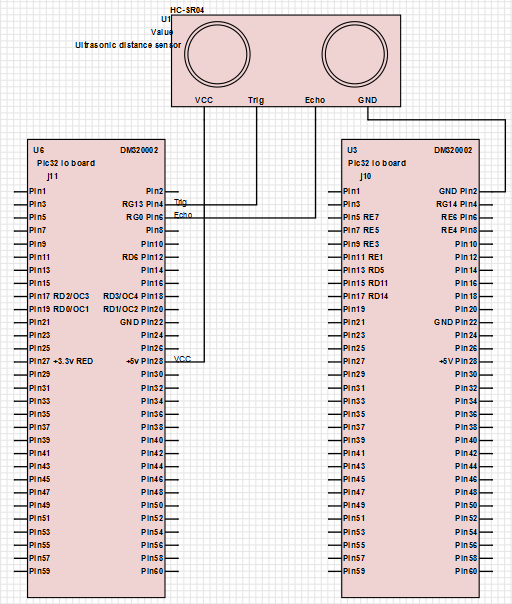


### Circuit Diagrams

Ultrasonic Distance Sensor Circuit Diagram

Figure 7

Ultrasound distance sensor circuit diagram



IR Circuit Diagrams

Figure 8a

IR circuit diagram (connections to expansion board)

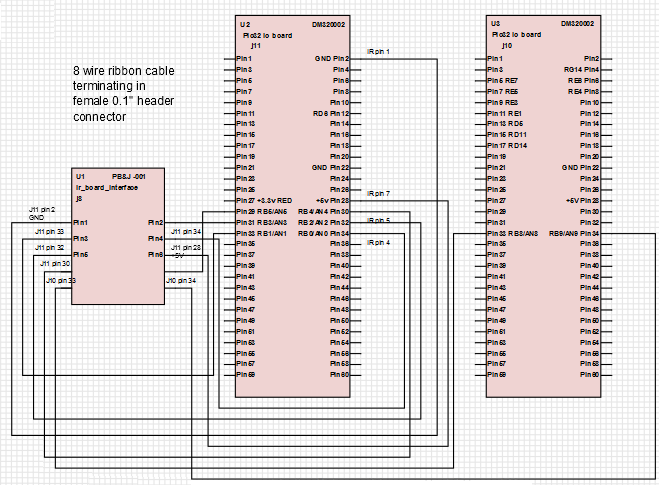
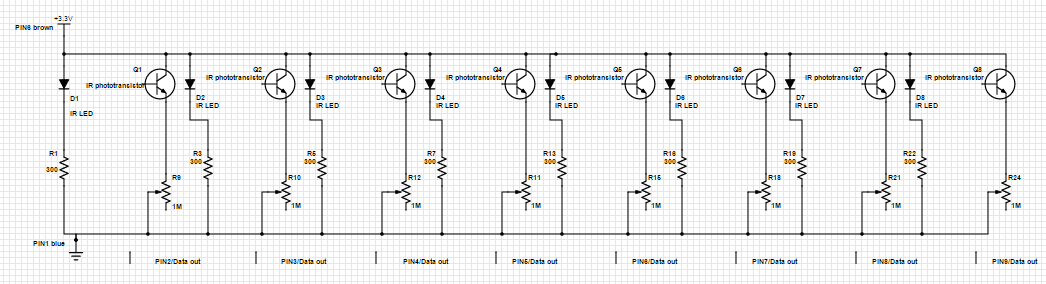
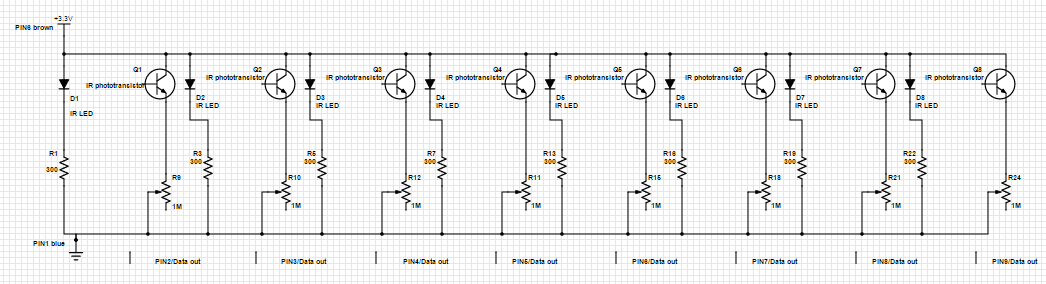


Figure 8b

IR circuit diagram (internal connections)

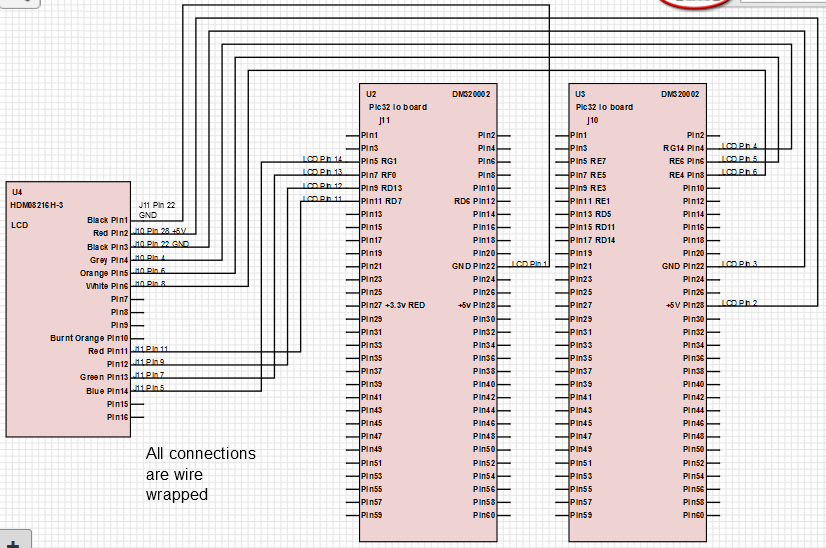




LCD Circuit Diagram

Figure 9

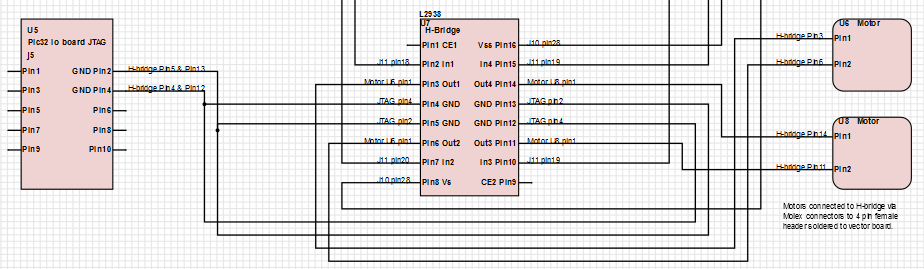
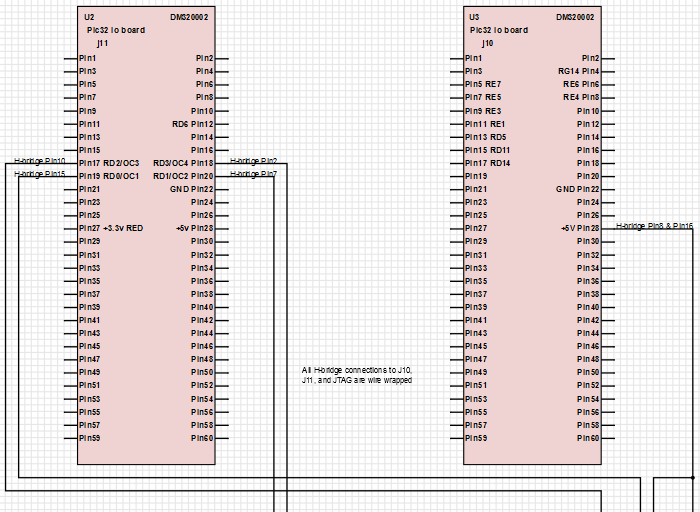
LCD circuit diagram



Motor/H-Bridge Circuit Diagram

Figure 10

H-bridge and motors circuit diagram (supposed to be one diagram, but SchemeIt sucks)



# Software Design

### Overview

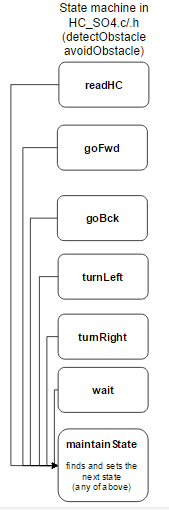
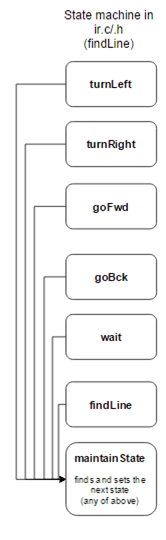
Given the additional complexity of this project when compared to previous labs, the focus of the software design for the final is on scalability. The software is built in a way that enables both easy isolation of the robot subsystems and allows for easy modification of robot behavior at a high level. ‘main.c’ implements a state machine that carries out high level tasks while lower level tasks controlled by sub-state machines. This structure should make for simpler debugging.

This being said, the high level state machine will have states that that run tasks similar to the following: track line, check for robot, avoid obstacle etc. These tasks will then be defined as their own state machine in their corresponding .h and .c file.

Given the scalable nature of our software architecture, we were able to create new and updated code from our previous lab. This was necessary because we upgraded several hardware components, most importantly, the IR sensors and the distance sensor. To keep our design backwards compatible, we modified only the required functions, and did so by defining new ones with the key word “new” in the new function. This makes the integration

The rightmost state machine below describes the high level state machine that we will be implementing in main.c. As described previously in this document, the states findLine, handleCollision, avoidObstacle, and detectObstacle in the main.c state machine will be located in separate files. The descriptions of these state machines are depicted by the diagrams below. Additionally, the diagram shows files that the state machines are in will be listed in. Each state will call a function that carries out an action. For example: the turnLeft state will call a function turnLeft() that makes the robot turnLeft. The readHc state will call a function readHc() that reads from the sensor. The same goes for all of the other states.

### Block Diagram



### Function Descriptions

Functions in pwm.c/.h

void initPWM();

Initialize the PWM for the motors and configure the pins for the robot.

void setMotorsIdle();

Sets motors to Idle.

void setMotorsBackward(int s);

Sets the motors to move in reverse with a speed of s.

void setMotorsForward(int s);

Sets the motors to move forward with a speed of s.

void setMotorsLeft(int s);

Sets the left motor to move forward with a speed of s, and right motor to stop.

void setMotorsRight(int s);

Sets the right motor to move forward with a speed of s, and left motor to stop.

void motorPiviotLeft(int s);

Sets the left motor to move forward with a speed of s, and the right motor to move backward with a speed of s.

void motorPiviotLeft(int s);

Sets the right motor to move forward with a speed of s, and the left motor to move backward with a speed of s.

void motorFindLine(int s);

Makes the robot move in a pattern that attempts to find a line.

Functions in ir.c/.h

void initIR();

Initialize the IR sensors and configure the pins for the robot.

void printIR();

Read from the IR sensors and write to the LCD.

void analogPrintIR();

Analog read from the IR sensors and write to the LCD.

int readIR();

Read from the IR and return an integer that can be decoded into the values read from each IR sensor.

int readNewIR();

Read from the IR and return an integer that can be decoded into the values read from each IR sensor. This function is new and has been upgraded to work with our 8 ir emitter/ receiver combination.

irStateType parseNewIRData(int Data)

Parse the data returned from the readNewIR() function, and return a state that the trackLine() state machine can use. This function is new and has been upgraded to work with our 8 ir emitter/ receiver combination.

int analogReadIR();

Analog read from the IR and return an integer that can be decoded into the values read from each IR sensor.

int trackLine();

A state machine, as described above, that enables the robot to track a line.

irStateType parseIRData(int data);

Parse the data returned from the readIR() function, and return a state that the trackLine() state machine can use.

Functions in HC\_S04.c/.h

int readHC();

Read data from the hc-S04 sensor, return an int with the read information.

void initHC();

Initialize the hc-S04 sensor and the pins it connects to.

handleHC();

A state machine that controls the operations related to reading from the hc-S04 sensor.

hcStateType parseHCData(int data);

Parse the data returned from the readHC() function, and return a state that the handleHC () state machine can use.

# Testing Design

### Overview

The testing for this design will be done in a modular and step-wise fashion. In doing so we can assure all the parts that will be connected to the robot work individually. Once the robot starts to be assembled more tests can be performed to ensure parts work when implemented together.

### Tests

TEST 1: IR Sensors:

The purpose of this test is to ensure the functionality of the IR sensor and the pins that it will be connected to.

The first part of this test involves determining if the pins that the IR sensor will be connected to are functioning properly. In order to do this each pin was first mapped as an input. Then, the functions: printIR(), testIR() were used to read the values on the four IR pins and display those values onto the LCD. When nothing is connected to the pins the expected output on pins 31,32,33,34 respectively is “0000” since all the pins are reading logic low. In order to test each pin, the pins were individually connected to logic high using a jumper. When the pin is connected to high a 1 should be displayed onto the LED. For example, when pin 32 is connected to logic high the LCD should display “0100”.

The second part of the test is to ensure the IR sensor itself works. Since the pins and the LCD display function were tested for correctness in the previous part of the test, the functionality of the IR sensor will be able to be tested fully now. In order to do this the IR sensor was connected to pins 31,32,33,34 and first pointed at an all-white surface. The expected reading display on the LCD for this setup is “1111” since the IR sensor should be causing the pins to be logic high. The next step is to face the IR sensor at an all-black surface. The expected reading display on the LCD for this setup is “0000” since the IR Sensor should not be causing the pins to be logic high.

TEST 2: IR Potentiometer Tuning

The purpose of this test is to tune the potentiometers so that the values read by the IR sensor are as close to logic low (1.0V) and logic high (3.75V) as possible. In doing so the IR sensor will not have to use the ADC converter.

The first step in performing this test is to connect power to the input of the IR board and connect the oscilloscope to the output of the IR board. The oscilloscope is setup to read the voltage coming off the of IR board output, this voltage is what needs to be as close as possible to logic low and logic high given a white or black surface respectively. The process of tuning the potentiometer involved flashing the black and white surfaces in front of the IR sensor and adjusting the potentiometer so that the voltage readings were close to logic high and logic low. The value of the potentiometer was read and recorded using the Digital Multi Meter. This process was chosen as opposed to working out the resistance value by hand because it provides more accurate results that can be seen on the oscilloscope.

Attached images show the voltage readings for black and white IR sensor readings.

Test 3: Testing Motor Functions

The purpose of this test is to ensure that all the different motor functions are properly controlling the motors. The reason this test is needed is because several different motor functions have been written that all serve a different purpose in terms of controlling the movement of the robot, and since any can be used for the competition every function needs to be tested.

In order to test the motor functions (setMotorsRotate, setMotorsSweepForward, setMotorsSweepBackwards, setMotorsForward, setMotorsBackwards) the functions were called with different hardcoded values for the input (ADCBufferValue). The robot was then programmed using these hardcoded values and the results of the motors were observed. For example the setMotorsSweepForward function was tested by calling it with an input of 400. The expected result is for the right motor to move forward at full speed while the left motor moves forward at about 80% speed. The code for testing this function can be found in the testMotorFunctionality() function.

Test 4: Testing the Range Detector

The purpose of this test is to ensure that the newly acquired sensor (the range detector) works properly. The reason this test is necessary is because the range detector will be used to control the states of the motors and it must be tested independently of the motors to ensure it is working correctly.

In order to perform this test the range detector will be connected to the microcontroller and the values will be read in via digital pins. An object will be moved in front of the sensor and then the distance the object is from the sensor will be varied. The values being read from the sensor will then be displayed onto the LCD using a previously tested function for printing onto the LCD. The values will be tested against an actual measurement of the distance from the object to the sensor using a measuring tape so the accuracy and precision of the sensor can be obtained.

There will be up to a ~40 ms delay for the sensor to return distance readings. We will hook up an oscilloscope to the TRIG pin and the ECHO pin and measure the time intervals for TRIG output to see if it matches 10µs as specified in the datasheet; for ECHO to see if it matches the 38ms as specified in the datasheet.



