Final Report

# 

Team 203 PB&J

Pierce Simpson

Brandon Libjanic

Abigail Francis

Jonathan Hawkins

December 9, 2015

Table of Contents

[Overview 2](#_Toc437266407)

[Design Summary 3](#_Toc437266408)

[Hardware Design 3](#_Toc437266409)

[Hardware 3](#_Toc437266410)

[Additional Part description 9](#_Toc437266411)

[Pin Assignments and Pin Usage Descriptions 10](#_Toc437266412)

[Circuit Diagrams 12](#_Toc437266413)

[Software Design 17](#_Toc437266414)

[Overview 17](#_Toc437266415)

[Block Diagram 18](#_Toc437266416)

[Function Descriptions 19](#_Toc437266417)

# 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

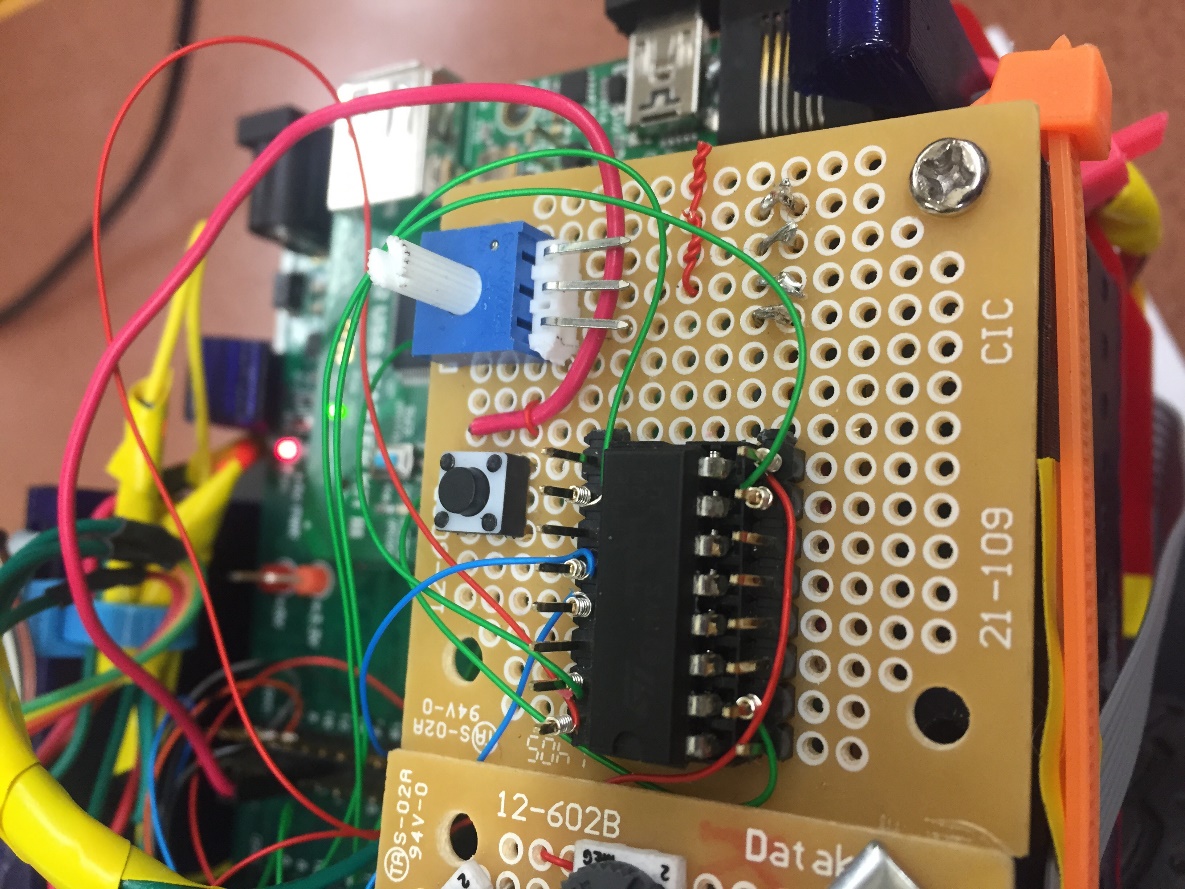
### Hardware

**Figure 1**



Ultrasonic sensor mounted to front of the robot

Figure 2



H-Bridge for Motor Control

Figures 3 and 4

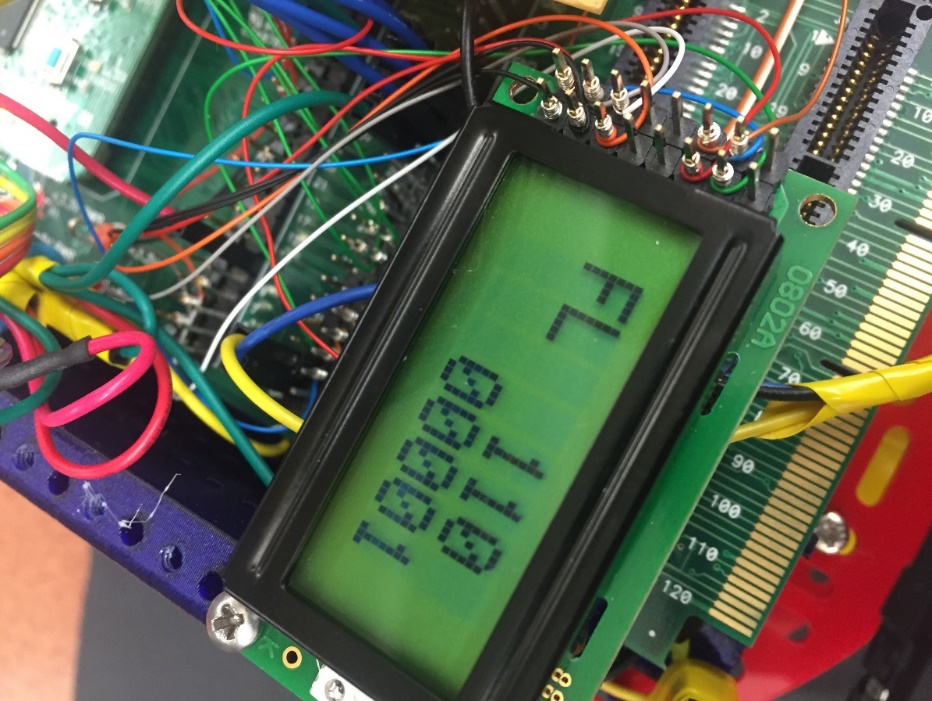
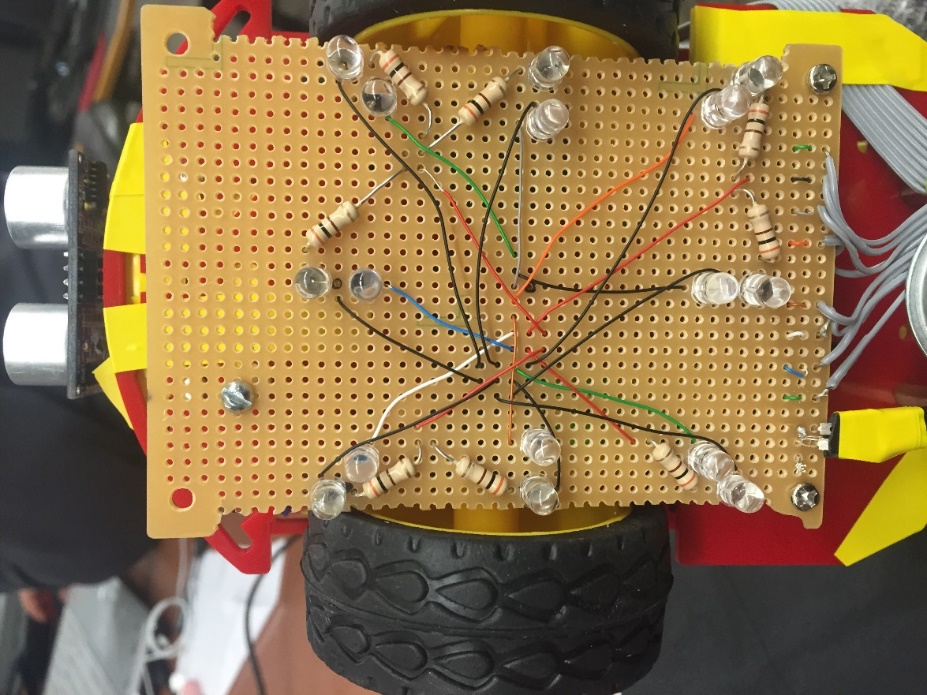


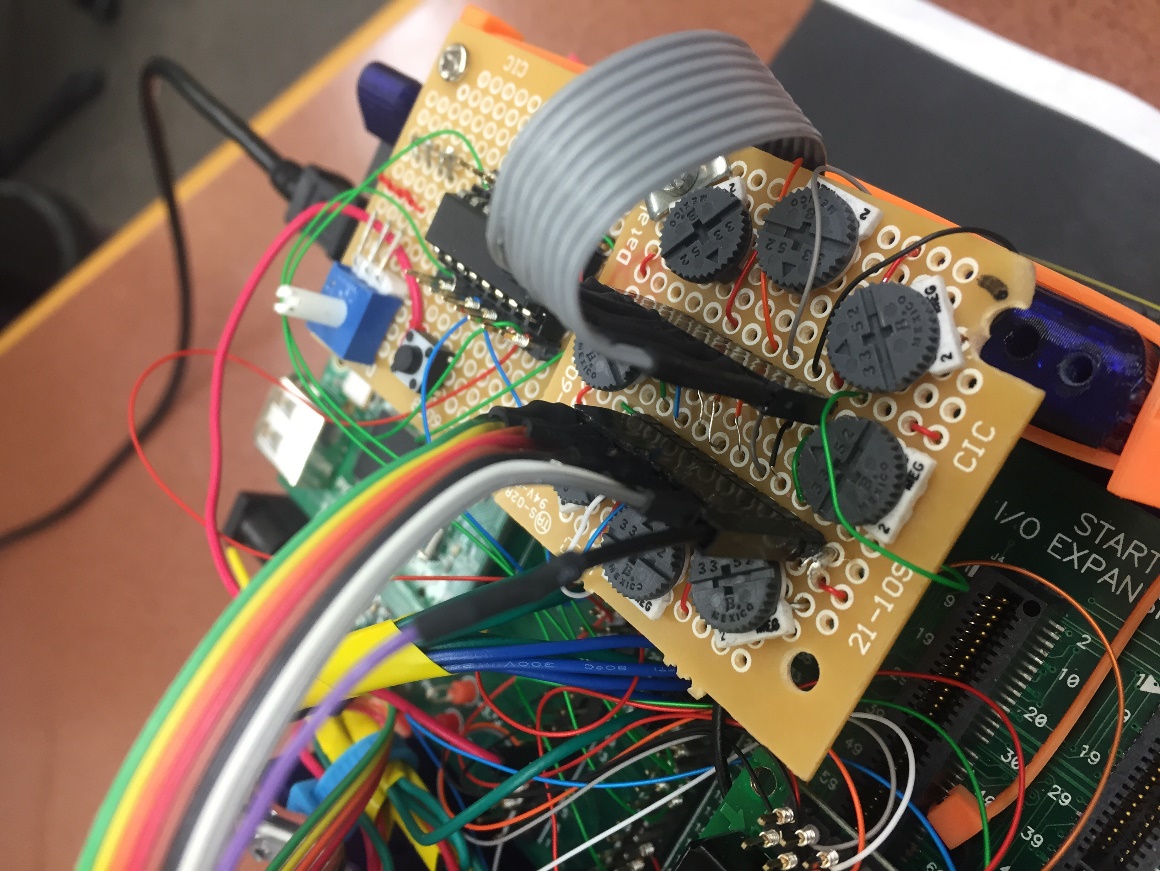
Figure 3 (mirrored): IR LED and sensor array board

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.

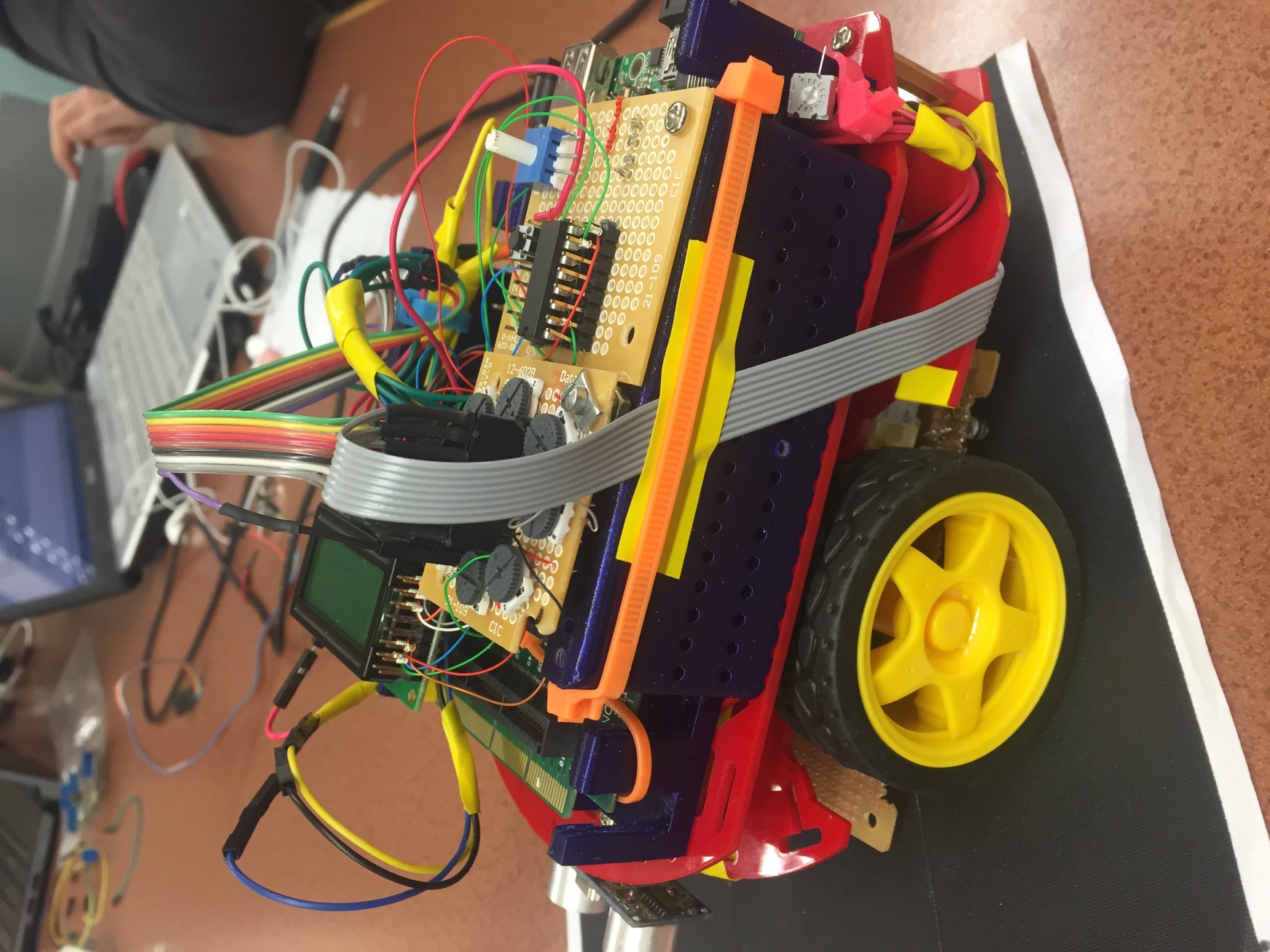
**Figure 5**



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

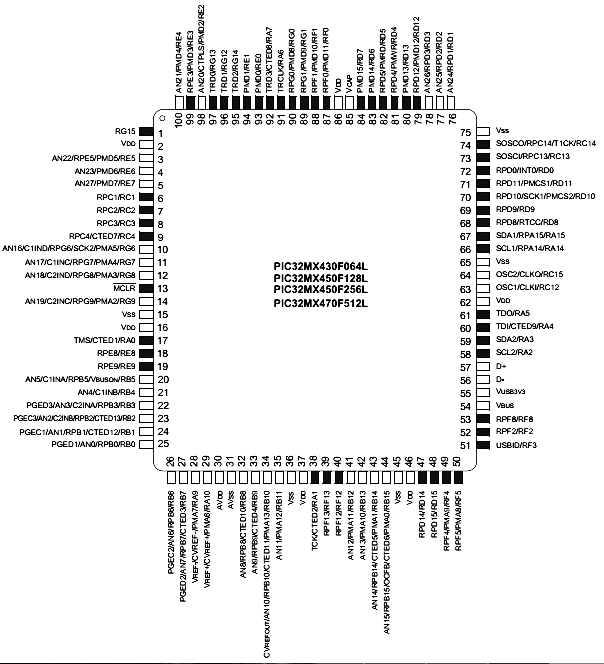


Table 2

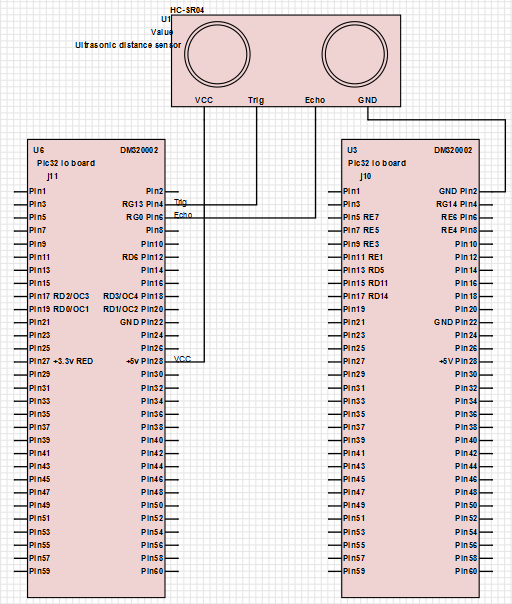


### Circuit Diagrams

Ultrasonic Distance Sensor Circuit Diagram

Figure 7

Ultrasound distance sensor circuit diagram



IR Circuit Diagrams

Figure 8a

IR circuit diagram (connections to expansion board)

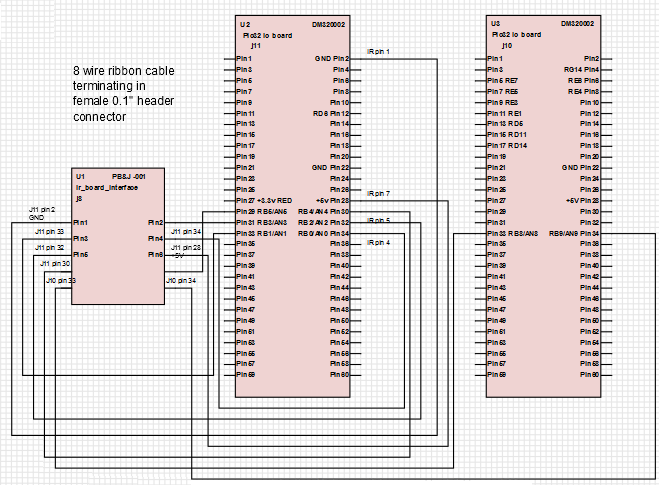
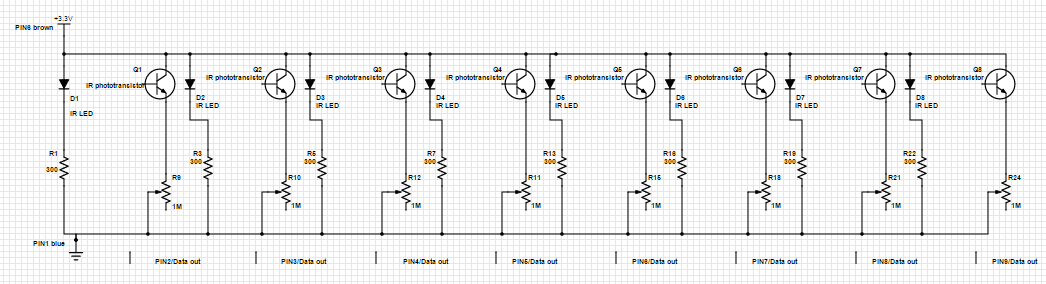
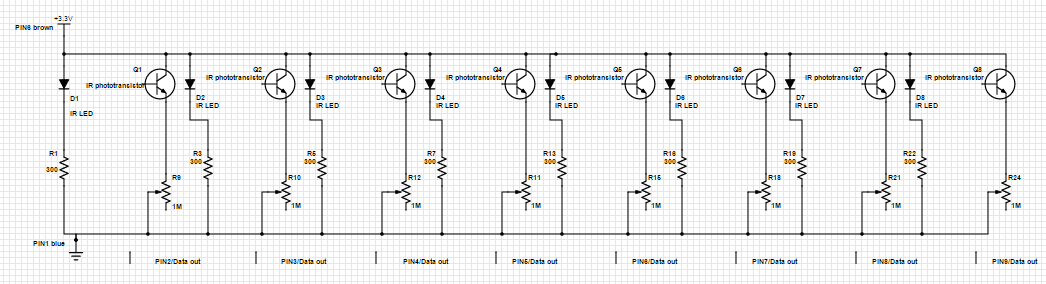


Figure 8b

IR circuit diagram (internal connections)

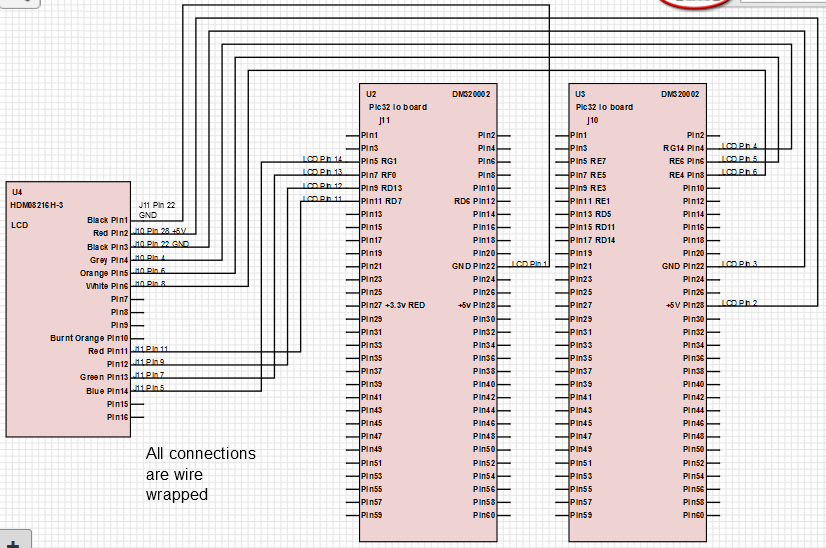




LCD Circuit Diagram

Figure 9

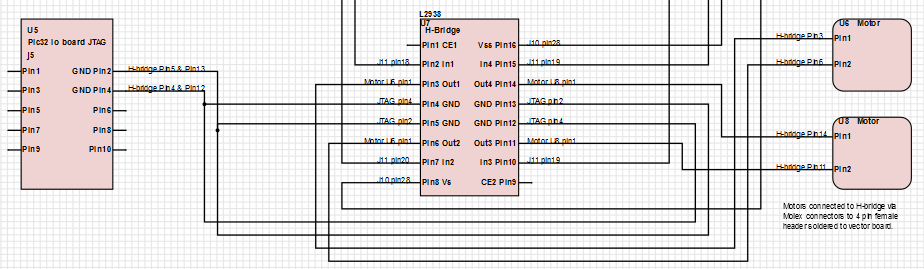
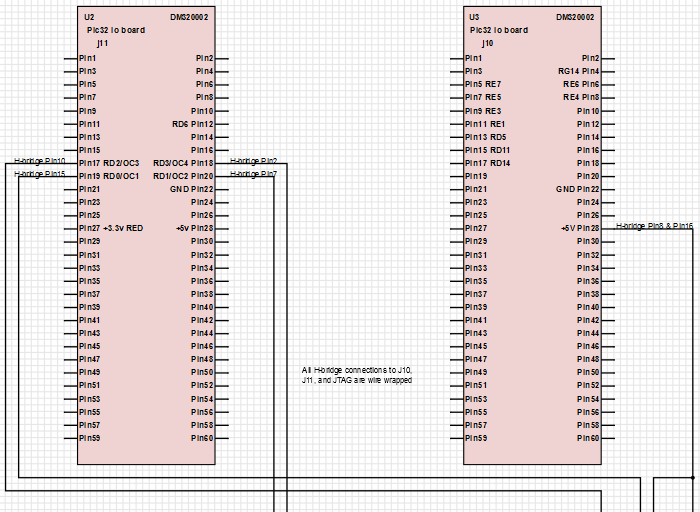
LCD circuit diagram



Motor/H-Bridge Circuit Diagram

Figure 10

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



# Software Design

### Overview

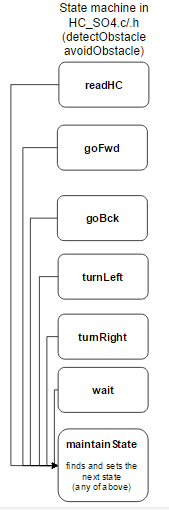
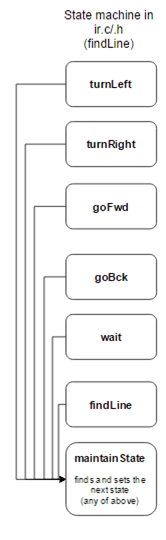
Given the additional complexity of this project when compared to previous labs, the focus of the software design for the final is on scalability. The software is built in a way that enables both easy isolation of the robot subsystems and allows for easy modification of robot behavior at a high level. ‘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 easy. Moreover, we are also defining new functions with the key word “analog” for the updated functions that use analog reading.

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 readAnalogIR()

Read from the IR and return an integer that can be decoded into the values read from each IR sensor, this function has been updated to both allow for analog reads, and to work with our new 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.