Car Lab: Final Project
Infrared Tracking Omnibot

Mike Freyberger and Rediet Desta

May 18, 2015

Contents

1 Introduction

Our final project is an omnibot that tracks Infrared LED's. Using multiple IR LED beacons with unique signatures, the car is able to determine where it is in the room. Then, the user can inform the car where he is by communicating with the car over IP through an android application. Finally, the car is able to drive to the user by controlling it's motion with the two IR receivers that act as eyes, and two ultrasonic pingers that allow the car to avoid obstacles on it's left and right. A full schematic of our project is provided in Figure 1, our final car is provided in Figure 2, and a detailed diagram of our car is provided in Figure 3.

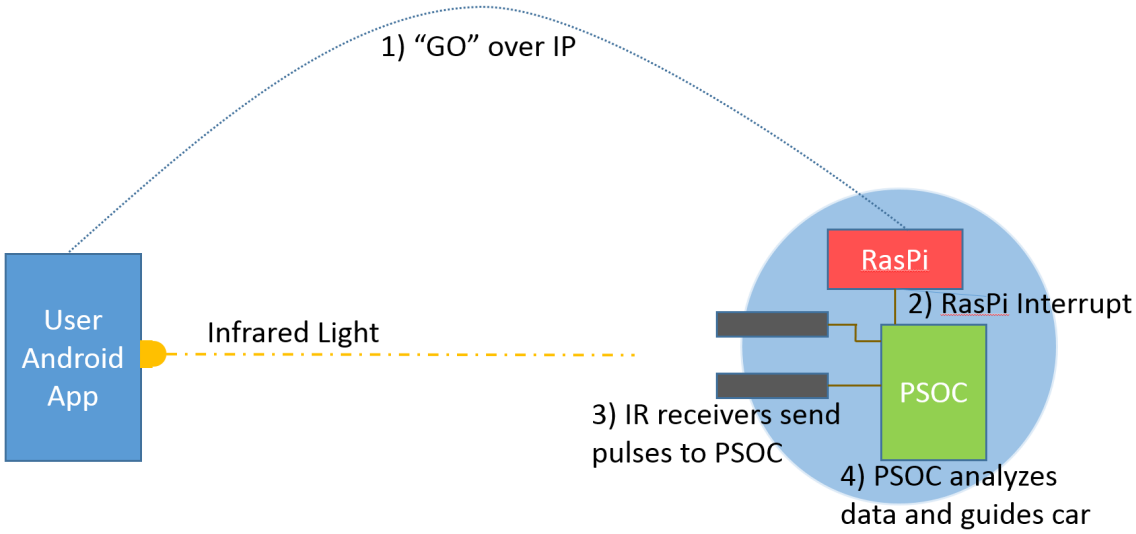

Figure 1: Schematic of the application.

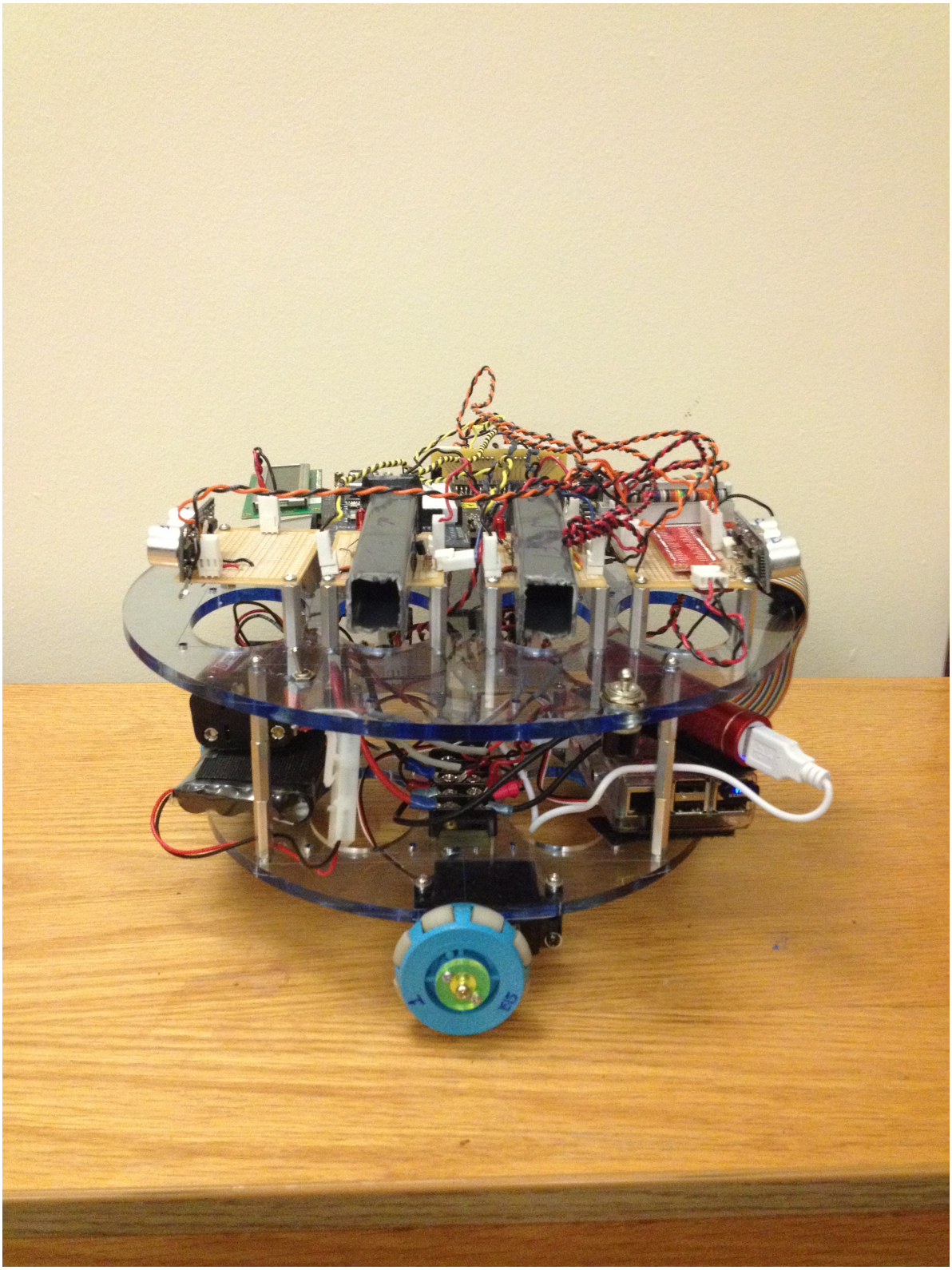

Figure 2: Final product.

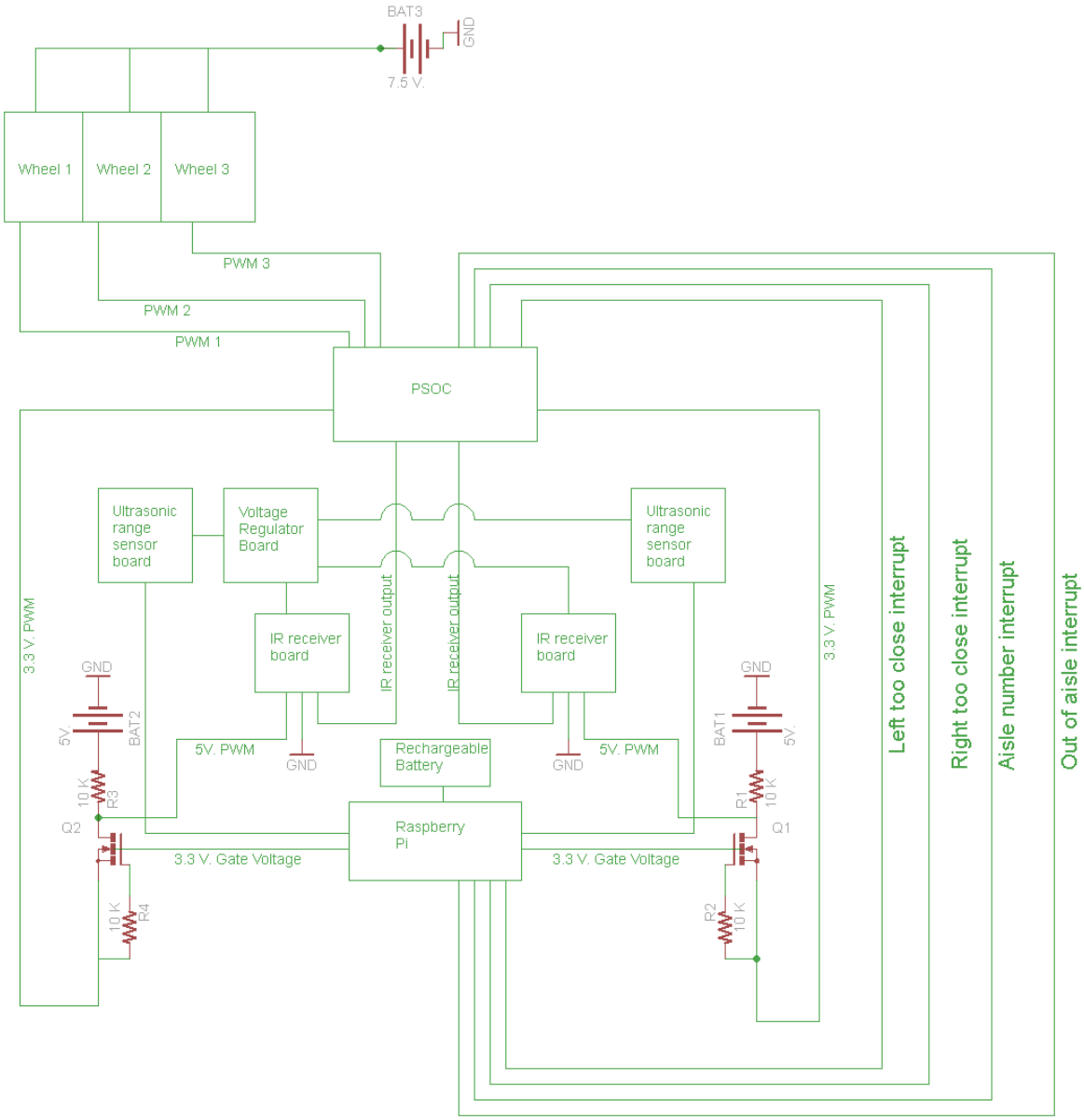

Figure 3: Detailed schematic of all the components on our car.

2 Objectives

To begin this independent project, we set out to create a car that can do the following:

1. Perform point turns.
2. Find and follow an Infrared Light Emitting Diode and measure the envelope frequency of the Infrared LED's.
3. Avoid obstacles when driving in confined spaces, namely the aisles in the ELE undergraduate lab.
4. Begin motion when notified by an Android Application over the internet.
5. Find the user calling the car from anywhere in the ELE undergraduate lab.

3 Objective One: New Chasis

In order to achieve objective 1, we had to change our car completely. Rather than use the typical car that was used for the first two modules, we switched to a 3 wheeled omni bot, as demonstrated by Figure 2. The old 4 wheel version is provided as reference in Figure 3.

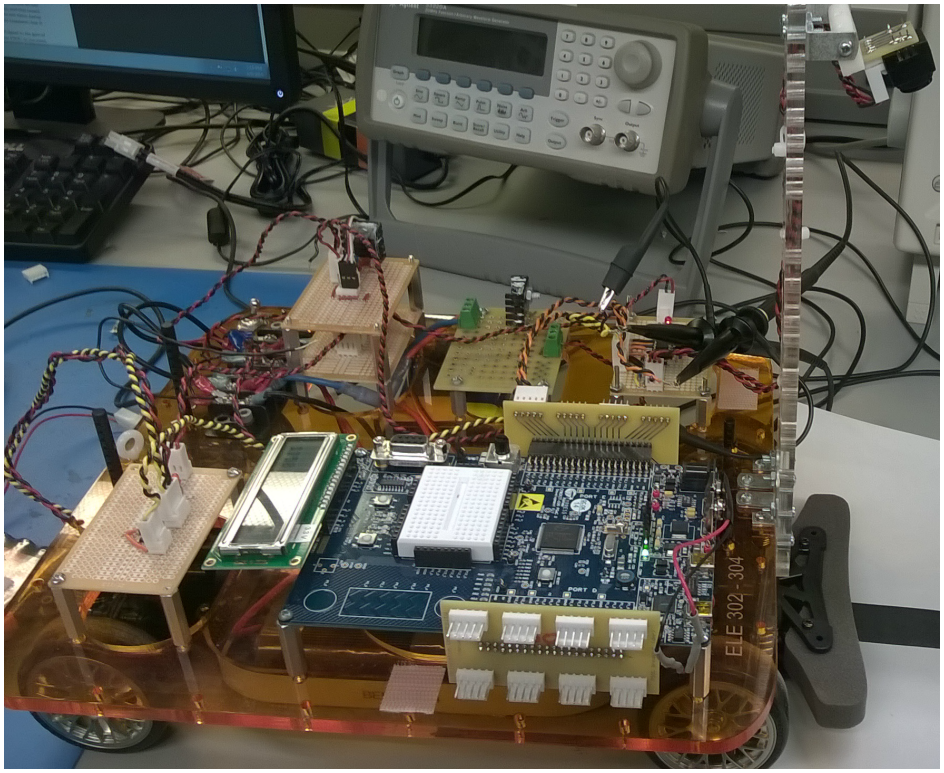

Figure 4: Old car.

3.1 Hardware

The majority of the new chassis was provided for us by our lab manager, David Radcliff. We edited the chassis given to us by replacing normal continuous rotation servos with high power continuous rotation servos.

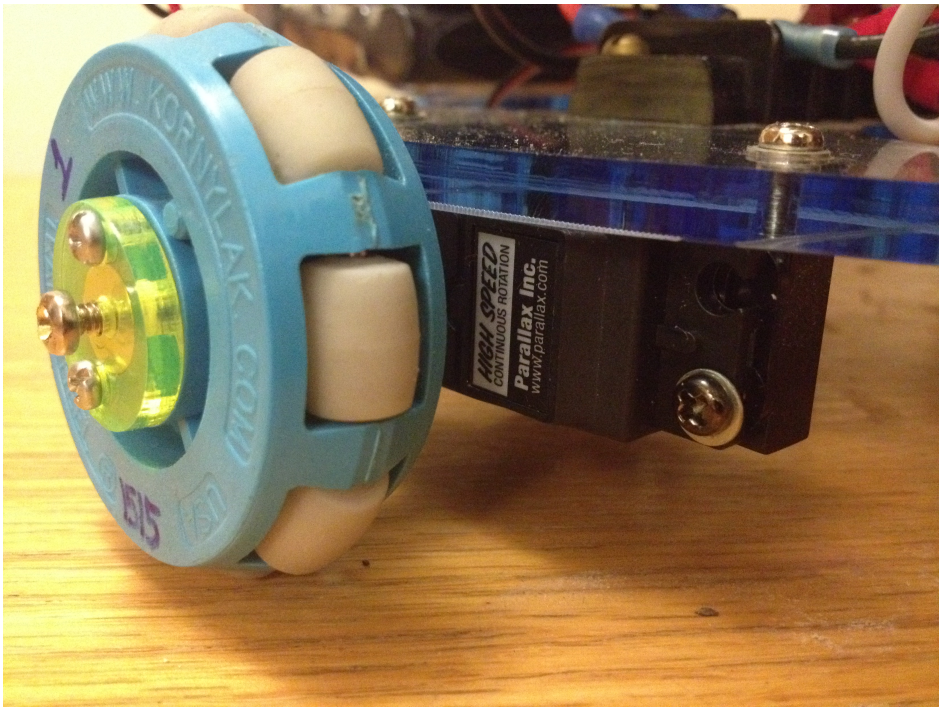

Figure 5: Wheel controlled by a high power continuos rotation servo.

A photo of the wheel with the new high power servo is provided in Figure 5. The high power servo takes an input voltage of 6V to 8V. Therefore, we created out own 7.5V battery pack out of 5 1.5V AA batteries.

Each of the wheels is controlled via a PWM signal. A PWM signal with width 1.5ms makes the tire stop, similar to the servo controlling the angle of the wheels in module 2. The high power continuous rotation servos are set to turn counter clockwise with a PWM width that is greater than 1.5ms, and they turn clockwise with a PWM signal that is less than 1.5ms. The servos can accept a PWM of width in the range from 1.3ms to 1.7ms.

3.2 PSOC PWM

We implement the PWM signals used to control each tire with the PSOC, demonstrated by the PWM components provided in Figure 6. The three PWM signals in the figure are used to control each wheel. Each

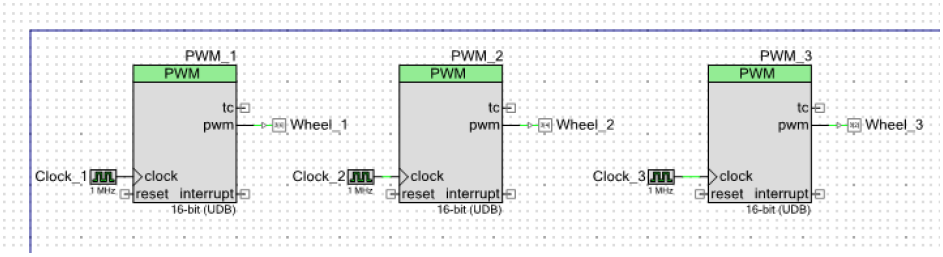

Figure 6: The components used to drive the car.

PWM signal uses a period of 10ms, and connects the pwm output port to a digital output pin that is then connected with twisted pairs to the driving board, which is the board that receives the 7.5V of power from the battery pack, and connect the PWM signal from the PSOC to the input line of the wheel's servo. In order to control the car, we output a specific pwm signal to each tire, and the three vectors combine to make

the car go straight, turn, go backwards, etc. This specific values for each PWM is implemented in PSOC code, and a code snippet is provided in Listing 1.

```
#define stop1 1515 //PWM width for stopping wheel 1
#define stop2 1515 //PWM width for stopping wheel 2
#define stop3 1505 //PWM width for stopping wheel 3

/*  Object for 3 wheel speeds. PWM widths above
    the stop value go counter clockwise, and PWM
    widths below the stop value go clockwise.
*/
struct wheelSpeed {
    const uint16 wheel1;
    const uint16 wheel2;
    const uint16 wheel3;
};

typedef struct wheelSpeed wheelSpeed;

/* ------------------------------------------------------
    Constart vectors for common directions the
    car must go


/*    Stop the car    */
wheelSpeed stopped = {stop1, stop2, stop3};

/*     Rotate clockwise     */
wheelSpeed rotate = {stop1 - 40, stop2 - 40, stop3 - 40};

/* Drive straight. Theoretically wheel 1 and wheel 3 have equal
   and opposite distances from the stop value. Trial and error
   yielded this vector combination to drive straight.
*/
wheelSpeed straight = {stop1 + 190, stop2 - 112, stop3};

/*
    Bear left or bear right. The car continues to move forward,
    but since one tire is rotating the faster, the car turns
    as welll.
*/
wheelSpeed slightLeftWheel = {stop1 + 60, stop2 - 30, stop3};
wheelSpeed slightRightWheel = {stop1 + 30, stop2 - 60, stop3};

/*------------------------------------------------------------*/

/* Static functions used to write the common directions to
   each wheel
*/
static void stopCar() {
    PWM_1_WriteCompare(stopped.wheel1);
    PWM_2_WriteCompare(stopped.wheel2);
    PWM_3_WriteCompare(stopped.wheel3);
}

static void rotateCar() {
    PWM_1_WriteCompare(rotate.wheel1);
    PWM_2_WriteCompare(rotate.wheel2);
    PWM_3_WriteCompare(rotate.wheel3);
}

static void driveStraight() {
    PWM_1_WriteCompare(straight.wheel1);
    PWM_2_WriteCompare(straight.wheel2);
    PWM_3_WriteCompare(straight.wheel3);
}

static void slightLeft() {
```

```
66      PWM_1_WriteCompare(slightLeftWheel.wheel1);
67      PWM_2_WriteCompare(slightLeftWheel.wheel2);
68      PWM_3_WriteCompare(slightLeftWheel.wheel3);
69  }
70
71  static void slightRight() {
72      PWM_1_WriteCompare(slightRightWheel.wheel1);
73      PWM_2_WriteCompare(slightRightWheel.wheel2);
74      PWM_3_WriteCompare(slightRightWheel.wheel3);
75  }
```

Listing 1: Code snippet demonstrating how the car is controlled using three PWM signals.

For each different desired action for the car, an object is created with the PWM signals for each tire. A few examples of these **wheelSpeed** objects can be found in Listing 1. In order to stop the car, the servo's require a PWM signal with a width of approximately 1.52ms. We tested this on each tire and we determined that two of the wheels stopped with a 1.515ms pwm signal, and one tire required a pwm signal of 1.505ms. Therefore, in order to stop the car, we set the width of the pwm signal for each tire to their respective stop width. This is demonstrated in the **stopCar()** function.

The following motions are required for our car to successfully be controlled to the infrared emitter:

- **Rotate**: This allows the car to perform a point turn. Each tire turns clockwise at the same rate by delivering pwm widths that are all .4ms shorter than width required to stop the wheel.
- **Straight**: Move the car in the direction the two infrared receivers are facing. This requires the wheel directly between the two infrared receivers to be still. The back two wheels must be rotating at the same rate; however, one wheel must be moving clockwise, and the other counter-clockwise. When trying to make the car drive straight we found that we could not simply set the back two tires to have the same change in pwm width. After trial and error we determined the pwm widths that actually made the car drive straight. The pwm values are provided in **wheelSpeed straight**.
- **Left and Right** In order to turn left and right, we slow the car down dramatically, and make one tire move faster than other. Once again one tire moves clockwise, and the other clockwise so that both tires are rotating forwards. The exact pwm configurations can be found in **wheelSpeed slightLeftWheel** and **wheelSpeed slightRightWheel**.

Using all of these **wheelSpeed** objects and the three PWM components on the PSOC, we were succesffuly able to fully control the car, and the necessary movements for future objectives were complete.

4 Objective Two: Infrared Control

In order to achieve objective two, we had to build IR LED emitters and receivers. The IR receivers had to interface with the PSOC in order to drive the car.

4.1 Voltage Regulator Board

First of all, the IR receivers on our car need to be powered by 5V; therefore, we use a 5V voltage regulator. For more information on our voltage regulator board, please see our Speed Control Report; a Figure of the voltage regulator board is provided in Figure 7.

4.2 IR Emitter and Receiver

For our Infrared emitters, we used a high-performance infrared emitter LED from RadioShack whose peak wavelength is 940nm, has a forward voltage of 1.28V, rated at 100mA. We could have just attached it directly to a voltage source with the appropriate resistor to limit current and gotten an IR generator. However it is the standard within the industry and for very good reasons to emit IR by coupling it with a carrier wave of

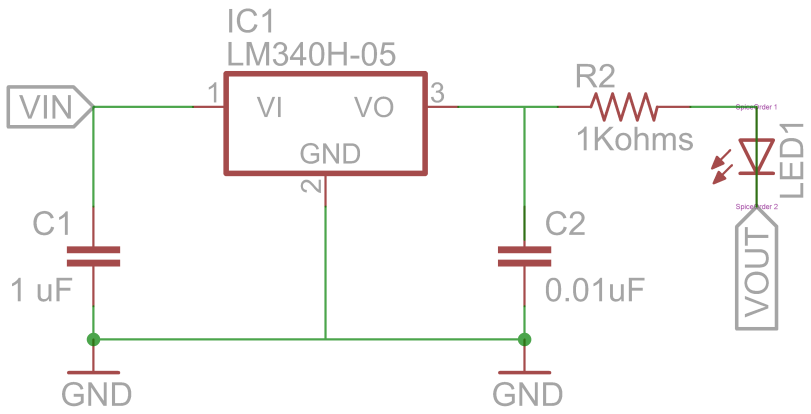

Figure 7: A schematic of the linear voltage regulator LM340-5. Notice the use of capacitors connecting input and output voltage to ground in order to decrease noise in the system.

certain frequency usually ranging between 30-56 KHz. The major reason for this practice is the fact that the IR signal one wants to send through the atmosphere is going to be drowned out with noise. Especially in the lower frequencies very pernicious type of noise known as 1/f noise has predominance which means that a continuous IR signal will easily be lost in the ambient environment. The best and probably the easiest solution to this problem, is embedding a desired signal in a carrier signal with a certain frequency. This approach for reasons we will not delve into here will ensure that our receivers will pick up an IR signal that is not actually noise. Because our IR receivers were calibrated to detect IR waves with a carrier frequency of 38 KHz, we made sure that our IR emitters were being mixed with a carrier frequency that was exactly 38 KHz.

To make this happen in reality we used the classic 555 timer circuit which is widely used to generate square waves of desired frequencies. For our setup and for reasons we will discuss later in this section, we used the LM556 chip which packs into one two 555 timers. We used one of the 555 timer found in the LM556 chip's circuit to generate a 38 KHz square wave.

The 555 timer has three modes of operation, namely the monostable mode, the bistable mode and lastly and the one we exclusively operated under, the astable mode. In astable mode, the 555 timer operates as an oscillator which can produce specific frequencies depending upon the values of external capacitors and resistors connected to its pins. Equation 1 relates the frequency of the square wave generated by a 555 timer and the values of the resistors and capacitors connected to it.

$$f = \frac{1}{ln(2) * C * (R_1 + 2R_2)} \tag{1}$$

To make our system more robust we chose to have our R_2 be a combination of a 10Kohms resistance and a 10K potentiometer. Hence, through this configuration we can easily configure our 555 timer to output exactly 38 KHz without being worried about the inexactitudes that arise from the errors in resistance values and the inner workings of the LM556 chip as a result of errors in the manufacturing process.
Pin 9 is the output of one of the 555 timer's packed into the LM556 chip, demonstrated by the schematic provided in Figure 8. We initially attached the high-performance infrared LED we got from RadioShack to this output in series with a 560 Ohms resistor. However, as we thought more about increasing the range of our IR output, we decided to change the value of the series resistor to a 100 Ohms value resistor so as by decreasing the resistance value, the current through the LED increases thereby increasing the power of the LED which in turn gives us more range. To put this in mathematical terms, we know that power is directly proportional to intensity; more precisely the equation that relates electromagnetic intensity to power is provided in Equation 2.

$$|I| = \frac{P}{A_{surf}} = \frac{P}{4\pi r^2} \tag{2}$$

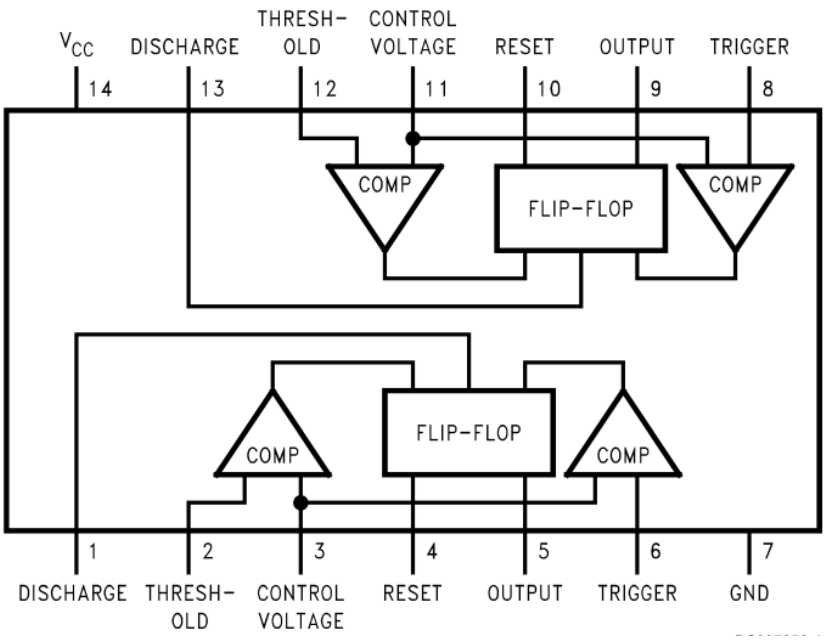

Figure 8: Pin schematic of the 556 timer.

We also know that maximum range of an electromagnetic wave is directly related to the square root of the intensity of the source and is precisely described by Equation 3, where I_e stands for intensity of the electromagnetic wave and E_{emin} stands for the sensitivity of the receiver. Obviously this equation does not take into account other factors that definitely influence the maximum range of an IR emitter's signal such as the reflective conditions of the room, the optical transmittance of windows or light guides in front of the IR receiver and the disturbance conditions. Taking these facts into account we increased the power and hence the range of our IR emitters by decreasing the series resistor's value.

$$d_{max} = \sqrt{\frac{I_e}{E_{emin}}} \tag{3}$$

We want our car to be able to tell its current location and also figure out its next step so as to reach its ultimate goal. We humans have come up with multiple ways of figuring out our immediate location and navigating our way to our destination. Before the arrival of the GPS system, humans used dead reckoning, landmark memorizing techniques and orientation using the sun or night sky to navigate their way through life. It would have been a true blessing if the GPS system could have been used in virtually every type of environment anywhere in the world. Unfortunately that is not the case. Ignoring the fact that GPS requires a tremendous amount of infrastructural setup to run and maintain, it is still the case that it has other more technical limitations. The United States government currently claims 13 feet RMS (25 feet 95% Confidence Interval) horizontal accuracy for civilian (SPS) GPS. This accuracy range is highly unacceptable for mobile in-house systems like ours which require accurate location information ranging from a couple of feet to hundreds of feet. That's why we decided to use an IR emitting-receiving system as it provided us with certain advantages that can't be provided by a GPS system.

Our car navigates its way around a given perimeter using the old techniques humans have been using before the arrival of the GPS system. Even though our car doesn't have an integrated system that tells it its compass direction; it takes full advantage of landmark memorization techniques and dead reckoning to find its way. In our system we use the IR emitters as landmarks which the receivers on the car are able to recognize uniquely.

For each emitter to be uniquely identified we understood that we needed to embed additional information in the carrier frequency of the IR signal; information that can help us distinguish each emitter uniquely. Since

our IR receivers can detect IR signals modulated with a limited but arbitrary range of envelope -frequency on top of the required 38 KHz frequency, we decided to further modulate our IR signal with frequencies that range from 0.05 KHz to 2 KHz. Hence we actually used two carrier frequencies for our IR emitters: the first one being the required 38 KHz frequency which is further modulated by a second carrier frequency, unique for every emitter board. The signal generated looks like the one shown in Figure 9. This is why we decided

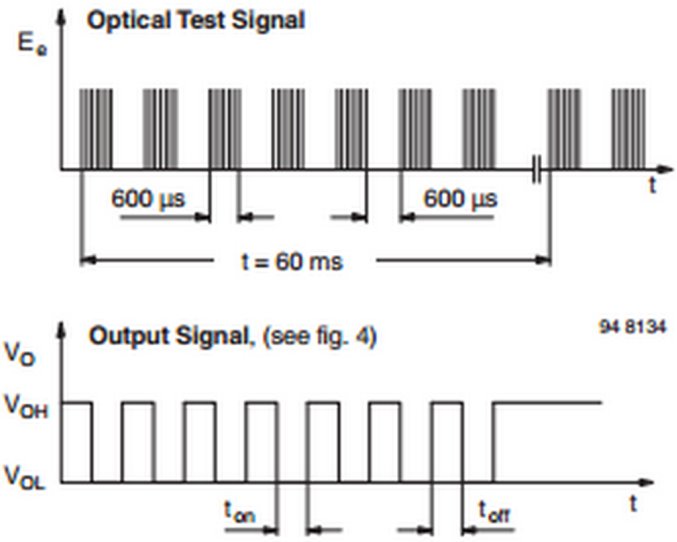

Figure 9: The IR receiver is only on when it is reading infrared red modulated at 38KHz.

to use the LM556 chip; because we needed to create and ultimately mix two square waves with different frequencies. It meant that in one board we can neatly fit one chip which on one side has the circuitry that produces the common 38 KHz frequency and on the opposite side have the circuitry that generates the unique frequency of that particular emitter board.

The choice of our second envelope frequency was limited by our receiver's acceptable range of envelope frequencies with specific duty cycles just as the 38 KHz envelope frequency was chosen because of our IR receiver's specifications.

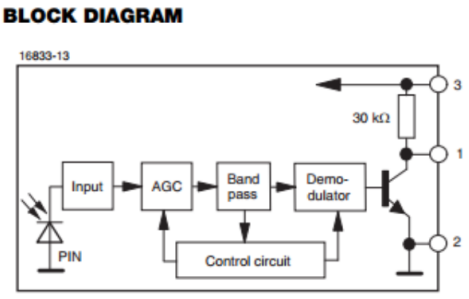

Figure 10: The inner workings of the IR receiver. The important component for our case is the AGC block.

When designing the receiver board, our major concern was to select a receiver sensor which has a very robust noise cancellation system. IR is emitted by various objects; mostly objects that are not necessarily designed to emit it. IR signals are emitted from bulbs and fluorescent lights and really any object which is at a very high temperature. The receiver we finally decided to use for our system, TSOP3438 from Vishay Semiconductors, has a very robust noise cancellation mechanism that uses Automatic Gain Control (AGC). AGC is used to suppress spurious output pulses due to noise or disturbance signals. The AGC module inside the receiver is demonstrated in Figure 10. This means that if the receiver thinks the incoming data signal is mixed with a significant amount of noise, AGC kicks in and reduces the sensitivity of the receiver thereby preventing the output of a spurious pulse in response to an incorrectly recognized signal.

Vishay Semiconductors has five different types of AGC algorithms that differ from each other in the range of frequencies and duty cycles they consider to be proper IR signaling. Our receiver, the TSOP3438 has an AGC2 system. As shown in Figure 11 the range of envelope frequencies that an AGC2 receiver accepts is roughly between 50 Hz to 2 KHz. We built multiple emitter boards with unique envelope frequencies that ranged from 400 Hz to 2 KHz, but the ones that worked really well had frequencies of 400 Hz, 600 Hz and 1 KHz.

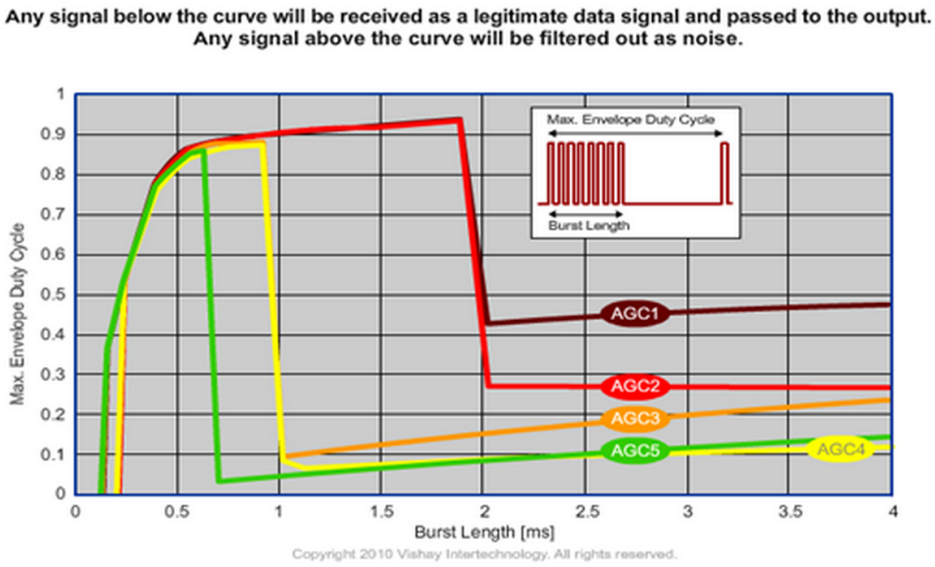

Figure 11: The AGC curves demonstrating the envelope frequencies that we can use.

After building our emitter boards and had determined the type of IR receiver we wanted to use we focused our attention to building the receiver board. The external circuitry built around the IR receiver is very simple in that we only connected the input voltage pin of the receiver to a resistor of 330 ohm and an LED to protect the receiver during electrical overstress and gauge the proper delivery of power to it.

However, soon enough we encountered some problems that forced us to make the external circuitry more robust. As stated in previous sections, we were convinced from the beginning of our project that we needed to build two receiver boards with two IR receiver boards perched on top. This kind of setup with two receivers looking out for an incoming IR signal, if properly implemented, would be crucial in the proper control and guidance of the car to a desired location. Two receivers meant that we have the capability to tell the orientation of the vehicle with respect to the source of an IR signal. If one of the receivers detects an IR signal and the other one doesn't, it means that our car is oriented in a certain way and we can respond appropriately as will be described in more detail in our discussion of the algorithm that we use to handle various cases encountered by the vehicle. However building a system that ensures that in the appropriate positioning only the proper receiver detects the signal is not as simple as it sounds.

Imagine two receivers on two receiver boards implemented with the proper circuitry and ready to detect an IR signal. Let's say that an IR signal is being emanated from an IR emitter board a few feet in front of the two receivers and positioned significantly to the right of the center of the two boards. In this particular scenario, ideally, we want the receiver perched to the right relative to the other receiver should detect it while the left receiver is not detecting any IR signal. However an IR signal is emitted in all directions from our High output IR emitter which means there is high probability that both IR receivers will detect the signal, an undesired behavior. Not only that, IR being an electromagnetic wave will go around corners and bounce off various objects to unfortunately be detected by the receiver positioned at the wrong direction.

To ensure that our IR receivers detect an IR signal only when it is either directly facing them or is veered at a slight angle to the right or left of them, this case depending upon the receiver's we began to look for ways of enclosing them in a protective covering that blocks out IR waves that come in from all the wrong direction and selectively lets in only IR signals coming in from desired directions. We noticed that anything from a cardboard to a human body can block IR signals from our emitters hence there was no big constraint on the type of material. After a bit of searching we stumbled upon a two rectangular tubes made out of hard plastic. After of bit of thinking and sketching we decided to drill a hole into one of the sides of both rectangular tube so as to house the receiver inside the tube after sealing the backside of and any holes that might be present only leaving one hole for any type of signal to gain access to the receiver. The tubes we used to house our receivers are shown below. Housing the tubes really helped in making the receivers very directional only detecting IR signals coming from emitters placed at desired directions relative to the center of the two receiver boards. Now this system was not perfect. Sometimes the wrong receiver still detected IR signals coming from an IR emitter. However this happened so infrequently that it was not affecting the performance of our system significantly. In the event that this type of error leads our car astray, we have set up algorithms that ensure that the vehicle still ultimately ends up achieving its goal.

Once we dealt with the directionality of our IR receiver circuits there was another inconsistency in the response of the IR receiver response to apparent exposure to a legitimate IR signal from one of our emitters. The issue was very curious in that it appeared at very specific instances in our test of our IR receiver circuit. Let us describe the situation in detail. Imagine that we place an IR emitter which is emitting IR at 38 KHz frequency with an envelope frequency of 1 KHz, right in front of one of two IR receiver boards. As expected, our IR receiver sensor easily pick up the signal and dutifully turn on and off at a frequency of 1 KHz. Imagine now we began to slowly place the emitter board further and further away from the receiver board while still maintaining the same line of sight. As mentioned in earlier paragraphs the range of our IR emitters is around 40 feet. Hence as the emitter gets further and further away from the receiver and the strength of the IR signal reaching the receiver begins to decrease significantly the ever powerful AGC system of the receiver kicks in and stops detecting the received IR signal as by that time it is highly mixed with a significant amount of noise. This behavior is acceptable for our system. However something curios happen right after this apparent rejection of the emitted IR signal happens after a certain range. When we put the

emitter back into the range in which the emitted IR signal had previously been detected by the receiver, all of a sudden the receiver no longer seems to accept it.

After some experimentation, we noticed that this behavior can be reversed so that the receiver accepts as a proper signal the IR signal being emitted by the emitter by turning off then on the power supply of the IR receiver circuit. It seems to us that after the AGC2 algorithm starts thinking that a certain signal is noise and rejects it there is no going back unless the system is reset to the default state and allowed to analyze signals all over again. After we confirmed that this very manual procedure of turning on/off of the receiver circuit solved the problem we decided to make it an automatic process by replacing the constant 5 V. supplied to the receiver board by a PWM signal of 5 V. max value with a specific duty cycle. We used our PSOC PWM generating ability to power our receiver boards. However the PSOC only outputs PWM signals of maximum value of 3 V. Hence we used a logic level converter to amplify the 3V. signal received from the PSOC to a 5V. signal to power the receiver.

The final schematic for our IR emitter is provided in Figure 13, and the final schematic for our IR receiver is provided in Figure 12.

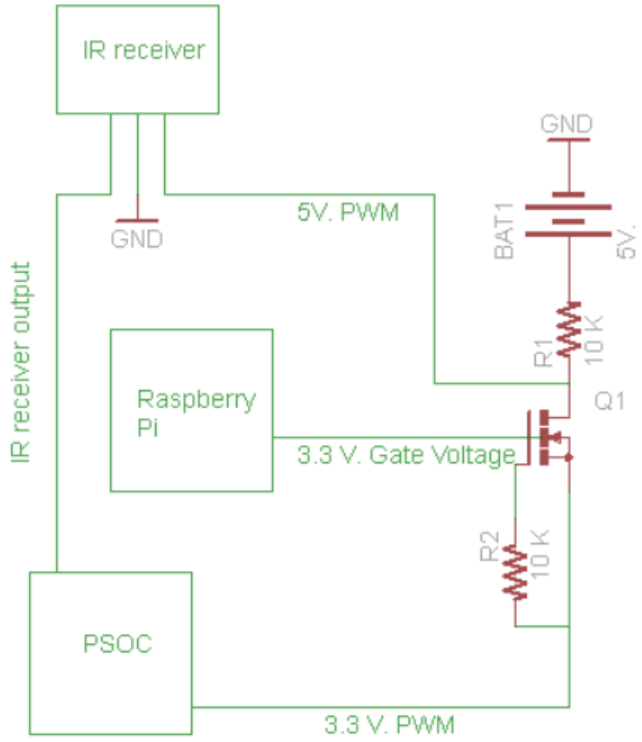

Figure 12: The IR receiver.

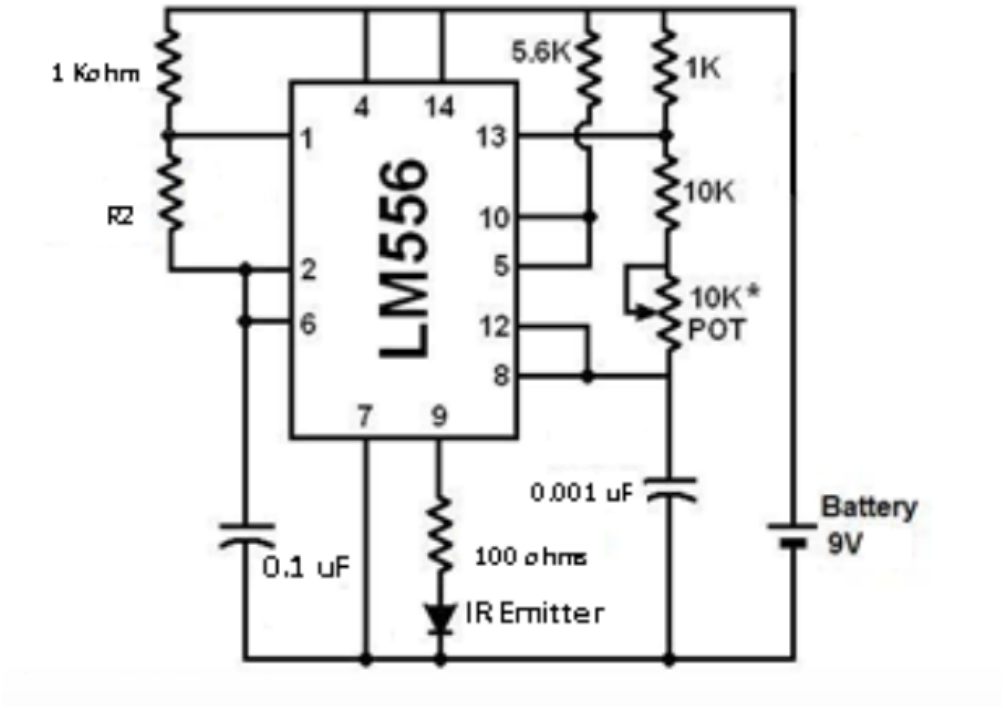

Figure 13: The IR emitter. The schematic is edited from the schematic found at botmag.com

4.3 PSOC

The components used to control the car and respond to the output of the IR receivers is provided in Figure 14.

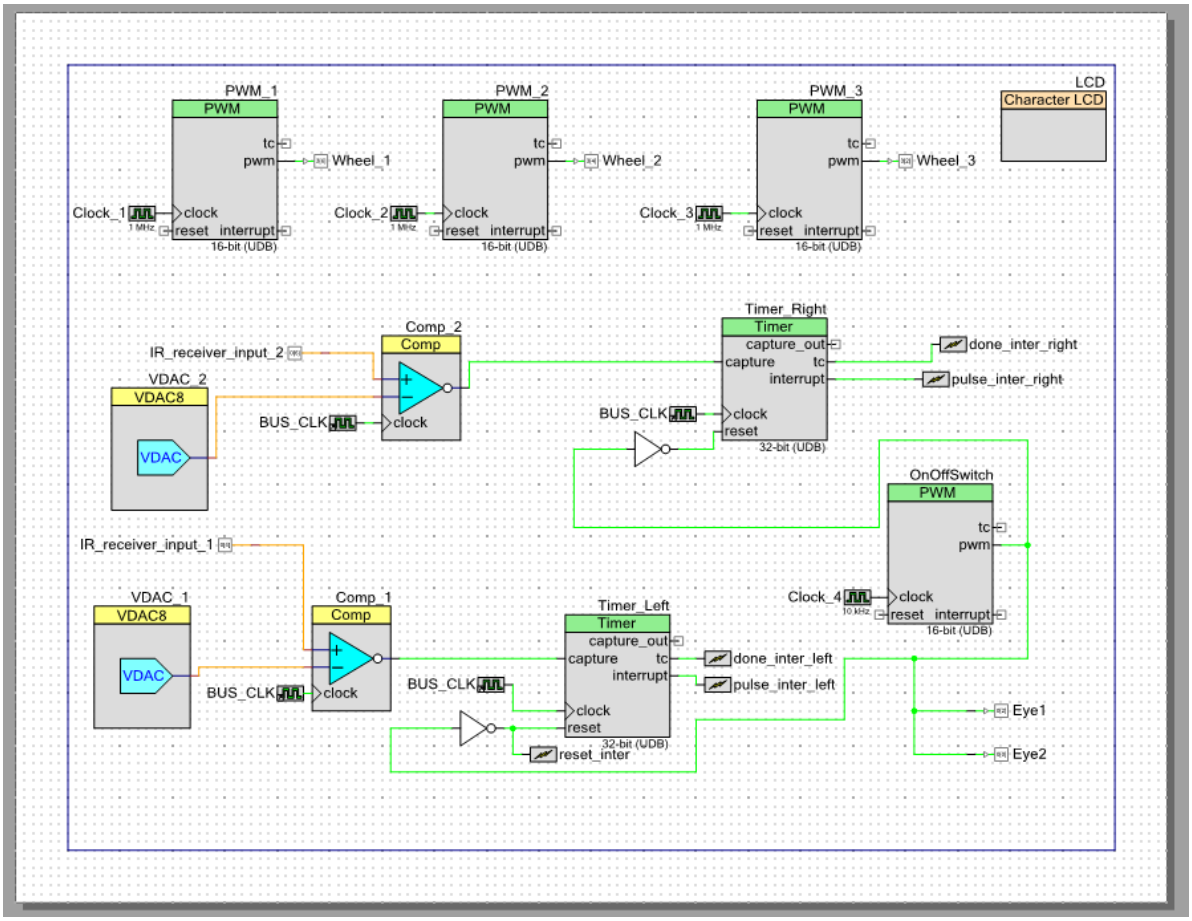

Figure 14: The components used for driving control.

- **Wheel PWM Signals**: The three PWM signals at the top of the figure are the signals that control each wheel. A full discussion of how to control the car using these signals is provided in Section 3.
- **LCD Screen**: The LCD screen is used for debugging. When the car is in various states, we print information to the screen in order to understand what the car is currently doing.
- **IR Input**: Analog signal that is delivered to the PSOC from the output of the IR receivers on the car. We have two IR Inputs (one for each eye). This signal takes on values between 3V and .5V. The signal goes low when there is a valid 38KHz IR Light that it is seeing. The IR receiver signal will have the same frequency as the envelope frequency of IR emitter.
- **VDAC**: In order to determine when the IR signal goes low, we make the input signal analog, and use a reference voltage. The reference voltage is set to 1V.
- **Comparator**: This is used to determine if the IR input signal is greater or less and 1V. When the signal goes below 1V, the comparator goes high. This is considered an inverted comparator; however, this is necessary for our situation because when the IR signal goes low, that indicates there is the receiver is seeing a valid signal.

- **Timer**: The timers are used to measure how many times the IR receiver sees a valid signal during the time interval of the Timer, which is 100ms. The output of the comparator is connected to the capture of the timer, and for each capture we fire an interrupt. The interrupt service routine is very simple, and just increments a counter which is storing the number of pulses for each time interval. Since the time interval is is 100ms, we expect the number of pulses to be roughly $\frac{1}{10}$ of the envelope frequency of the emitter. For instance, if the emitter's envelope frequency is 1KHz, we expect to see 100 pulses in a 100ms time interval. In order to know when the 100ms time interval is complete, the Timer sends an interrupt when it is done. When the done interrupt fires, we analyze the number of pulses during that time interval, and respond appropriately.
- **PWM - On Off Switch**: As mentioned in the previous section on measuring frequency, there is a persistence issue with the IR receivers that is fixed by turning the receivers on and off. This is achieved the PWM component called **OnOffSwitch**. This PWM signal signal outputs 3.3V for 110ms, and then outputs 0V for 10ms. When the PWM signal is high, each of the two output pins, **Eye1** and **Eye2**, turn the IR receiver on. Therefore, the receiver is on for 110ms. This is enough time to capture a full pulse count measurement, which takes 100ms, as specified above. The output of the PWM then goes low, which turns the IR receivers off for 10ms. The inverse of the output of the PWM is also connected to the reset input to each timer. This allows the Timers to be on when the IR receiver is on, and when the IR receiver's turn off for 10ms, the Timers reset. This allows each timer to begin the 100ms measurement at the same time the IR receivers are turned on.

With these PSOC components we are able to control the car, and make the car follow an IR LED. Figure 15 provides the finite state is machine is used to control the car. The car begins by rotating until each eye has

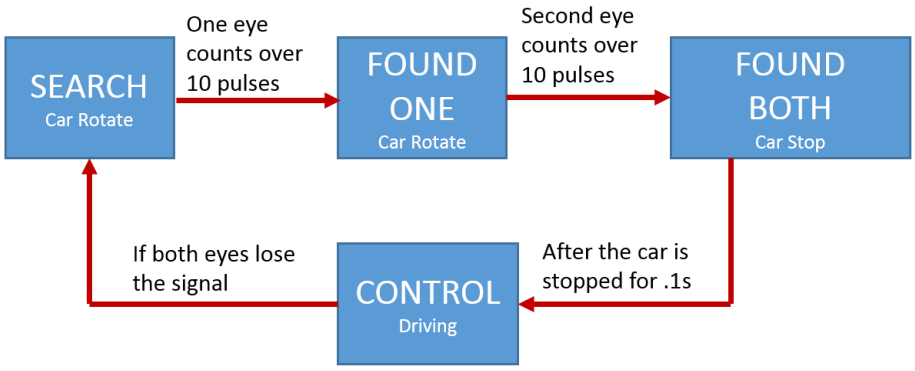

Figure 15: The finite state machine used to control the car.

seen 10 pulses, which is simply a minimal threshold that will always be exceeded by an emitters envelope frequency, but will reject noise that can cause the count of pulses to be a small value even when there is no LED in sight. Therefore, once each eye has seen 10 pulses in a 100ms interval, we can be confident that the car is facing an IR LED. At this point the car begins to drive towards the LED source using something similar to bang-bang control. Essentially what happens is that when one eye can no longer see the signal, the car turns so that the eye that is currently missing the signal will see the signal. When both eyes are seeing the signal, we drive straight. If both eyes lose the signal, we make the car rotate, and go back to the searching state. The code used to control the car is provided in Listing 2

```
static void controlCar() {
    LCD_ClearDisplay();
    LCD_Position(0,0);
    if (left.on == true && right.on == true) {
        LCD_PrintString("Both");
        driveStraight();
    }
    else if (left.on != true && right.on == true) {
```

```
9           LCD_PrintString("RIGHT");
10          slightLeft();
11      }
12      else if (left.on == true && right.on != true) {
13          LCD_PrintString("LEFT");
14          slightRight();
15      }
16      else {
17          LCD_PrintString("Lost");
18          state = SEARCH;
19          rotateCar();
20      }
21  }
```

Listing 2: Code snippet demonstrating how the car drives when in the control state.

Now the car can drive towards an IR LED when it sees any IR emitter. However, the car also must be able to identify the envelope frequency that is currently looking at. In order to successfully measure the envelope frequency, we add two states to the FSM which measure the frequency in each eye. The FSM used to control the car and measure the frequency is provided in Figure 16. These two states measure the frequency

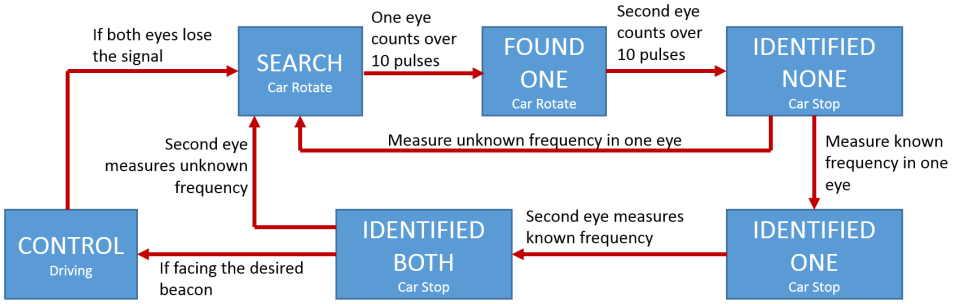

Figure 16: The finite state machine used to control the car and measure the envelope frequency.

that each eye sees. In order to measure the frequency, the eye must see 10 consecutive pulse counts. This leads to measuring the frequency by simply counting the number of pulses in a 1 second interval, which is the definition of frequency. At first, the desired beacon is only the beacon with an envelope frequency of 600Hz, because that is the emitter that will attached to the android, which will eventually be used to initialize the car. Therefore, if each eye measures approximately 600Hz, the car will start to drive towards the IR LED. If the car sees any another frequency, it will begin searching again for the 600Hz envelope frequency.

With all of the Infrared components, and the PSOC developed, the car is now able to perform a significant amount of the tasks required for our final project. We now need to make sure the car can drive backwards when it cannot see a signal.

5 Objective Three: Avoid obstacles in aisles.

A main part of out project is making the car exit one of ELE undergraduate aisles, and then find the user. In order to do, the car must be able to identify it is in the aisle, which is now possible since we can measure the frequency of IR emitters. For instance, the envelope frequency of the emitter at the end of the aisle is 1KHz; therefore, if the car measures this frequency, the car knows it is in the aisle. Once the car knows it is in an aisle, we want the car to be able to drive out of the aisle.

At first we thought this would be as simple as rotating the car until it sees the beacon at the end of the aisle, and then just drive backwards, without any control. However, this approach failed very quickly because the car would not identify the beacon when it is perfectly in line with the beacon. Therefore, if the car is slightly rotated when it identifies the signal, the car will drive backwards directly into a desk or a

chair. In order to prevent crashing when the car is driving backwards, we implemented ultrasonic pingers. These pingers are used to identify when the car is too close to an object and when the car is in the out of the aisle.

5.1 Hardware

One of the major issues one faces when implementing a system that is constantly on the move is how to make it detect and avoid obstacles that will hinder it from reaching its ultimate goal. Unlike Google we don?t have the resources to acquire for our car, Velodyne?s high-tech HDL-64E LiDAR laser range sensor which according to the company packs in "64 fixed-mounted lasers to measure the surrounding environment, each mechanically mounted to a specific vertical angle, with the entire unit spinning". Our best options after a bit of online searching in terms of reliability and best-fit technology coupled with reasonable pricing were IR proximity sensors or ultrasonic range sensors.

We ultimately decided to use ultrasonic range sensors in our system mainly because IR proximity sensors are very prone to errors of distance calculation depending upon the types of surfaces, colors and shades they are directly facing. On top of this we are already using an elaborate IR emitting and sensing system which means there is a lot IR wave bouncing around the vicinity of our system already, increasing the probability of error the IR proximity sensors even more. Moreover in terms of the range these sensors can detect at least the IR proximity sensor we were looking into, namely the Sharp Infrared Proximity Sensor -GP2Y0A21YK has the meager range of 2.6 feet.

In contrast the ultrasonic range sensors we were looking into, namely, the Ping Ultrasonic distance sensor from Parallax has an integrated technology which generates an ultrasonic burst and uses the time of arrival of the echo to calculate the distance to the nearest object facing it. It can also detect objects that are as far as 9.8 feet which works well with our system. The interfacing protocol of the system is very simple and intuitive. The sensors will not generate an ultrasonic burst until a "Go" signal is sent from the user which is just one pulse of any duty cycle. Once it receives the "Go" signal the sensor sends out a short 40 KHz ultrasonic burst, receives back the echo and sends an output pulse that has a width that is equivalent to the distance travelled by the ultrasonic ping signal. These three features led us to make use of the ultrasonic range sensor to determine distance between the nearest object to our vehicle hence helping us detect obstacles. The circuitry involved is very simple as is shown in Figure 17. The actual pinger on our board is provided in Figure 18.

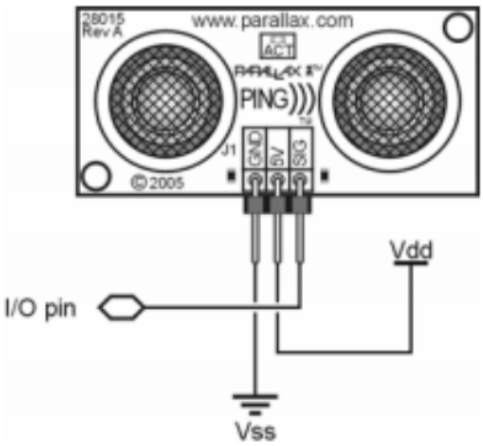

Figure 17: Pinger schematic.

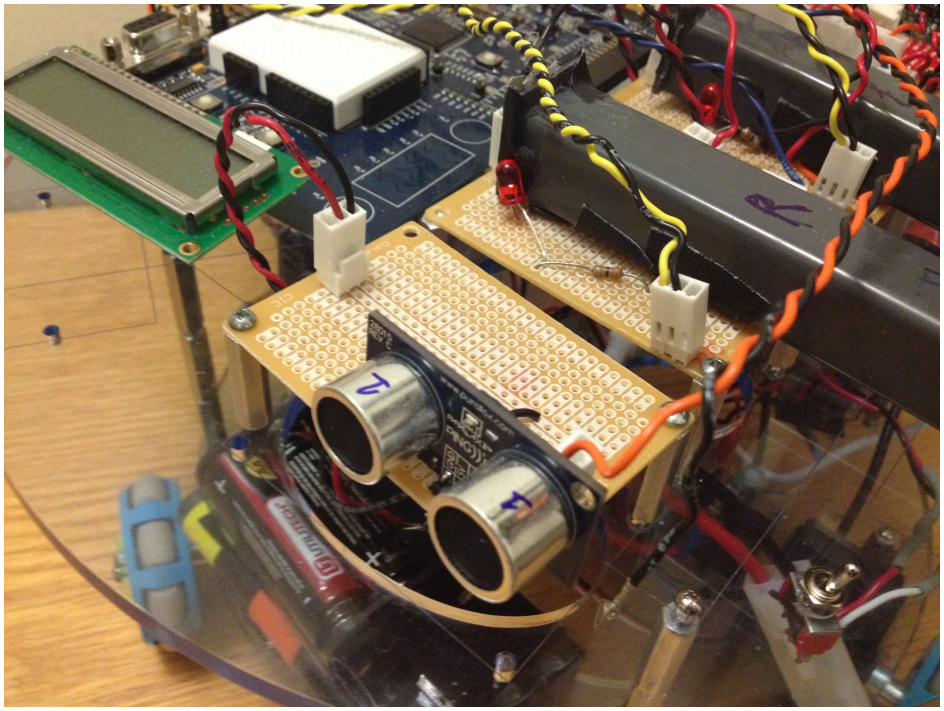

Figure 18: A photo of our pinger.

5.2 Software

Interfacing with the pingers is very simple. The two pingers that we use are a little different; however, interfacing with them in code does not change dramatically. The main idea involve soutputing a high signal for a small amount of time on the trigger pin. Then listen on the echo line for a pulse. The width of the pulse corresponds to the distance from the pinger to the nearest object. Given the timing measurement of the pulse, the distance between the pinger and the nearest object is calculated in centimeters. Using this distance measurement, we determine if the car is too close on either side to an object, or if the car is out of the aisle.

When the car is too close on one side or the other, the raspberry pi sends an interrupt to the PSOC. The PSOC responds to this interrupt by turning the correct direction, re straightening, and then continuing to back up straight until it sees another interrupt from the pingers. The driving commands used to slightly turn the car when driving backwards is provided in Listing 3. This code snippet also demonstrates how we use these turn commands to make sure we avoid obstacles on our left and right when exiting the aisle.

```
wheelSpeed backwards = {stop1 - 136, stop2 + 106, stop3};
wheelSpeed slightRightWheelBack = {stop1 - 60, stop2 + 30, stop3};
wheelSpeed slightLeftWheelBack = {stop1 - 30, stop2 + 60, stop3};
wheelSpeed slowRightRotateWheel = {stop1 - 40, stop2 - 40, stop3-40};
wheelSpeed slowLeftRotateWheel = {stop1 + 40, stop2 + 40, stop3+40};

static void driveBackwards() {
    PWM_1_WriteCompare(backwards.wheel1);
    PWM_2_WriteCompare(backwards.wheel2);
    PWM_3_WriteCompare(backwards.wheel3);
}
static void slightLeftBackwards() {
    PWM_1_WriteCompare(slightLeftWheelBack.wheel1);
    PWM_2_WriteCompare(slightLeftWheelBack.wheel2);
    PWM_3_WriteCompare(slightLeftWheelBack.wheel3);
}
```

```
static void slightRightBackwards() {
    PWM_1_WriteCompare(slightRightWheelBack.wheel1);
    PWM_2_WriteCompare(slightRightWheelBack.wheel2);
    PWM_3_WriteCompare(slightRightWheelBack.wheel3);
}
static void slowRightRotate() {
    PWM_1_WriteCompare(slowRightRotateWheel.wheel1);
    PWM_2_WriteCompare(slowRightRotateWheel.wheel2);
    PWM_3_WriteCompare(slowRightRotateWheel.wheel3);
}
static void slowLeftRotate() {
    PWM_1_WriteCompare(slowLeftRotateWheel.wheel1);
    PWM_2_WriteCompare(slowLeftRotateWheel.wheel2);
    PWM_3_WriteCompare(slowLeftRotateWheel.wheel3);
}
/*-----------------------------------------------
  This is the state the car is normally in
  when backing up. It just drives straight
  backwards until one of the pingers tells
  it to do otherwise.
-----------------------------------------------*/

else if (state == EXIT_AISLE) {
    /* Set to zero. Need counter var during backup */
    left.count = 0;
    right.count = 0;
    LCD_PrintString("Exit");
    driveBackwards();
}

/*-----------------------------------------------
  The car is only in one of these correction
  states if the pingers sends an interrupt
  to the PSOC, notifying the PSOC it is too
  close to an object on either the left or
  right side.
-----------------------------------------------*/
else if (state == CORRECT_RIGHT) {
    right.count++;
    /* Turn right and go backwards for 8 seconds. */
    if (right.count < 80) {
        slightRightBackwards();
    }
    /* Rotate left to straighten out for .5 seconds */
    else if (right.count < 85) {
        slowLeftRotate();
    }
    /* Drive straight backwards once turning scheme is over */
    else if (right.count == 85) {
        state = EXIT_AISLE;
    }
}
else if (state == CORRECT_LEFT) {
  /* Turn left and go backwards for 8 seconds. */
    if (left.count < 80) {
        slightLeftBackwards();
    }
    /* Rotate right to straighten out for .5 seconds */
    else if (left.count < 90) {
        slowRightRotate();
    }
    /* Drive straight backwards once turning scheme is over */
    else if (left.count == 90) {
        state = EXIT_AISLE;
    }
}
```

Listing 3: Code snippet demonstrating how the car turns when driving backwards.

Something that would be improved in an update version would be to include a pinger on the back of the car as well. The current implementation only avoids obstacles on the left or right. It'd be best to avoid obstacles behind the car as well.

The last thing that must be done is notify the PSOC when the car is out of the aisle. This is completed by another interrupt line that fires when either pinger measures a distance that is greater than 3000cm. When this interrupt is fired, the car moves to a new state in the finite state machine, and the car continues to find the user. The final steps used once the car is out of the aisle is discussed in Section 7.

6 Objective Four: Communicate between Android Application and Raspberry Pi.

Setting up the wireless communication between the android and the Raspberry Pi requires IP network programming. The code used on the android application is provided in Listing 4 and the code used on the Raspberry Pi as the server is provided in Listing 7

The main aspect of this part of the project is setting up a socket that awaits connections on the Raspberry Pi at a specific port. This is completed using C network programming. In order for the Raspberry Pi to have access to wireless internet, we purchased a WiFi dongle, *Edimax EW-7811Un 150Mbps 11n Wi-Fi USB Adapter*. This dongle comes with a MAC, hardware address, and once that MAC address is registered with Princeton, the Raspberry Pi has access to the internet.

In order to connect to the socket on the Raspberry Pi, the Android Application runs a background process upon the user clicking the "Connect" button. This background process then attempts to connect to the IP address, which is hard coded in the application, and the port number, which is provided by the user of the application. Establishing a connection to a socket in java is very simple, because there is a dedicated **Socket** class.

Once the application is connected to the Raspberry Pi, which will be made known via the user interface of the application, the user must tell the Raspberry Pi where he is standing in the room. The user can either send a 0, which indicates the general open area of the ELE undergraduate lab, or aisle 1, 2, or 3, which are three of the lab aisles. The Raspberry Pi server will be awaiting this message, and upon receiving the message, the Pi will notify the PSOC which aisle the user is in, and then the PSOC will begin to search for Infrared LED's and begin traveling. The Raspberry Pi communicates the aisle number to the PSOC by sending 1 pulse if the message is 0, 2 pulses if the message is 1, and so on. The PSOC counts the number of pulses it sees in a 1 second time interval, stores the aisle number of the user, and begins the process the infrared data around it. A photo of our android application case, with a switch to turn on the attached emitter is provided in Figure 19. Furthermore, a picture of the side of our car, with the Raspberry Pi demonstrating the WiFi is on, with the blue light coming from the WiFi dongle is provided in Figure 20

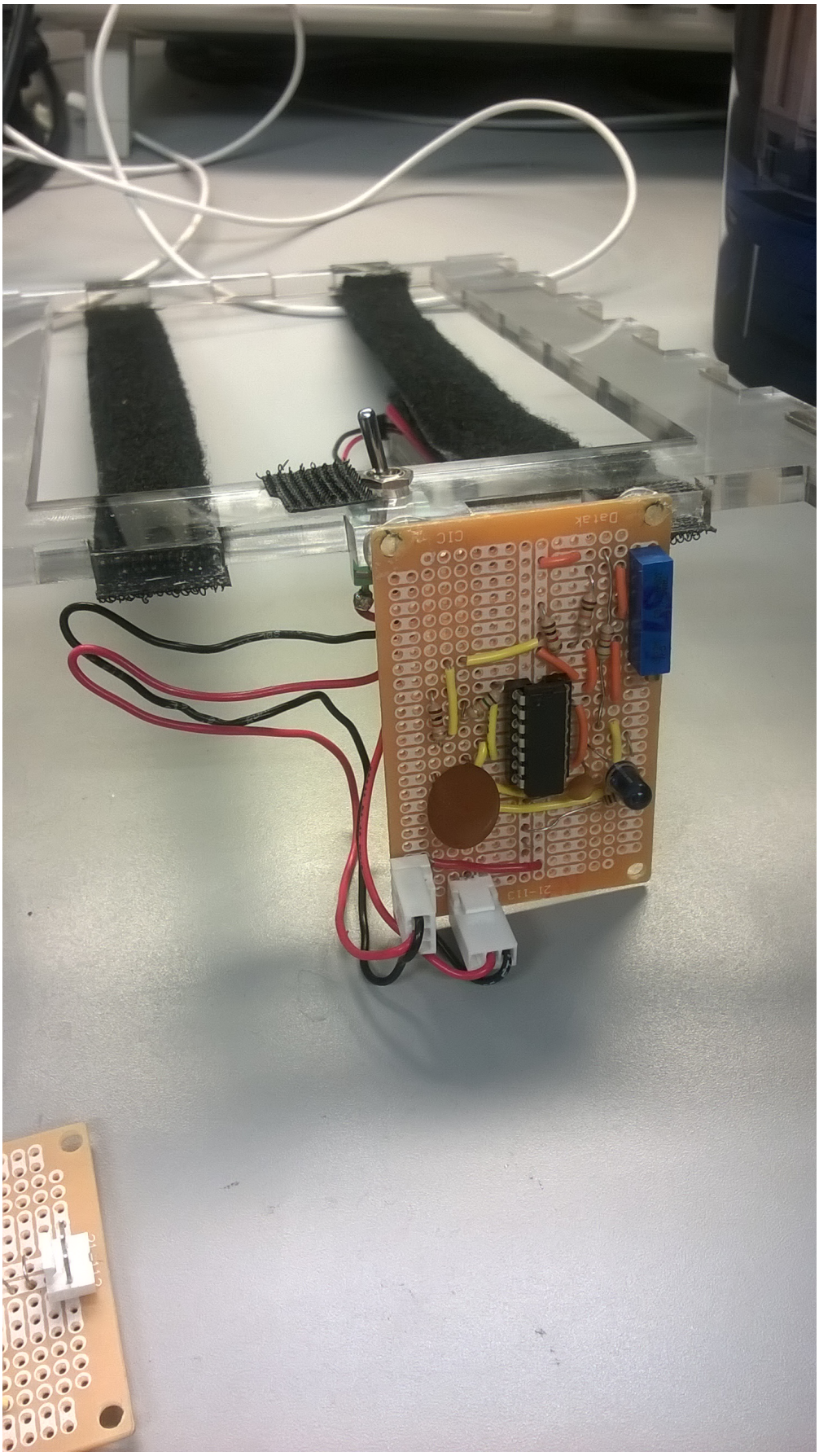

Figure 19: Android application case. With switch at the top to turn the IR emitter on and off.

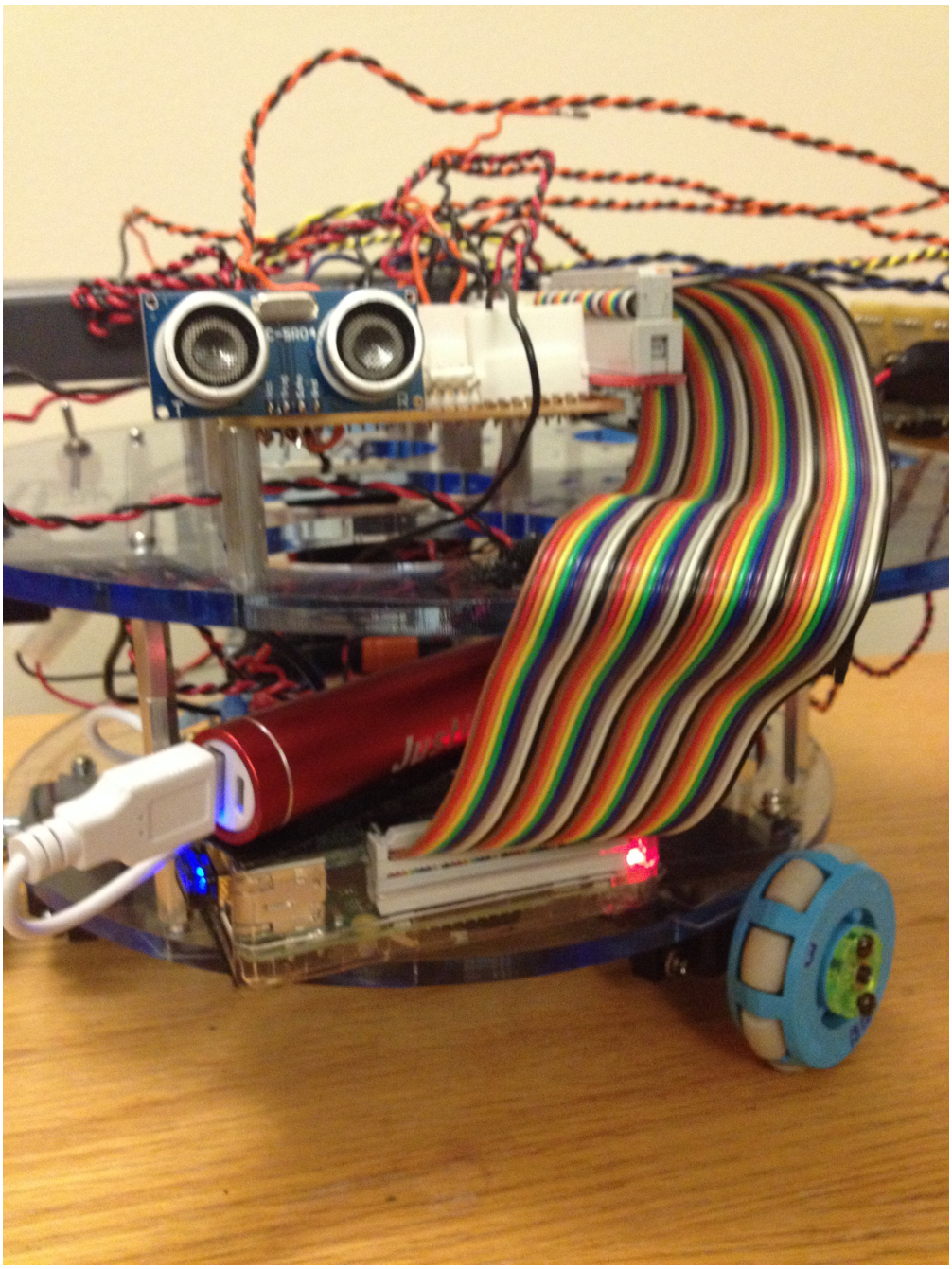

Figure 20: A side view of our car with the blue light indicating the WiFi is on.

7 Objective Five: Find the user

At this point in the project the car can drive towards an IR signal, back out of an aisle, and learn the user's general location via IP communication. Now, we put this all together to make the car find the user. There are many cases that we are interested in making work, and they will be provided here. There is one IR emitter at the end of aisle two. One emitter outside of the aisles, which is facing perpendicular to the aisles and located at the edge of aisle 1. The room setup is provided in Figure 21

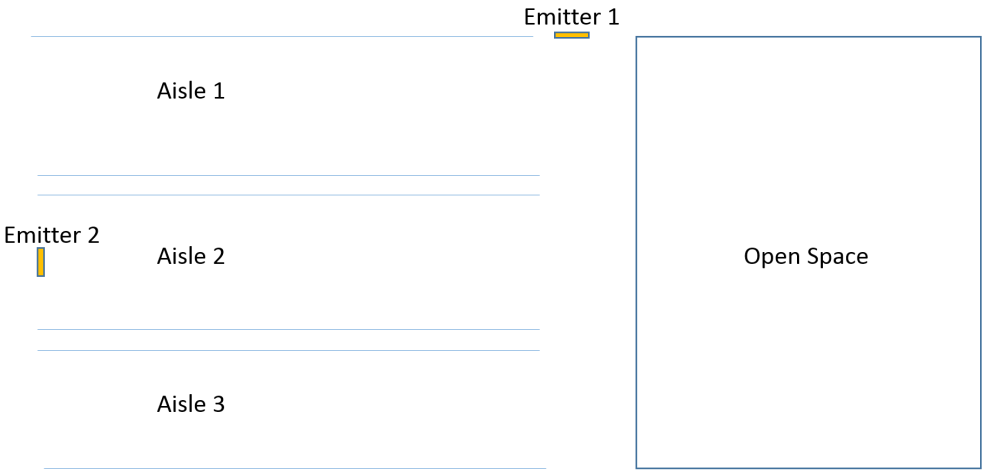

Figure 21: The room setup.

- **Car: Open Space, User: Open Space** - The car rotates until is finds the emitter attached to the android. The car will then control itself to the IR emitter that is attached to the android.
- **Car: Aisle 2, User: Open Space** - The car rotates until is finds Emitter 2. Then the car will exit Aisle 2, using the pingers to make sure it does not crash into any obstacles on it's sides. The pinger will notify the car when it has exited the aisle. At that point, the car will begin to search for the emitter connected to android. The car will find the emitter connected to the android, and drive to the user.
- **Car: Aisle 2, User: Aisle 1** - The car rotates until is finds Emitter 2. Then the car will exit Aisle 2, using the pingers to make sure it does not crash into any obstacles on it's sides. The car will then search for Emitter 1. Once it has found Emitter 1, it will drive towards Emitter 1 for 12 seconds. The 12 second interval is what we determined to be the time it takes for the car to cross the aisle. Once the car has crossed the aisle, it will begin to search for the emitter connected to the android. It will find the emitter attached to the android, and drive towards the user.
- **Car: Aisle 2, User: Aisle 2** - The car rotates until is finds the emitter connected to the android. It is probable that the car will find Emitter 2 during the process, but if it does find this emitter, it will ignore the signal and continue to rotate. Once the car find the emitter attached to the android, it will drive towards the user.
- **Car: Aisle 2, User: Aisle 3** - The car rotates until is finds Emitter 2. Then the car will exit Aisle 2, using the pings to make sure it does not crash into any obstacles on it's sides. Once the car is out of the aisle, it will search for Emitter 1. Once it has found Emitter 1, it will begin to back up towards Aisle 3. Once again, when the car is backing up, it uses the pingers on the side to make sure it does not crash into anything on its side. Once the car has backed up for 12 seconds, it will begin to look for the emitter connected to the android. Once it finds the emitter connected to the android, it will drive towards the user.

8 Conclusion and Future Work

Our final product worked successfully. On demo day we were able to demonstrate a few cases listed above. Namely we demonstrated the case when both the car and user are in open space. Furthermore, we demonstrated the car going from aisle 2 to open space, and from aisle 2 to aisle 3.

While our final product looks good, a lot of work still needs to be done in order to make the car truly come to the user anywhere in the lab. Here are a few things we must do in order to make our product more robust:

- **More Emitters** Currently our system uses three emitters. Therefore, the car can only begin in aisle 2, or in open space with the user. In order to make the car work all around the lab, there needs to be an emitter for each aisle. This will be difficult because we are constrained by AGC2 range of valid envelope frequencies; however, we are confident it is possible to create around 20 distinct emitters.
- **Obstacle Avoidance** Currently we avoid obstacles on the left and right when backing up. However, this must be extended to include obstacle avoidance on all sides. Furthermore, we must include obstacle avoidance when tracking signal in addition to backing up.

Our project was a success.

References

[1] *Voltage Regulator* https://blackboard.princeton.edu/courses/1/ELE302_S2015/content/_1795641_1/LM340.pdf

[2] *IR Receiver* http://www.mouser.com/ds/2/427/tsop322-542448.pdf

[3] *AGC* http://www.vishay.com/docs/49860/0811wd_d.pdf

[4] *Logic Level Converter* https://www.fairchildsemi.com/datasheets/2N/2N7000.pdf

[5] *LM556* http://www.learnabout-electronics.org/Downloads/lm556.pdf

[6] *Ultrasonic PING* http://www.jameco.com/Jameco/Products/ProdDS/282861.pdf

A Android Application Code

A.1 main.java

```
1  package com.example.carlabpro;
2
3
4  import java.io.ByteArrayOutputStream;
5  import java.io.IOException;
6  import java.io.InputStream;
7  import java.io.OutputStream;
8  import java.io.PrintStream;
9  import java.io.UnsupportedEncodingException;
10 import java.net.InetAddress;
11 import java.net.Socket;
12 import java.net.UnknownHostException;
13
14 import android.os.AsyncTask;
15 import android.os.Bundle;
16 import android.annotation.SuppressLint;
17 import android.app.Activity;
18 import android.view.View;
19 import android.view.View.OnClickListener;
20 import android.widget.Button;
21 import android.widget.EditText;
22 import android.widget.TextView;
23
24 public class MainActivity extends Activity {
25
26     TextView textResponse, intro, aisleExplanation;
27     EditText editTextPort, aisleTextPort;
28     Button buttonGo, buttonConnect, buttonClear, buttonDisconnect;
29     static Socket socket = null;
30
31     public static void send(String message) {
32         if (socket == null) {
33             return;
34         }
35         OutputStream outputStream = null;
36         try {
37             outputStream = socket.getOutputStream();
38         } catch (IOException e) {
39             // TODO Auto-generated catch block
40             e.printStackTrace();
41         }
42
43         PrintStream printStream = new PrintStream(outputStream);
44         printStream.print(message);
45     }
46
47     public static String recv() {
48         if (socket == null) {
49             return null;
50         }
51         ByteArrayOutputStream byteArrayOutputStream =
52         new ByteArrayOutputStream(1024);
53         byte[] buffer = new byte[1024];
54
55         int bytesRead = 0;
56         InputStream inputStream = null;
57         try {
58             inputStream = socket.getInputStream();
59         } catch (IOException e) {
60             // TODO Auto-generated catch block
61             e.printStackTrace();
62         }
63
```

```
64          /*
65          * notice:
66          * inputStream.read() will block if no data return
67          */
68          try {
69              bytesRead = inputStream.read(buffer);
70          } catch (IOException e) {
71              // TODO Auto-generated catch block
72              e.printStackTrace();
73          }
74          byteArrayOutputStream.write(buffer, 0, bytesRead);
75
76          try {
77              return byteArrayOutputStream.toString("UTF-8");
78          } catch (UnsupportedEncodingException e) {
79              // TODO Auto-generated catch block
80              e.printStackTrace();
81          }
82          return null;
83      }
84
85      @Override
86      protected void onCreate(Bundle savedInstanceState) {
87          super.onCreate(savedInstanceState);
88          setContentView(R.layout.activity_main);
89
90          editTextPort = (EditText)findViewById(R.id.port);
91          aisleTextPort = (EditText)findViewById(R.id.aisle);
92          buttonGo = (Button)findViewById(R.id.go);
93          buttonConnect = (Button)findViewById(R.id.connect);
94          buttonClear = (Button)findViewById(R.id.clear);
95          buttonDisconnect = (Button) findViewById(R.id.disconnect);
96          textResponse = (TextView)findViewById(R.id.response);
97          intro = (TextView)findViewById(R.id.intro);
98          intro.setText("Welcome to Car Lab Final Demo! Make sure the car is turned on (both
        power and wheels), and placed in the open track space, or the center of aisle 2. Enter
        the port number that I tell you to. Then hit connect!");
99          aisleExplanation = (TextView)findViewById(R.id.aisle_explain);
100         aisleExplanation.setText("This is your chance to tell the car where you are before
        it sets in motion. Enter the corresponding aisle (1-3) that you are in, if you're in an
        aisle. Enter a 0, if you're in the open track space. Then hit Car Come.");
101         buttonGo.setOnClickListener(buttonGoOnClickListener);
102
103         buttonConnect.setOnClickListener(buttonConnectOnClickListener);
104
105         buttonDisconnect.setOnClickListener(buttonDisconnectOnClickListener);
106
107         buttonClear.setOnClickListener(new OnClickListener(){
108
109             @Override
110             public void onClick(View v) {
111                 textResponse.setText("");
112             }
113
114         });
115     }
116
117
118     @SuppressLint("NewApi") OnClickListener buttonGoOnClickListener =
119     new OnClickListener(){
120         @Override
121         public void onClick(View arg0) {
122             if (socket == null) {
123                 textResponse.setText("Must connect first!");
124                 return;
125             }
126             if (aisleTextPort.getText().toString().isEmpty()) {
127                 textResponse.setText("Please enter an aisle number.");
```

```
128                 return;
129             }
130 
131             int aisle = Integer.parseInt(aisleTextPort.getText().toString());
132 
133             if (aisle >= 0 && aisle < 4) {
134                 ConnectionTask connectionTask = new ConnectionTask(aisle);
135                 connectionTask.execute();
136             }
137             else {
138                 textResponse.setText("Invalid Aisle Number");
139             }
140         }
141     };
142 
143     @SuppressLint("NewApi") OnClickListener buttonConnectOnClickListener =
144     new OnClickListener(){
145 
146         @Override
147         public void onClick(View arg0) {
148             if (editTextPort.getText().toString().isEmpty()) {
149                 textResponse.setText("Please enter a port number.");
150                 return;
151             }
152 
153             int port = Integer.parseInt(editTextPort.getText().toString());
154 
155             if (port > 40000 && port < 50000) {
156                 MyClientTask myClientTask = new MyClientTask(port);
157                 myClientTask.execute();
158             }
159             else {
160                 textResponse.setText("Invalid Port number");
161             }
162         }
163     };
164 
165     OnClickListener buttonDisconnectOnClickListener =
166     new OnClickListener(){
167 
168         @Override
169         public void onClick(View arg0) {
170             MyClientTask myClientTask = new MyClientTask(0);
171             myClientTask.execute();
172         }
173     };
174 
175 
176 
177     public class ConnectionTask extends AsyncTask<Void, Void, Void> {
178 
179 
180         String message;
181         int aisleValue;
182         String errMessage = null;
183         String response = null;
184 
185         ConnectionTask(int aisle){
186             aisleValue = aisle;
187         }
188             @Override
189             protected Void doInBackground(Void... arg0) {
190 
191                 if (aisleValue == 0) {
192                     message = "ONE\n";
193                 }
194                 if (aisleValue == 1) {
195                     message = "TWO\n";
```

```
                }
                if (aisleValue == 2) {
                    message = "THREE\n";
                }
                if (aisleValue == 3) {
                    message = "FOUR\n";
                }
                MainActivity.send(message);
                response = MainActivity.recv();

                return null;
            }

        @Override
        protected void onPostExecute(Void result) {
            if (errMessage == null && response != null) {
                textResponse.setText(response);
            }
            else
                textResponse.setText(errMessage);
            super.onPostExecute(result);
        }

    }

    public class MyClientTask extends AsyncTask<Void, Void, Void> {

        String dstAddress = "10.8.92.220";
        int dstPort;
        String errMessage = null;
        String message = null;

        MyClientTask(int port){
            dstPort = port;
        }

        @Override
        protected Void doInBackground(Void... arg0) {

            try {

                if (dstPort == 0 && socket != null) {
                    socket.close();
                    socket = null;
                    message = "Disconnected.\n";
                }
                else if (socket != null) {
                    message = "Already connected!\n";
                    return null;
                }
                else if(socket == null) {
                    InetAddress serverAddr = InetAddress.getByName(dstAddress);
                    socket = new Socket(serverAddr, dstPort);
                    if (socket != null) {
                        message = MainActivity.recv();
                    }
                }


            }
            catch (UnknownHostException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
                errMessage = "UnknownHostException: " + e.toString();
            }
            catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
```

```
                errMessage = "IOException: " + e.toString();
            }

            return null;
        }

        @Override
        protected void onPostExecute(Void result) {
            if (errMessage == null && message != null) {
                textResponse.setText(message);
            }
            else
                textResponse.setText(errMessage);

            super.onPostExecute(result);
        }
    }
}
```

Listing 4: Main java file used in Android Application.

A.2 layout.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:orientation="vertical"
    tools:context=".MainActivity" >
    <TextView
        android:id="@+id/intro"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText
        android:id="@+id/port"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="dstPort" />
    <TextView
        android:id="@+id/aisle_explain"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText
        android:id="@+id/aisle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Aisle Number" />
    <Button
        android:id="@+id/connect"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Connect..."/>
     <Button
        android:id="@+id/go"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Car Come..."/>
    <Button
        android:id="@+id/clear"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Clear"/>
    <TextView
        android:id="@+id/response"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

       <Button
        android:id="@+id/disconnect"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Disconnect"/>

</LinearLayout>
```

Listing 5: Main layout used in Android Application.

B PSOC Code

B.1 main.c

```
/* ========================================
 * Mike Freyberger and Rediet Desta
 * Final Car Lab Project
 * May 8, 2015
 * ========================================
*/
#include <device.h>
#include <stdio.h>

#define stop1 1515
#define stop2 1515
#define stop3 1505

struct dad {
    uint16 sampleCount;
    uint16 bitCount;
    uint16 valid;
    uint16 dadAisle;
};

static uint16 min_time_ellapsed = 0;
static uint16 reset_count = 0;
static uint16 free_search_count = 0;
static uint16 countTest = 0;
static uint16 searchForDad = 0;
static uint16 searchForLeft = 0;
static uint16 searchForRight = 0;

typedef struct dad dad;

dad ip_info = {0, 0, 0, 0};

static uint16 OnePercentOff = 10500;

static char strBuffer[8];

typedef enum {
    true,
    false
} bool;

typedef enum {
    INIT,
    SEARCHING,
    FOUND_ONE,
    IDENTIFIED_NONE,
    IDENTIFIED_ONE,
    IDENTIFIED_BOTH,
    CONTROL,
    EXIT_AISLE,
    CHANGE_AISLE,
    CORRECT_RIGHT,
    CORRECT_LEFT,
    LOST_BACK,
    TEMP_EXIT_STOP,
    FINISH_EXIT,
    FREE_SEARCH,
    DRIVE_AWAY
} stateMachine;

stateMachine state = INIT;

struct wheelSpeed {
```

```
    const uint16 wheel1;
    const uint16 wheel2;
    const uint16 wheel3;
};

typedef struct wheelSpeed wheelSpeed;

wheelSpeed stopped = {stop1, stop2, stop3};
wheelSpeed rotate = {stop1 - 40, stop2 - 40, stop3 - 40};
wheelSpeed straight = {stop1 + 190, stop2 - 112, stop3};
wheelSpeed backwards = {stop1 - 136, stop2 + 106, stop3};
wheelSpeed slightLeftWheel = {stop1 + 60, stop2 - 30, stop3};
wheelSpeed slightRightWheel = {stop1 + 30, stop2 - 60, stop3};
wheelSpeed slightRightWheelBack = {stop1 - 60, stop2 + 30, stop3};
wheelSpeed slightLeftWheelBack = {stop1 - 30, stop2 + 60, stop3};
wheelSpeed slowRightRotateWheel = {stop1 - 40, stop2 - 40, stop3 -40};
wheelSpeed slowLeftRotateWheel = {stop1 + 40, stop2 + 40, stop3 +40};
wheelSpeed driveLeftBackwardWheel = {stop1 - 160, stop2 + 80, stop3 + 40};

static void driveStraight() {
    PWM_1_WriteCompare(straight.wheel1);
    PWM_2_WriteCompare(straight.wheel2);
    PWM_3_WriteCompare(straight.wheel3);
}

static void slightLeft() {
    PWM_1_WriteCompare(slightLeftWheel.wheel1);
    PWM_2_WriteCompare(slightLeftWheel.wheel2);
    PWM_3_WriteCompare(slightLeftWheel.wheel3);
}

static void slightRight() {
    PWM_1_WriteCompare(slightRightWheel.wheel1);
    PWM_2_WriteCompare(slightRightWheel.wheel2);
    PWM_3_WriteCompare(slightRightWheel.wheel3);
}

static void slightLeftBackwards() {
    PWM_1_WriteCompare(slightLeftWheelBack.wheel1);
    PWM_2_WriteCompare(slightLeftWheelBack.wheel2);
    PWM_3_WriteCompare(slightLeftWheelBack.wheel3);
}

static void slightRightBackwards() {
    PWM_1_WriteCompare(slightRightWheelBack.wheel1);
    PWM_2_WriteCompare(slightRightWheelBack.wheel2);
    PWM_3_WriteCompare(slightRightWheelBack.wheel3);
}
static void slowRightRotate() {
    LCD_ClearDisplay();
    LCD_Position(0,0);
    LCD_PrintString("Slow right rotate.");
    PWM_1_WriteCompare(slowRightRotateWheel.wheel1);
    PWM_2_WriteCompare(slowRightRotateWheel.wheel2);
    PWM_3_WriteCompare(slowRightRotateWheel.wheel3);
}
static void slowLeftRotate() {
    LCD_ClearDisplay();
    LCD_Position(0,0);
    LCD_PrintString("Slow left rotate.");
    PWM_1_WriteCompare(slowLeftRotateWheel.wheel1);
    PWM_2_WriteCompare(slowLeftRotateWheel.wheel2);
    PWM_3_WriteCompare(slowLeftRotateWheel.wheel3);
}

static void stopCar() {
    PWM_1_WriteCompare(stopped.wheel1);
    PWM_2_WriteCompare(stopped.wheel2);
```

```
    PWM_3_WriteCompare(stopped.wheel3);
}

static void rotateCar() {
    PWM_1_WriteCompare(rotate.wheel1);
    PWM_2_WriteCompare(rotate.wheel2);
    PWM_3_WriteCompare(rotate.wheel3);
}

static void driveBackwards() {
    PWM_1_WriteCompare(backwards.wheel1);
    PWM_2_WriteCompare(backwards.wheel2);
    PWM_3_WriteCompare(backwards.wheel3);
}

static void driveLeftBackwards() {
    PWM_1_WriteCompare(driveLeftBackwardWheel.wheel1);
    PWM_2_WriteCompare(driveLeftBackwardWheel.wheel2);
    PWM_3_WriteCompare(driveLeftBackwardWheel.wheel3);
}

struct eye {
    uint8 name;
    bool found;
    uint16 ticks;
    uint16 count;
    bool identified;
    uint16 totalTicks;
    bool on;
    uint16 aisle;
};

typedef struct eye eye;

eye left = {1, false, 0, 0, false, 0, false, 0};
eye right =  {2, false, 0, 0, false, 0, false, 0};

static void setLCD(eye *curEye) {
    if (curEye->name == 1) {
        LCD_Position(0,0);
        LCD_PrintString("L: ");
    }
    else if (curEye->name == 2) {
        LCD_Position(1,0);
        LCD_PrintString("R: ");
    }
    else {
        LCD_ClearDisplay();
        LCD_Position(0,0);
        LCD_PrintString("Super big error.");
    }
}

static void controlCar() {
    LCD_ClearDisplay();
    LCD_Position(0,0);
    if (left.on == true && right.on == true) {
        LCD_PrintString("Both");
        driveStraight();
    }
    else if (left.on != true && right.on == true) {
        LCD_PrintString("RIGHT");
        slightLeft();
    }
    else if (left.on == true && right.on != true) {
        LCD_PrintString("LEFT");
        slightRight();
    }
```

```
200     else {
201         LCD_PrintString("Lost");
202         rotateCar();
203     }
204 }
205 
206 static void analyzeFreq(eye *curEye) {
207     bool validSignal = false;
208 
209     /* Analyze the frequency */
210     if (curEye->totalTicks > 600 && curEye->totalTicks < 820) {
211         curEye->aisle = 1;
212         validSignal = true;
213     }
214 
215     else if (curEye->totalTicks > 820 && curEye->totalTicks < 1400) {
216         curEye->aisle = 3;
217         validSignal = true;
218     }
219 
220 
221     else if (curEye->totalTicks > 400 && curEye->totalTicks < 600) {
222         curEye->aisle = 12;
223         validSignal = true;
224     }
225 
226     /* Check if the frequency was valid */
227     if (validSignal == true) {
228         curEye->on = true;
229         curEye->identified = true;
230         if (state == IDENTIFIED_NONE) {
231             state = IDENTIFIED_ONE;
232         }
233         else if (state == IDENTIFIED_ONE) {
234             state = IDENTIFIED_BOTH;
235         }
236     }
237 
238     /* If the frquency is invalid */
239     else {
240         curEye->on = false;
241         curEye->identified = false;
242         curEye->count = 0;
243         curEye->totalTicks = 0;
244         curEye->aisle = 0;
245     }
246 }
247 
248 
249 static void resetEye(eye *curEye) {
250     curEye->identified = false;
251     curEye->count = 0;
252     curEye->totalTicks = 0;
253     curEye->on = false;
254     curEye->found = false;
255     curEye->aisle = 0;
256     curEye->ticks = 0;
257 }
258 
259 static void resetIP(dad* curDad) {
260     curDad->bitCount = 0;
261     curDad->sampleCount = 0;
262     curDad->dadAisle = 0;
263     curDad->valid = 0;
264 }
265 
266 static void updateFSM(eye *curEye) {
267     if (state == INIT) {
```

```
268         stopCar();
269     }
270 
271     else if (state == SEARCHING) {
272         rotateCar();
273         if (curEye->ticks > 10) {
274             curEye->found = true;
275             state = FOUND_ONE;
276         }
277     }
278     else if (state == FOUND_ONE) {
279         LCD_ClearDisplay();
280         LCD_Position(0,0);
281         LCD_PrintString("FoundOne.");
282         if (curEye->found != true && curEye->ticks > 10) {
283             curEye->found = true;
284             state = IDENTIFIED_NONE;
285             stopCar();
286         }
287     }
288     else if (state == IDENTIFIED_NONE) {
289         setLCD(curEye);
290         if (curEye->ticks > 10) {
291             LCD_PrintString("Onski.");
292             curEye->totalTicks = curEye->totalTicks + curEye->ticks;
293             curEye->count = curEye->count + 1;
294             sprintf(strBuffer, "%u", curEye->count);
295             LCD_PrintString(strBuffer);
296             if (curEye->count == 10) {
297                 setLCD(curEye);
298                 LCD_PrintString("FREQ.");
299                 analyzeFreq(curEye);
300             }
301         }
302         else if (reset_count < 10) {
303             setLCD(curEye);
304             LCD_PrintString("Reset.");
305             reset_count++;
306             curEye->count = 0;
307             curEye->totalTicks = 0;
308         }
309         else {
310             setLCD(curEye);
311             LCD_PrintString("Reset.");
312             curEye->count = 0;
313             curEye->totalTicks = 0;
314             resetEye(&left);
315             resetEye(&right);
316             reset_count = 0;
317             state = SEARCHING;
318         }
319     }
320     else if (state == IDENTIFIED_ONE) {
321         setLCD(curEye);
322         if (curEye->identified != true && curEye->ticks > 10){
323             LCD_PrintString("Onbro.");
324             curEye->totalTicks = curEye->totalTicks + curEye->ticks;
325             curEye->count = curEye->count + 1;
326             sprintf(strBuffer, "%u", curEye->count);
327             LCD_PrintString(strBuffer);
328             if (curEye->count == 10) {
329                 analyzeFreq(curEye);
330             }
331         }
332         else if (reset_count < 10) {
333             setLCD(curEye);
334             LCD_PrintString("Reset.");
335             reset_count++;
```

```
            curEye->count = 0;
            curEye->totalTicks = 0;
        }
        else if (curEye->identified != true && curEye->ticks <= 10) {
            setLCD(curEye);
            LCD_PrintString("Reset.");
            curEye->count = 0;
            curEye->totalTicks = 0;
            resetEye(&left);
            resetEye(&right);
            reset_count = 0;
            state = SEARCHING;
        }
    }
    else if (state == IDENTIFIED_BOTH) {
        LCD_ClearDisplay();
        LCD_Position(0,0);
        LCD_PrintString("LEFT:");
        sprintf(strBuffer, "%u", left.totalTicks);
        LCD_PrintString(strBuffer);
        LCD_PrintString("--");
        sprintf(strBuffer, "%u", left.aisle);
        LCD_PrintString(strBuffer);
        LCD_Position(1,0);
        LCD_PrintString("RIGHT:");
        sprintf(strBuffer, "%u", right.totalTicks);
        LCD_PrintString(strBuffer);
        LCD_PrintString("--");
        sprintf(strBuffer, "%u", right.aisle);
        LCD_PrintString(strBuffer);

        if (left.aisle != right.aisle) {
            resetEye(&left);
            resetEye(&right);
            state = SEARCHING;
            LCD_ClearDisplay();
            LCD_Position(0,0);
            LCD_PrintString("Not equal.");
        }
        else {
            if (left.aisle == 1) {
                state = CONTROL;
                controlCar();
            }
            else if (searchForDad == 1) {
                resetEye(&left);
                resetEye(&right);
                LCD_ClearDisplay();
                LCD_Position(0,0);
                LCD_PrintString("Dad.");
                state = FREE_SEARCH;
            }
            else if (left.aisle != 12 && searchForLeft == 1) {
                resetEye(&left);
                resetEye(&right);
                 LCD_ClearDisplay();
                LCD_Position(0,0);
                LCD_PrintString("Left");
                state = FREE_SEARCH;
            }
            else if (left.aisle == 12 && searchForLeft == 1) {
                state = CONTROL;
                Cross_Aisle_Timer_Start();
                controlCar();
            }
            else if (left.aisle == 12 && searchForRight == 1) {
                driveBackwards();
                Cross_Aisle_Timer_Start();
```

```
404                 state = FINISH_EXIT;
405             }
406             else if (left.aisle == ip_info.dadAisle) {
407                 resetEye(&left);
408                 resetEye(&right);
409                  LCD_ClearDisplay();
410                 LCD_Position(0,0);
411                 LCD_PrintString("Same");
412                 state = FREE_SEARCH;
413             }
414             //The car sees an aisle beacon
415             else if (left.aisle == 3) {
416                 state = EXIT_AISLE;
417                 driveBackwards();
418                 Min_Exit_Timer_Start();
419             }
420             //The car sees a horizontal beacon.
421             else if (left.aisle == 12 || left.aisle == 14) {
422                 if (ip_info.dadAisle == 1) {
423                     resetEye(&left);
424                     resetEye(&right);
425                      LCD_ClearDisplay();
426                 LCD_Position(0,0);
427                 LCD_PrintString("Horizontal");
428                 state = FREE_SEARCH;
429                 }
430                 else {
431                     resetEye(&left);
432                     resetEye(&right);
433                     resetIP(&ip_info);
434                     searchForDad = 0;
435                     searchForLeft = 0;
436                     searchForRight = 0;
437                     stopCar();
438                     LCD_ClearDisplay();
439                     state = INIT;
440                 }
441             }
442         }
443     }
444
445     else if (state == FREE_SEARCH) {
446         rotateCar();
447         LCD_ClearDisplay();
448                 LCD_Position(0,0);
449                 LCD_PrintString("Free Search");
450         free_search_count++;
451         if (free_search_count > 10) {
452             state = SEARCHING;
453             free_search_count = 0;
454         }
455     }
456
457     else if (state == CONTROL) {
458         if (curEye->ticks > 10) {
459             curEye->on = true;
460         }
461         else {
462             curEye->on = false;
463         }
464         setLCD(curEye);
465         LCD_PrintString("Control");
466         controlCar();
467     }
468
469     else if (state == DRIVE_AWAY) {
470         driveLeftBackwards();
471         left.count++;
```

```
472         if (left.count > 22) {
473             left.count = 0;
474             state = CHANGE_AISLE;
475         }
476     }
477 
478     else if (state == EXIT_AISLE) {
479         /* Set to zero. Need counter var during backup */
480         left.count = 0;
481         right.count = 0;
482         LCD_PrintString("Exit");
483         driveBackwards();
484     }
485 
486     else if (state == FINISH_EXIT) {
487         LCD_ClearDisplay();
488         LCD_Position(0,0);
489         LCD_PrintString("Early exit");
490         driveBackwards();
491     }
492 
493     else if (state == CHANGE_AISLE) {
494         LCD_ClearDisplay();
495         LCD_Position(0,0);
496         LCD_PrintString("Time to rotate.");
497         if (ip_info.dadAisle == 1) {
498             searchForDad = 1;
499             resetEye(&left);
500             resetEye(&right);
501             state = SEARCHING;
502         }
503         else if(ip_info.dadAisle == 2) {
504             searchForLeft = 1;
505             resetEye(&left);
506             resetEye(&right);
507             state = SEARCHING;
508         }
509         else if (ip_info.dadAisle == 4) {
510             searchForRight = 1;
511             resetEye(&left);
512             resetEye(&right);
513             state = SEARCHING;
514         }
515     }
516     else if (state == CORRECT_RIGHT) {
517         right.count++;
518         if (right.count < 80) {
519             slightRightBackwards();
520         }
521         else if (right.count < 85) {
522             slowLeftRotate();
523         }
524         else if (right.count == 85) {
525             state = EXIT_AISLE;
526         }
527     }
528     else if (state == CORRECT_LEFT) {
529         if (left.count < 80) {
530             slightLeftBackwards();
531         }
532         else if (left.count < 90) {
533             slowRightRotate();
534         }
535         else if (left.count == 90) {
536             state = EXIT_AISLE;
537         }
538     }
539     else if (state == LOST_BACK) {
```

```
        rotateCar();
    }

    else if (state == TEMP_EXIT_STOP) {
        left.count++;
        if (left.count == 5) {
            state = EXIT_AISLE;
        }
    }

    curEye->ticks = 0;

}
//
//CY_ISR(stall_vector) {
//    if (state == IDENTIFIED_NONE || state == IDENTIFIED_ONE) {
//        state = SEARCHING;
//    }
//}

CY_ISR(cross_aisle_vector) {
    searchForDad = 1;
    searchForLeft = 0;
    searchForRight = 0;
    resetEye(&left);
    resetEye(&right);
    LCD_ClearDisplay();
    LCD_Position(0,0);
    LCD_PrintString("Cross Aisle Vector.");
    state = SEARCHING;
    Cross_Aisle_Timer_Stop();
}

CY_ISR(out_of_aisle_vector) {
    if (state == EXIT_AISLE && min_time_ellapsed == 1) {
        resetEye(&left);
        resetEye(&right);
        left.count = 0;
        state = DRIVE_AWAY;
    }
}

CY_ISR(min_exit_vector) {
    min_time_ellapsed = 1;
    Min_Exit_Timer_Stop();
}

CY_ISR(reset) {
    left.ticks = 0;
    right.ticks = 0;
}

CY_ISR(done_left) {
    updateFSM(&left);
    if (state == INIT) {
        if (ip_info.bitCount > 0 && ip_info.valid == 0) {
            ip_info.sampleCount++;
        }
        if (ip_info.sampleCount > 10 && ip_info.dadAisle == 0) {
            ip_info.valid = 1;
            ip_info.dadAisle = ip_info.bitCount;
            LCD_PrintString("JC.");
            state = SEARCHING;
        }
    }
}

CY_ISR(done_right) {
```

```
608     updateFSM(&right);
609 }
610 
611 CY_ISR(count_dad_vector) {
612 
613     if (state == INIT) {
614         if (ip_info.valid == 0) {
615             ip_info.bitCount++;
616             LCD_ClearDisplay();
617             LCD_Position(0,0);
618             LCD_PrintString("INIT.  ");
619             sprintf(strBuffer, "%u", ip_info.bitCount);
620             LCD_PrintString(strBuffer);
621         }
622     }
623 }
624 
625 CY_ISR(pulse_left)
626 {
627     Timer_Left_ClearFIFO();
628     left.ticks = left.ticks + 1;
629 }
630 
631 CY_ISR(pulse_right)
632 {
633     Timer_Right_ClearFIFO();
634     right.ticks = right.ticks + 1;
635 }
636 
637 CY_ISR(right_ping_vector) {
638     if (state == EXIT_AISLE || state == FINISH_EXIT) {
639         state = CORRECT_RIGHT;
640     }
641 }
642 
643 CY_ISR(left_ping_vector) {
644     if (state == EXIT_AISLE || state == FINISH_EXIT) {
645         state = CORRECT_LEFT;
646     }
647 }
648 
649 void main()
650 {
651     /* Place your initialization/startup code here (e.g. MyInst_Start()) */
652 
653     CyGlobalIntEnable;
654 
655     PWM_1_Start();
656     PWM_2_Start();
657     PWM_3_Start();
658 
659     rotateCar();
660 
661     VDAC_1_Start();
662     Comp_1_Start();
663     Timer_Left_Start();
664     Timer_Left_SetInterruptCount(0);
665     pulse_inter_left_Start();
666     pulse_inter_left_SetVector(pulse_left);
667     done_inter_left_Start();
668     done_inter_left_SetVector(done_left);
669 
670     VDAC_2_Start();
671     Comp_2_Start();
672     Timer_Right_Start();
673     Timer_Right_SetInterruptCount(0);
674     pulse_inter_right_Start();
675     pulse_inter_right_SetVector(pulse_right);
```

```
676     done_inter_right_Start();
677     done_inter_right_SetVector(done_right);
678 
679     LCD_Start();
680     LCD_Position(0,0);
681 
682     OnOffSwitch_Start();
683     OnOffSwitch_WriteCompare(OnePercentOff);
684 
685     reset_inter_Start();
686     reset_inter_SetVector(reset);
687 
688     VDAC_Dad_Start();
689     Comp_Dad_Start();
690     count_dad_Start();
691     count_dad_SetVector(count_dad_vector);
692 
693     cross_aisle_Start();
694     cross_aisle_SetVector(cross_aisle_vector);
695 
696     right_ping_Start();
697     right_ping_SetVector(right_ping_vector);
698     left_ping_Start();
699     left_ping_SetVector(left_ping_vector);
700 
701     out_of_aisle_Start();
702     out_of_aisle_SetVector(out_of_aisle_vector);
703 
704     min_exit_timer_inter_Start();
705     min_exit_timer_inter_SetVector(min_exit_vector);
706 
707 
708     for(;;)
709     {
710         /* Place your application code here. */
711     }
712 }
713 
714 /* [] END OF FILE */
```

Listing 6: Main C file used on PSOC.

C Rasberry Pi Code

C.1 server.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netdb.h>
#include <netinet/in.h>
#include <wiringPi.h>
#include <fcntl.h>
#include <softPwm.h>

#define MAXBUF 4096
/* #define BACKLOG 128*/
#define BACKLOG 0

#define AISLE 0

void respond(char* buf, int socket) {
    int bytesrecvd = strlen(buf);
    int bytessent, byteswritten;

    for (byteswritten = 0; byteswritten < bytesrecvd; byteswritten+=bytessent) {
      if ((bytessent = write(socket, buf+byteswritten, bytesrecvd-byteswritten)) < 0) {
        printf("Error writing response to client.");
        shutdown(socket, 2);
        close(socket);
        exit(EXIT_FAILURE);
      }
    }
}

void sendPulse(int count) {
  int i = 0;
  for (i  = 0; i < count; i++) {
    digitalWrite(AISLE, HIGH);
    delay(100);
    digitalWrite(AISLE, LOW);
    delay(100);
  }
}

/* Create, bind, and listen on a stream socket on port pcPort and return socket */
int createSocket(char *pcPort) {
  struct addrinfo aHints, *paRes;
  int iSockfd;

  /* Get address information for stream socket on input port */
  memset(&aHints, 0, sizeof(aHints));
  aHints.ai_family = AF_UNSPEC;
  aHints.ai_socktype = SOCK_STREAM;
  aHints.ai_flags = AI_PASSIVE;
  getaddrinfo(NULL, pcPort, &aHints, &paRes);

  /* Create, bind, listen */
  if ((iSockfd = socket(paRes->ai_family, paRes->ai_socktype, paRes->ai_protocol)) < 0) {
    perror("CREATE error");
    exit(EXIT_FAILURE);
  }
  if (bind(iSockfd, paRes->ai_addr, paRes->ai_addrlen) < 0) {
    perror("BIND error");
    exit(EXIT_FAILURE);
  }
  if (listen(iSockfd, BACKLOG) < 0) {
```

```
64      perror("LISTEN error");
65      exit(EXIT_FAILURE);
66    }
67
68    /* Free paRes, which was dynamically allocated by getaddrinfo */
69    freeaddrinfo(paRes);
70
71    return iSockfd;
72  }
73
74  /* Call: server <port-number> */
75  int main(int argc, char** argv) {
76    int iPort, iSockfd, iClientfd, iLen, iRecv, pwmGood;
77    char pcBuf[MAXBUF];
78    struct sockaddr aClient;
79    wiringPiSetup();
80    pinMode(AISLE, OUTPUT);
81    digitalWrite(AISLE, LOW);
82
83
84    /* Single argument of argv[1] is the port number */
85    if (argc != 2) {
86      fprintf(stderr, "Usage: %s <port-number>\n", argv[0]);
87      exit(EXIT_FAILURE);
88    }
89
90    /* Prepare to process requests */
91    iSockfd = createSocket(argv[1]);
92    iLen = sizeof(struct sockaddr);
93
94    /* Handle clients, one at a time */
95    while (1) {
96      fprintf(stdout, "Awaiting connection\n");
97      /* Accept the client, skipping on failure */
98      if ((iClientfd = accept(iSockfd, &aClient, &iLen)) <=  0) {
99        perror(argv[0]);
100       close(iClientfd);
101       continue;
102     }
103
104     fprintf(stdout,"I am connected to a client!\n");
105     respond("CONNECTED!\n", iClientfd);
106
107     /* Print and move along */
108     while ((iRecv = recv(iClientfd, pcBuf, MAXBUF, 0)) > 0) {
109       write(1, pcBuf, iRecv);
110       //parse the request
111       if (strcmp(pcBuf, "ONE\n") == 0) {
112         respond("COMING!\n", iClientfd);
113         sendPulse(1);
114       }
115       else if (strcmp(pcBuf, "TWO\n") == 0) {
116         respond("COMING!\n", iClientfd);
117         sendPulse(2);
118       }
119       else if (strcmp(pcBuf, "THREE\n") == 0) {
120         respond("COMING!\n", iClientfd);
121         sendPulse(3);
122       }
123       else if (strcmp(pcBuf, "FOUR\n") == 0) {
124         respond("COMING!\n", iClientfd);
125         sendPulse(4);
126       }
127       else {
128         respond("Unknown Command!\n", iClientfd);
129       }
130       memset(pcBuf,0,sizeof(pcBuf));
131     }
```

```
132
133     close(iClientfd);
134   }
135
136   /* Clean up */
137   close(iSockfd);
138
139   return 0;
140 }
```

Listing 7: Server set up on Raspberry Pi.

C.2 pingerCtrl.py

```
import time
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

#PINGER 1 Setup

#with this pinger, the echo and trigger
#are on the same pin, and the pin must be
#set to output and input

#echo and trigger
ping1_pin1 = 22

#-----------------------------------
#PINGER 2 Setup

#with this pinger the echo and trigger
#are on seperate pins

#trigger/output
ping2_pin1 = 20
GPIO.setup(ping2_pin1, GPIO.OUT)

#echo/input
ping2_pin2 = 16
GPIO.setup(ping2_pin2, GPIO.IN)

#-----------------------------------

#RasPI Interrupt Line for right side too close
pinRight = 18

#RasPI Interrupt Line for left side too close
pinLeft = 23

#RasPI Interrupt Line for out of aisle
pinAisle = 6

#Set up interrupt lines to be output
GPIO.setup(pinRight, GPIO.OUT)
GPIO.setup(pinLeft, GPIO.OUT)
GPIO.setup(pinAisle, GPIO.OUT)

#Intialize interrupt lines low
GPIO.output(pinRight, 0)
GPIO.output(pinLeft, 0)
GPIO.output(pinAisle, 0)

#Forever loop
while 1:

  #PINGER 1 -- RIGHT SIDE
  #Make pinger one pin output to start
  GPIO.setup(ping1_pin1, GPIO.OUT)

  #Output a high signal for .1 seconds
  GPIO.output(ping1_pin1, 1)
  time.sleep(0.1)
  GPIO.output(ping1_pin1,0)

  #Change pinger one pin to input
  GPIO.setup(ping1_pin1, GPIO.IN)

  #measure the start and end time of the echo
  while GPIO.input(ping1_pin1)==0:
```

```
    starttime=time.time()
  while GPIO.input(ping1_pin1)==1:
    endtime=time.time()

  #convert timing measurment to distance in cm
  duration=endtime-starttime
  distance=duration*34000/2
  print ("1: %f" % distance)

  #If the distance is less than 35 cm, send an interrupt
  #notifying the car that the right side is too close to an object
  if (distance < 35):
    GPIO.output(pinRight, 1)
    time.sleep(0.1)
    GPIO.output(pinRight, 0)

  #If the distance is greater than 3000 cm, send an interrupt
  #notifying the car that it is out of the aisle
  if (distance > 3000):
    GPIO.output(pinAisle,1)
    time.sleep(0.1)
    GPIO.output(pinAisle, 0)

  # ---------------------------------------------------------
  #PINGER 2 -- LEFT SIDE
  #Output a high signal for .1 seconds
  GPIO.output(ping2_pin1, 1)
  time.sleep(0.1)
  GPIO.output(ping2_pin1,0)

  #measure the start and end time of the echo
  while GPIO.input(ping2_pin2)==0:
    starttime=time.time()
  while GPIO.input(ping2_pin2)==1:
    endtime=time.time()

  #convert timing measurment to distance in cm
  duration=endtime-starttime
  distance=duration*34000/2
  print ("2: %f" % distance)

  #If the distance is less than 35 cm, send an interrupt
  #notifying the car that the left side is too close to an object
  if (distance < 35):
    GPIO.output(pinLeft, 1)
    time.sleep(0.1)
    GPIO.output(pinLeft, 0)

  #If the distance is greater than 3000 cm, send an interrupt
  #notifying the car that it is out of the aisle
  if (distance > 3000):
    GPIO.output(pinAisle,1)
    time.sleep(0.1)
    GPIO.output(pinAisle, 0)
```

Listing 8: Python code interacting with the ultrasonic pingers.