Wireless Control Robotic Car

Matthew Reilly, Julia Roscher

Final

12/5/2016

CEN 4930 Cyber-Physical Systems

Instructor: Dr. Janusz Zalewski

Department of Software Engineering

Florida Gulf Coast University

Ft. Myers, FL 33965

Copyright © 2016 by Florida Gulf Coast University

1. Introduction

The objective of this project, is to be able to control a Remote Controlled (RC) Robotic Car with an Arduino board through Bluetooth connectivity. The robot will be controlled by an Android application which will enable the user to control the forward, reverse, left, and right motion of the car. The car will automatically stop if presented with a command that will run the robot into a large object.

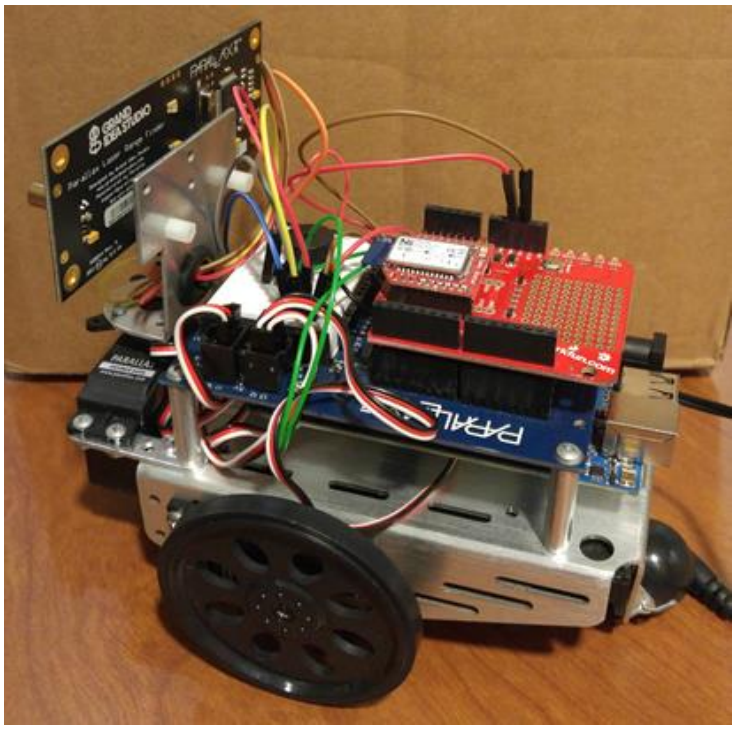


Figure 1. Picture of Arduino Car

The Arduino Robotic Car is of importance to the profession as it is a good way to show software developers a physical side to their code. Usually developers never get to see the application of their code in the physical world. By being able to see something in the physical world move and complete instructions written by themselves, it is a good way to let the programmer know the versatility of environments that their code can implemented in. Also, robotics are becoming more prevalent in everyday life. It is important to understand the applications within which robotics can be applied.

The technology being used in this project is an Arduino Mega ADK Board, an Android Phone, a RN42-XV Bluetooth Module connected to an SparkFun XBee Shield, the Parallax Laser Range Finder, a VCNL4000 proximity sensor, Parallax Board of Education Shield, and an LED light. The Arduino Board will be the main communicator between the car and the Android App User. The Arduino Board will communicate with the Android Application through Bluetooth. The Arduino Board will control the output of power to the motors and control which motors are on and which direction the motor will spin. The language being used to program the Arduino Board, will be in C/C++. The Parallax Laser Range Finder, will be able to measure the distance between the car and any objects the laser is pointing at. The VCNL4000 Proximity sensor, will be able to measure the amount of ambient light as well as any objects in front of the Robot up to 20cm.

Figure 2. Example of Android Phone

With the help of past projects [1], the car being used, has already been assembled. The car was previously wired with a Parallax Breadboard and Servo, batteries, two Servo motors controlling two wheels, Parallax Laser Range Finder and an Arduino Mega Board. The project will be focused on maintenance and the addition of the following features: a light with darkness sensors mounted on the Robot, password protected Android app, and accurate readings from proximity sensors to stop the car when an obstruction is in its path.

2. Software Requirements Specification

The objective of this project is to enhance upon the previously constructed Arduino Robotic Car by increasing the functionality of the Robot's proximity sensors and adding darkness sensitive light, and password protection to the Android App.

2.1 Introduction to Requirements

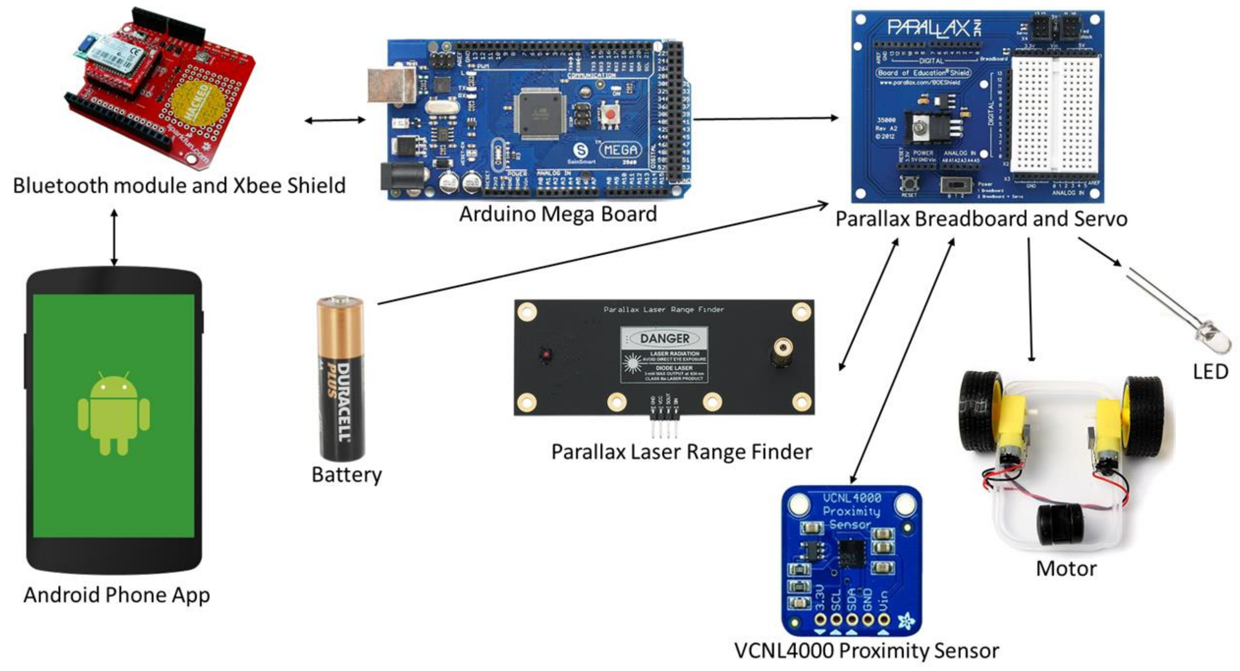

Bluetooth module and Xbee Shield
Arduino Mega Board
Parallax Breadboard and Servo
Android Phone App
Battery
Parallax Laser Range Finder
LED
Motor
VCNL4000 Proximity Sensor

Figure 3. Physical Diagram

The physical diagram (Figure 3) is a map of how the components of the project are connected to each other. On the left, the Android Phone App is connected to the Bluetooth Module located on the top of the robot. This sends commands from the user the Arduino Mega Board on the Robot. The Arduino will talk back to the phone with the battery status and robot’s status. All the components of the robot that interact with the physical environment will be connected through the Parallax Breadboard and Servo. These components will either be sending information back to the Parallax board or be receiving commands from the Parallax board that are given to it by the Arduino from Android.

2.2 Requirements Specification

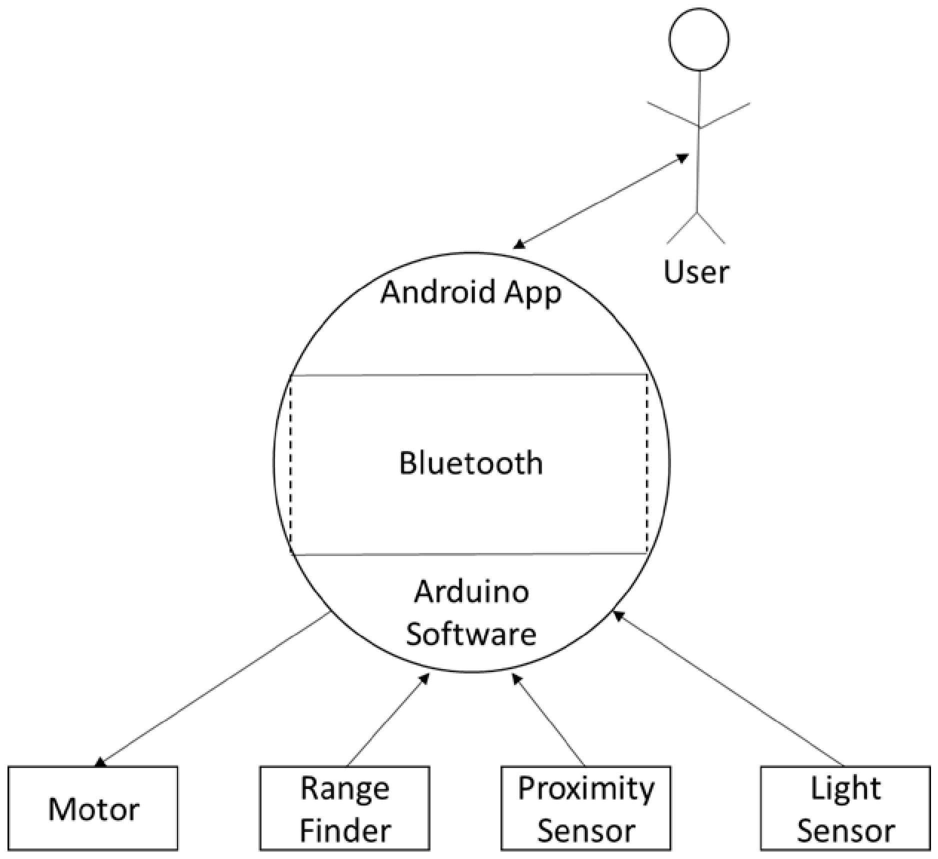


Figure 4. Context Diagram

The Context Diagram (Figure 4), shows what software will be developed and how that software connects with components in the physical world. In the figure, the Arduino Software and the Android App are the two software programs needed to make the robot function as intended. The Android App will communicate with the User. The App will communicate via Bluetooth with the Arduino Software, which will be the communicator between the physical sensors and motors of the Arduino Robot.

Based on the functional description presented above and the Context Diagram in Figure 4, specific requirements can be formulated as follows.

2.2.1 Arduino Software Requirements

1. The Arduino Software shall be able to respond to commands from an Android App. (Roscher)
2. The Arduino Software shall be able to sense when large objects are obstructing robot's pathway. (Reilly)
3. The Arduino Software shall be able move the robot left, right, forward, and backwards upon commands from the Android App. (Reilly)
4. The Arduino Software shall stop the robot before it hits objects obstructing its pathway. (Reilly)
5. The Arduino Software shall turn on LED when placed in darkness. (Reilly)

2.2.2 Android App Requirements

1. The Android Software shall be able to connect with local Bluetooth access points. (Roscher)
2. The Android Software shall enable the User to control the Arduino Robots direction and speed. (Reilly)
3. The Android Software shall have a clean UI to enhance user experience. (Reilly)
4. The Android Software shall connect to authorized users only through email authentication. (Roscher)

2.2.3 Design Constraints

1. The Arduino Robot must have functioning motor capabilities. (Roscher)
2. The Arduino Robot shall have Bluetooth connectivity. (Roscher)
3. The Arduino Robot must have charged batteries. (Reilly)
4. The Arduino Robot shall have Parallax Laser Range Finder. (Roscher)
5. The Arduino Robot shall have VCNL4000 Proximity Sensor. (Reilly)
6. The Android Software shall be able to run on Android version 4.4 and newer. (Roscher)
7. The Arduino Robot shall have a Bluetooth board properly attached to the Arduino board. (Roscher)
8. The Arduino Robot must have correct wiring between the breadboard and controllers. (Reilly)

3. Design Description

The design of this software involves the Arduino Software and Android App. The Arduino Software runs on the Arduino robot itself. The Software communicates with the Android Application via Bluetooth. The Android Application is the main communicator between the Arduino robot and the User. The Android Application sends commands to the Arduino Software which then takes those commands and sends them to the respective sensors and motors on the Arduino robot. The design description covers how the Arduino Software communicates with the physical and other aspects of the Arduino robot Program.

3.1 Software Architecture

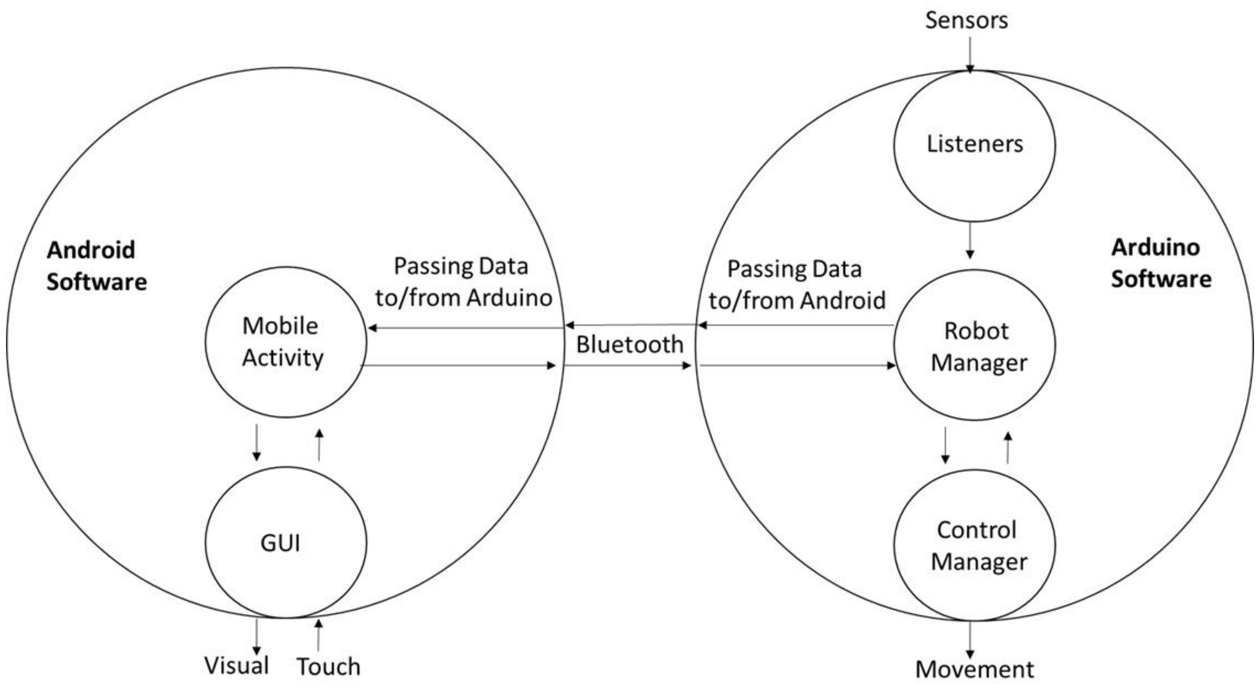


Figure 5: Software Architecture Diagram

The Android and Arduino Software Architecture Diagram (Figure 5) shows an overview of the relationships of the software components. The main components of the Android Software include Mobile Activity, and GUI. While the main components of the Arduino Software include the Listeners, Robot Manager, and Control Manager.

Users will connect with the Arduino via the download application to their Android device with Bluetooth enabled. The GUI is a visual display which has touch feedback enabled. The GUI will send commands entered by the User to the Mobile Activity. The Mobile Activity in the Android Software connects to the Robot Manager in the Arduino Software via Bluetooth. The Mobile Activity sends user commands to the Robot Manager. The Robot Manager then sends messages back to the Android Software whenever the commands being received contradict the reports from the sensors. This message will be displayed on the GUI. The Arduino Software takes in feedback from the sensors through way of the Listeners. The Listener sends the data to the Robot Manager which sends the appropriate commands to the Control Manager. The Control Manager then sends out the movements to be executed by the motors.

3.2 Detailed Design

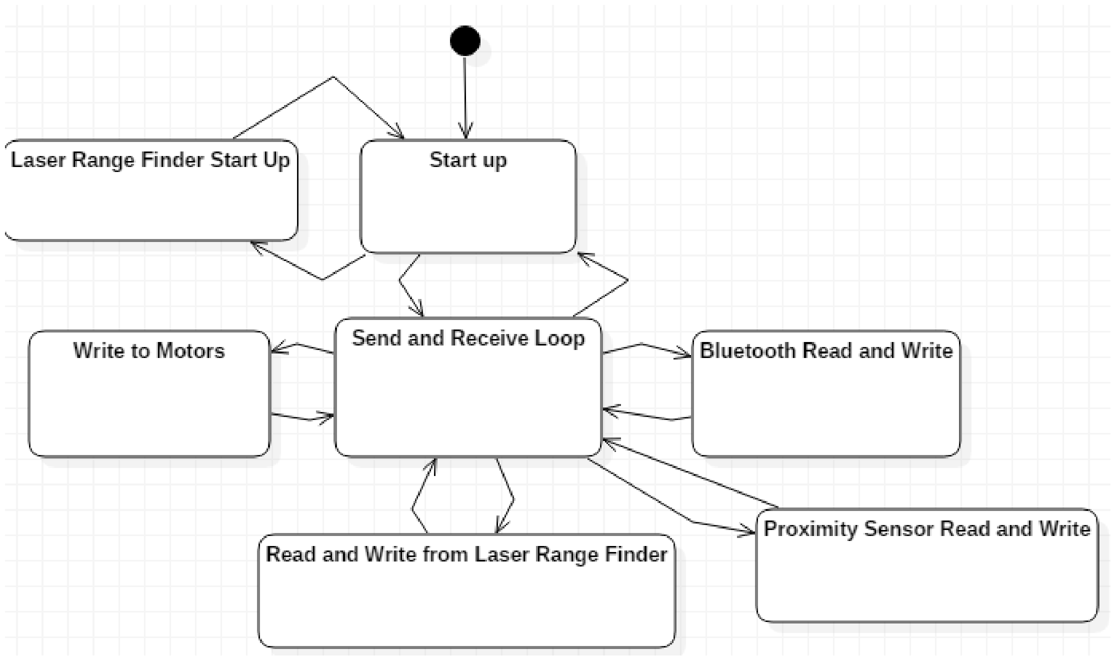

Laser Range Finder Start Up
Start up
Write to Motors
Send and Receive Loop
Bluetooth Read and Write
Proximity Sensor Read and Write
Read and Write from Laser Range Finder

Figure 6: State Diagram of the Robot Manager in Arduino Software

The State Diagram (Figure 6) visualizes the states of the Arduino Software and how they are connected. The Arduino Software starts out in an initial state which then points to the initial startup state. While the Arduino Software is starting up, the Laser Range Finder is also starting up. The Software then goes into a loop where Software is interacting with the outside components and constantly reading and writing to those components. From this loop the Software sends commands and receives alerts from the motors and the sensors on-board the Arduino Robot.

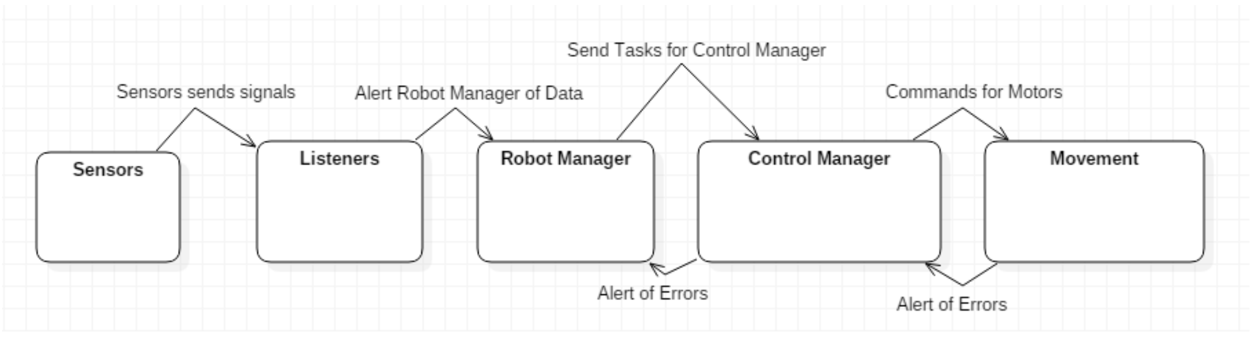


Figure 7: Dataflow Diagram for Arduino Software

The Data Flow Diagram (Figure 7) details how each component of the Arduino Software behaves with other components and modules. In this diagram, the sensors communicate with the Arduino Software through the Listeners in the Software. When the Listeners get the information from the Sensors, it alerts the Robot Manager of the presence of data from the Sensors. With the data from the Sensors, the Robot Manager will go ahead and run some calculations in order to tell the Robot where to move next. The Robot Manager then sends tasks for the Control Manager. The Control Manager is basically the main controller of the motors, and sends commands directly to the motors in order to manipulate them. The Robot Manager is in charge of sending back reports to the Android Software which will alert the User via Bluetooth.

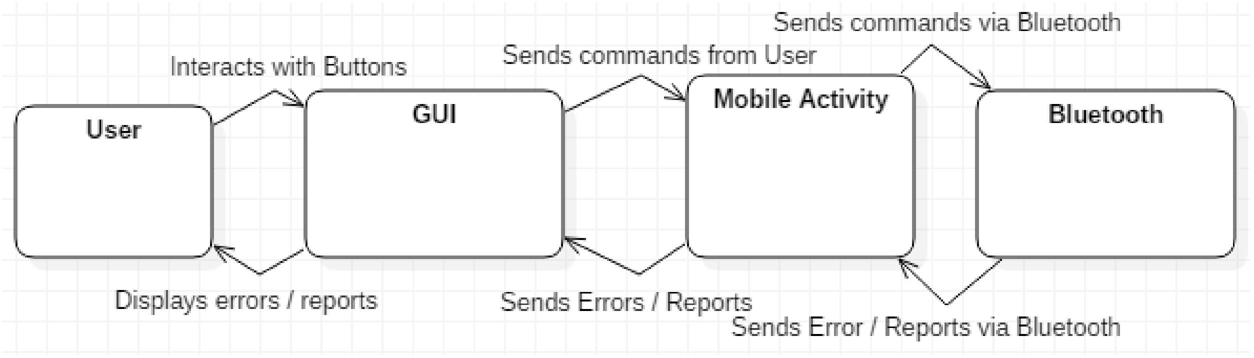


Figure 8: Dynamic State Diagram for Android Software

The Dynamic State Diagram for the Android Software (Figure 8) shows how to components of the Android Software all work and communicate with each other. The User interacts with the

GUI by pressing buttons on their screen. The GUI displays messages such as errors and reports in order for the User to see information. The GUI interacts with the Mobile Activity component which takes the commands from the GUI and puts them it into a form that can be received by Arduino Software. The Mobile Activity receives error reports and completion reports from the Bluetooth, and sends commands via Bluetooth to the Arduino Robot.

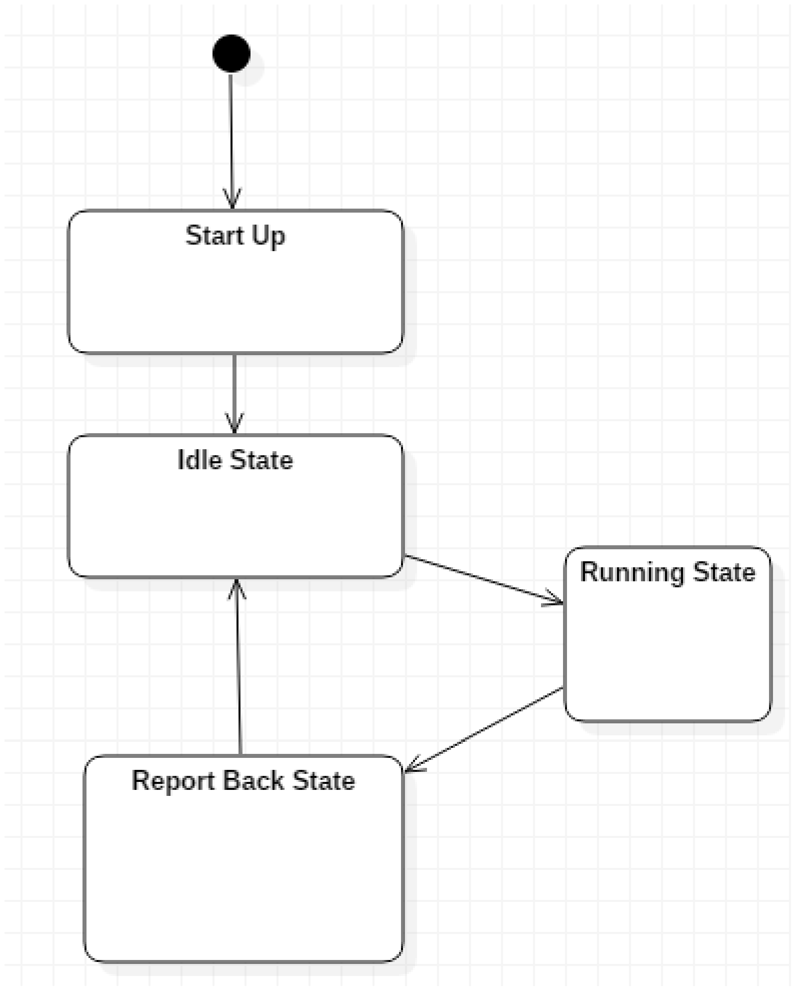


Figure 9: State Diagram for the Android Software

The State Diagram for the Android Software (Figure 9) goes over each state that the Android Software goes through while operating. The Android Software initially enters the Start Up State where the Arduino Board and laser range finder turn on. Then the Android Software enters into the Idle State, where the Android Software is waiting for the User to enter in commands to the GUI. Once the User has entered a command the software goes into the Running State as it is sending the user commands via Bluetooth to Arduino. These commands are translated into movements which the robot executes. The Running State continues until the Robot's proximity sensors detect something obstructing its path. The Android Software then enters the Report Back State where a message will be sent back to the User indicating that an obstruction has been identified. The Android Software will then return to the Idle State, where it will wait for the User’s next input.

4. Implementation and Testing

4.1 Assembly and Coding

Most of the physical components of the car have been constructed by a past project [1]. However, certain modifications have had to be made to the car to enable communication to be accurately transmitted between the various devices on the car and the Android app.

To begin with, the SparkFun XBee Shield, used to hold the RN42-XV Bluetooth Module, had to be replaced because it had been damaged since the last project. Also, the sensors had to be rewired to the Parallax Board, Breadboard, and Arduino so that they were properly acquiring power and sending/retrieving data. The extensive rewiring caused much of the past project's code to become obsolete and had to be rewritten.

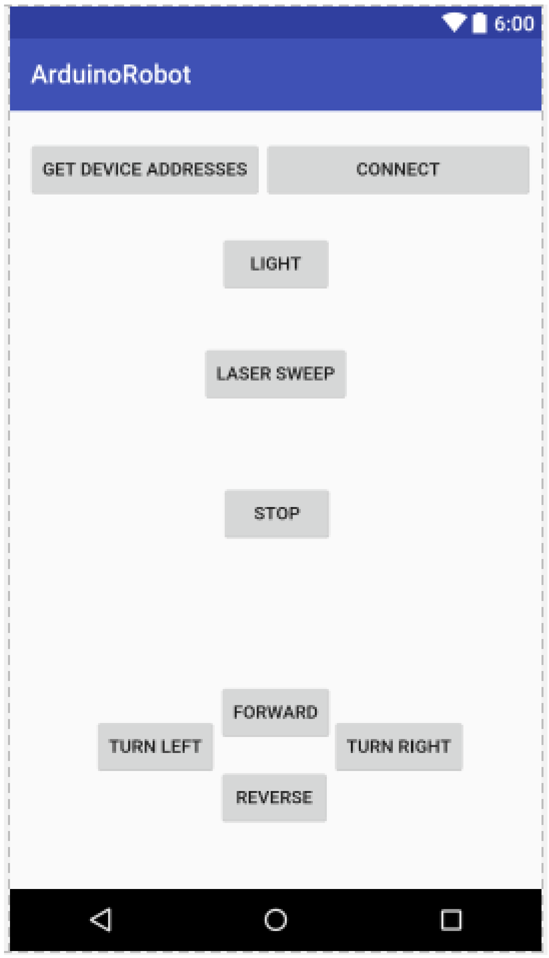


Figure 10: Android Application GUI

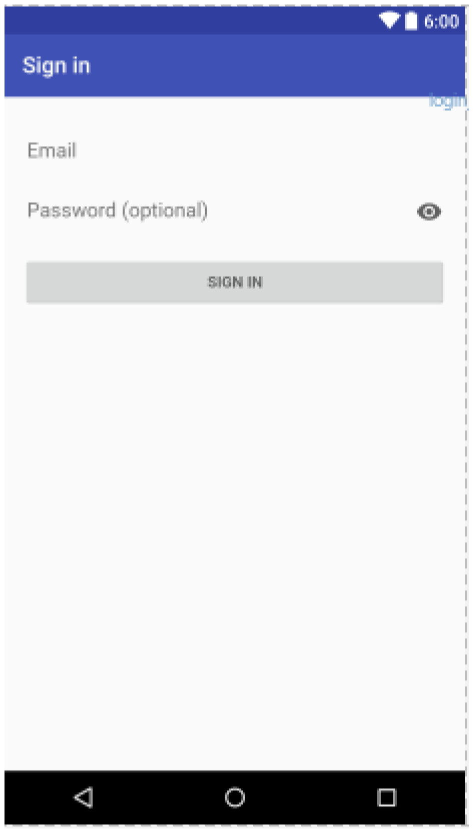
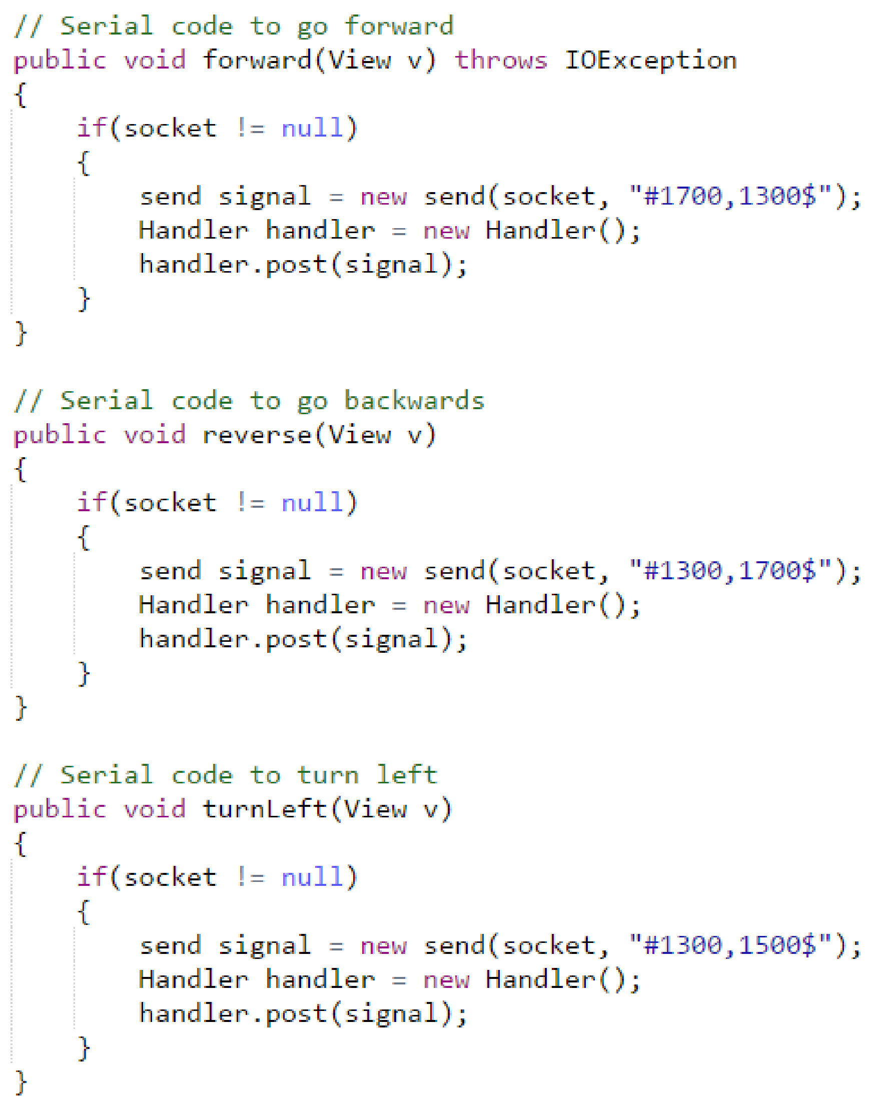


Figure 11: Android Application Login Page

The GUI Module is the interface that the User uses to interact directly with the Arduino Car (Figure 10 and Figure 11). Upon first opening the app, the user will be asked to authenticate themselves by entering an authorized email and password combination and selecting “login”. For the purpose of this project, a guest account was created of email: “Hello@gmail.com” and password: “password”. These credentials are case sensitive. Upon login, the user must select the “Connect” button. This button is to establish Bluetooth connection between the Android and the Arduino. Once connection has been established a message will be displayed on the app indicating such. Likewise, a message will be displayed if the connection between the devices was unsuccessful. The App is set-up so upon pressing a displayed button a corresponding message will be sent to the Mobile Activity Module which will send the message across the Bluetooth to the Arduino’s Robot Manager.

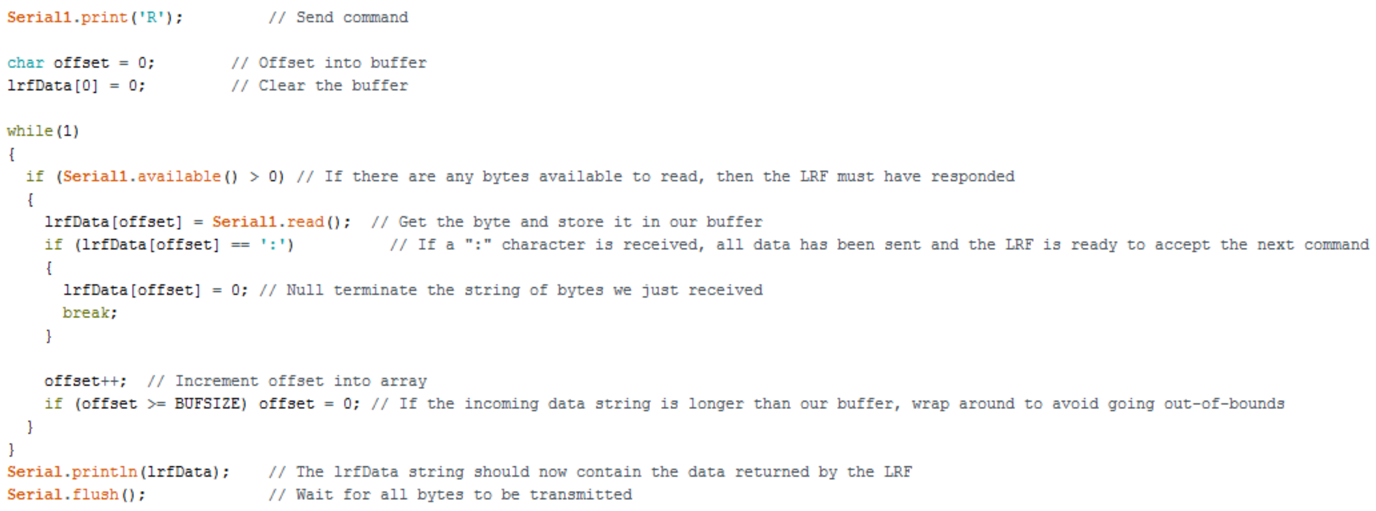


```
// Serial code to go forward
public void forward(View v) throws IOException
{
    if(socket != null)
    {
        send signal = new send(socket, "#1700,1300$");
        Handler handler = new Handler();
        handler.post(signal);
    }
}

// Serial code to go backwards
public void reverse(View v)
{
    if(socket != null)
    {
        send signal = new send(socket, "#1300,1700$");
        Handler handler = new Handler();
        handler.post(signal);
    }
}

// Serial code to turn left
public void turnLeft(View v)
{
    if(socket != null)
    {
        send signal = new send(socket, "#1300,1500$");
        Handler handler = new Handler();
        handler.post(signal);
    }
}
```

Figure 12: Android App Functionality Code Snippet

This interaction between the Android application and the Arduino Robot is shown in Figure 12. For example, when “Forward” is selected, the Arduino motor servos are sent the corresponding values to move the motors forward, 1700 ms and 1300 ms. Similar interactions happen when the other movement buttons are selected. It is still possible to manually tell the car to perform a sweep if one believes there's something in the Robot Car’s path that it does not see by selecting “Laser Sweep”. Also, “Get Device Addresses” works by displaying all the devices that one has connected to on their phone as an aid to help locate the Arduino you wish to connect to. Lastly, the “Light” button turns on/off the light attached to the Breadboard on top of the car. This light is programed to automatically turn on in low light levels and off in high light levels but can still be controlled manually if desired.

The Listener Module, operates with the Parallax Laser Range Finder (LRF) and the VCNL4000 Proximity Sensor (PS). Both sensors are receiving 5V of power and are attached to Ground through the Breadboard. The Parallax Laser Range Finder is also connected to a servo through Pin 11. The servo powers the motor that the LRF is mounted upon, the motor allows the LRF to swivel 180◦ in order to get a proximity reading on the car's surroundings.

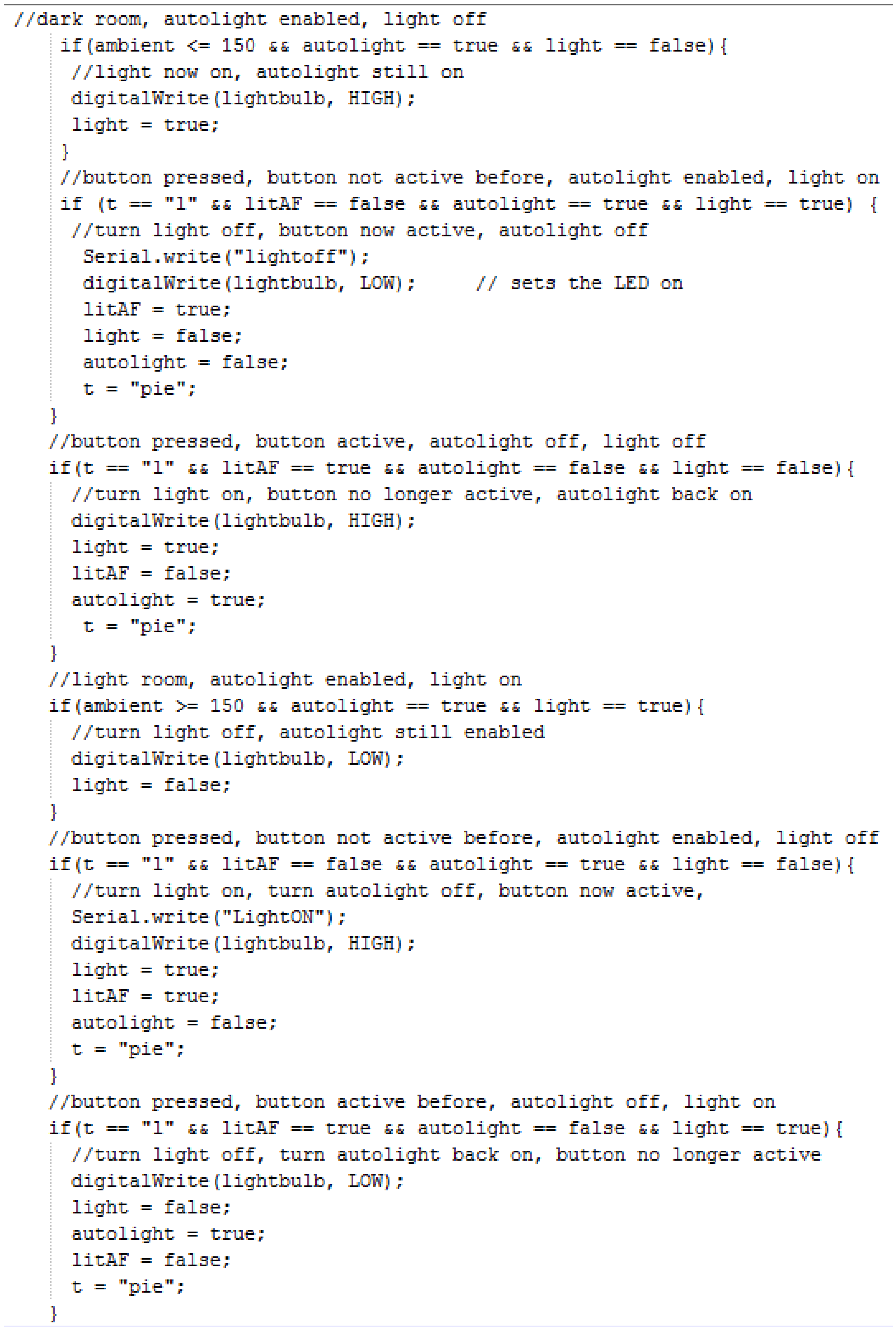


```
Serial1.print('R');         // Send command

char offset = 0;        // Offset into buffer
lrfData[0] = 0;         // Clear the buffer

while(1)
{
  if (Serial1.available() > 0) // If there are any bytes available to read, then the LRF must have responded
  {
    lrfData[offset] = Serial1.read();  // Get the byte and store it in our buffer
    if (lrfData[offset] == ':')          // If a ":" character is received, all data has been sent and the LRF is ready to accept the next command
    {
      lrfData[offset] = 0; // Null terminate the string of bytes we just received
      break;
    }

    offset++;  // Increment offset into array
    if (offset >= BUFSIZE) offset = 0; // If the incoming data string is longer than our buffer, wrap around to avoid going out-of-bounds
  }
}
Serial.println(lrfData);    // The lrfData string should now contain the data returned by the LRF
Serial.flush();             // Wait for all bytes to be transmitted
```

Figure 13: Parallax Laser Range Finder Code Snippet

Since the Arduino board is a Mega ADK it has additional serial ports that can be used besides the traditional “Serial” serial port. Serial1 on Pins 19 and 18 is used to enable communication between the LRF the Robot Manager module on the Arduino Board. This is shown in Figure 13, where ‘R’ is printed to Serial1. ‘R’ is basic command that is programed into the pieces firmware. ‘R’ is a command to get a “single range measurement (returns a 4-digit ASCII value in millimeters)” [2]. The returned value, stored in “lrfData”, is in the form D = 0123 mm. The LRF takes one reading of the Car’s surroundings to determine if objects are obstructing its path. If an object is found to be closer than 430 mm, or approximately 1.5 feet, than the motors of the car will stop and the car will wait for further instructions. In order to tell the car to take a LRF reading, the user must select “Laser Finder” on the Android app.

Similarly, Serial on Pins 20 and 21 are used to communicate between the VCNL4000 Proximity Sensor and the Robot Manager. Pin 21 is used for SCL communication and Pin 20 is used for SDA Communication on the Arduino Mega ADK. These pins are connected to their corresponding line on the Proximity Sensor. The SCL is a clock line, used to synchronize all data transfers; while the SDA is a data line. [3]



```
//dark room, autolight enabled, light off
    if(ambient <= 150 && autolight == true && light == false){
     //light now on, autolight still on
     digitalWrite(lightbulb, HIGH);
     light = true;
    }
    //button pressed, button not active before, autolight enabled, light on
    if (t == "1" && litAF == false && autolight == true && light == true) {
     //turn light off, button now active, autolight off
      Serial.write("lightoff");
      digitalWrite(lightbulb, LOW);     // sets the LED on
      litAF = true;
      light = false;
      autolight = false;
      t = "pie";
   }
   //button pressed, button active, autolight off, light off
   if(t == "1" && litAF == true && autolight == false && light == false){
     //turn light on, button no longer active, autolight back on
     digitalWrite(lightbulb, HIGH);
     light = true;
     litAF = false;
     autolight = true;
      t = "pie";
   }
   //light room, autolight enabled, light on
   if(ambient >= 150 && autolight == true && light == true){
     //turn light off, autolight still enabled
     digitalWrite(lightbulb, LOW);
     light = false;
   }
   //button pressed, button not active before, autolight enabled, light off
   if(t == "1" && litAF == false && autolight == true && light == false){
     //turn light on, turn autolight off, button now active,
     Serial.write("LightON");
     digitalWrite(lightbulb, HIGH);
     light = true;
     litAF = true;
     autolight = false;
     t = "pie";
   }
   //button pressed, button active before, autolight off, light on
   if(t == "1" && litAF == true && autolight == false && light == true){
     //turn light off, turn autolight back on, button no longer active
     digitalWrite(lightbulb, LOW);
     light = false;
     autolight = true;
     litAF = false;
     t = "pie";
   }
```

Figure 14: LED Code Snippet

The VCNL4000 Proximity Sensor, is responsible for determining the ambient light levels that the car is operating in. The PS takes the visible light and converts it to a current. It is converted into a current because lack of light cannot be determined by wavelength alone. It can read data from wavelengths 430 nm to 610 mn. These values are within the wavelengths that the human eye can see [3]. If the light levels are deemed too low, when “ambient” is below 150, then the LED on the Breadboard will turn on automatically. Likewise, if the light levels are normal, when “ambient” above 150, the LED will turn off. “ambient” is a variable in the code responsible for holding the converted bit reading from the light sensor on the PS. Also, this LED on the car can be turned on or off manually through the Android App by selecting “Light”. Once the “Light” button is selected the autolight function of the car is turned off. The LED will stay in its current state no matter the light levels. To return the autolight capabilities to the car, simply select the “Light” button again and the car will return to determining for itself if the light should be on or off. The application of this, is such that the car light can be manually turned off when in a dark area or manually turned on when in a lit area. The logic behind the LED test cases is shown in Figure 14.

The VCNL4000 Proximity Sensor, is also responsible for obtaining backup proximity data. The Parallax Laser Range Finder has an easier time getting an accurate proximity reading when objects are farther away, around 3 feet. Because of this, the PS was implemented to automatically stop the car if the LRF was not told to get a data reading in time and the car was within inches of an object. The PS is designed to be able to read proximity data with a range of up to 20 cm. It also takes multiple readings at once, taking into account environmental conditions, to determine an average to use as the main proximity reading [4]. This makes the VCNL4000 Proximity Sensor’s readings fairly accurate even when the car is moving. The PS takes readings continuously while the car is turned on. This way it can automatically instruct the car to stop if it gets within inches of an object.

```
// start servos
servoRight.attach(12);                    // Attach right signal to pin 12
servoRight.writeMicroseconds(1500);       // 1.5 ms stop


servoLeft.attach(13);                    // Attach left signal to pin 13
servoLeft.writeMicroseconds(1500);
```

Figure 15: Establishing Connection with the Wheel Motors Code Snippet

The Control Manager Module, controls the two servos that are connected the motors of the car's wheels. The left servo is connected to Pin 13 and the right servo is connected to Pin 12. This is demonstrated in Figure 15. By initializing each servo motor to 1500 ms the car is at 0 RPM. The RPM can change by changing the ms being written to the car by the Robot Manager. Any value less than 1500 ms will cause the car to go backwards will anything above 1500 ms will make the car move forward. Whenever, the Robot Manager is told by the Listener Module that there is an obstruction in the car's path, the Control Manager stops the car by changing the servo values to 1500 ms. Similarly, when movement is dictated by the User via the Android App the power being sent to the servo’s are adjusted accordingly.

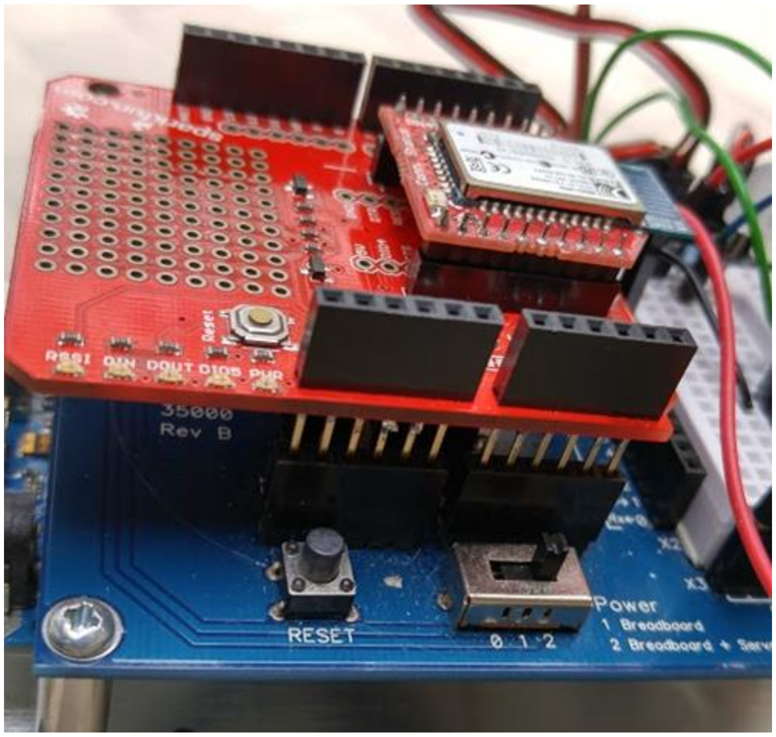


Figure 16: Establishing Connection of Bluetooth Hardware

The Bluetooth Module, enables communication between the Arduino and the Android App. The Bluetooth connection is established by connecting the RN42-XV Bluetooth Module to the XBee Sockets on the SparkFun XBee Shield which is attached to the Parallax Board of Education Shield. In order to attach the XBee Shield to the Parallax board, headers have to be soldered onto the XBee Shield. This is shown in Figure 16. This set-up is to ensure that connection is established between the Bluetooth Module and the Arduino so when the User sends a command through the Android Software, the command will be sent to the Arduino Software to execute.

4.2 Testing

The tests for the Android interaction with the Arduino robot are described in this section. The specific requirements tested are listed with the Test Case ID.

Test Case ID: 01, referring to number 1 requirement of the Arduino Software Requirements.

Objective: Test the functionality of the Arduino Software to respond to commands sent from the Android App.

Description: The Arduino Software shall be able to respond to given commands sent from the User of the Android App.

Test Conditions: The Arduino Car shall be on, and connected to the phone that has the Android App on it.

Expected Results: The Arduino Car is expected to be able to respond to given commands from the Android App.

Result Test Case ID: 01
Who Ran Test: Matthew Reilly
Environment: Arduino Studio
Pass/Fail: Pass

Test Case ID: 02, corresponding to requirement 2 of the Arduino Software Requirements

Objective: Test the functionality of the Arduino Software to be able to sense objects that are obstructing Robot’s path.

Description: The Arduino Robot will be able to use its Laser Range Finder and Proximity

Sensor to be able to deter Robot from objects.

Test Conditions: Robot shall be placed on a flat surface, away from obstructions.

Expected Results: The Laser Range Finder and Proximity Sensor shall alert the Arduino Robot of objects obstructing its path.

Result Test Case ID: 02
Who Ran Test: Julia Roscher
Environment: Arduino Studio
Pass/Fail: Pass

Test Case ID: 03, corresponding to requirement 3 of the Arduino Software Requirements

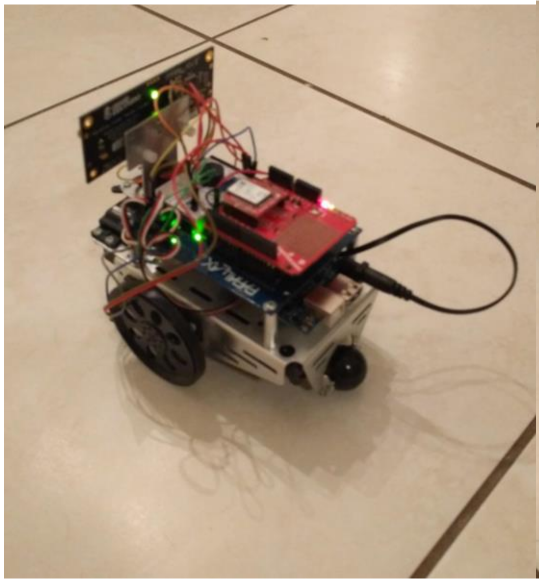
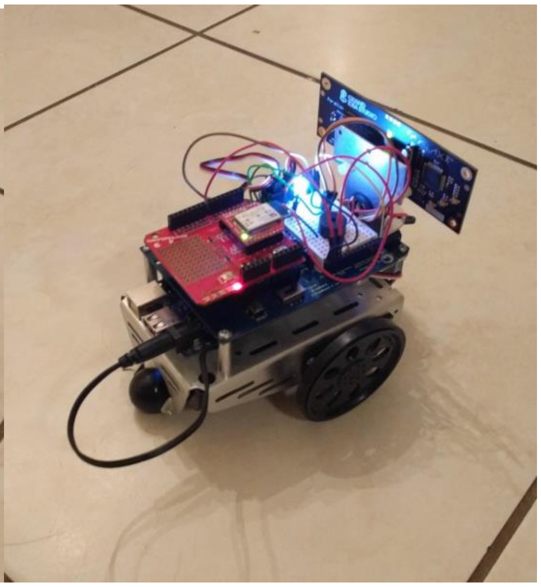
Objective: Test the functionality of the Arduino Software to be able to move the Arduino Robot forward, left, right, and backwards on command from the Android App.

Description: The Arduino Robot shall be able to move forward, left, right, and backwards, via commands from the Android App.

Test Conditions: Robot shall be placed on a flat surface, away from obstructions, while being connected to the Android App.

Expected Results: The Arduino Robot shall be able to move in all directions at the time commands are given to it from the Android App.

Result Test Case ID: 03
Who Ran Test: Julia Roscher
Environment: Arduino Studio
Pass/Fail: Pass



Robotic Car Turning Left: Robotic Car Turning Right:

Test Case ID: 04, corresponding to requirement 4 of the Arduino Software Requirements

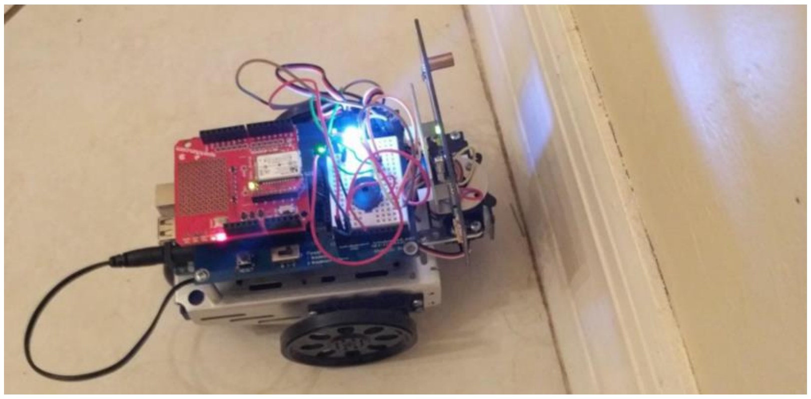
Objective: Test the functionality of the Arduino Software to stop itself if an obstruction gets in the way.

Description: The Arduino Software will stop the Arduino Robot if the Robot gets too close to an obstruction.

Test Conditions: Robot shall be placed on a flat surface, and be driven into an obstruction.

Expected Results: The Arduino Robot shall stop when an obstruction gets in front of the Arduino Robot.

Result Test Case ID: 04
Who Ran Test: Matthew Reilly
Environment: Arduino Studio
Pass/Fail: Pass



Robotic Car Stopped Right Before Wall:

Test Case ID: 05, corresponding to requirement 5 of the Arduino Software Requirements

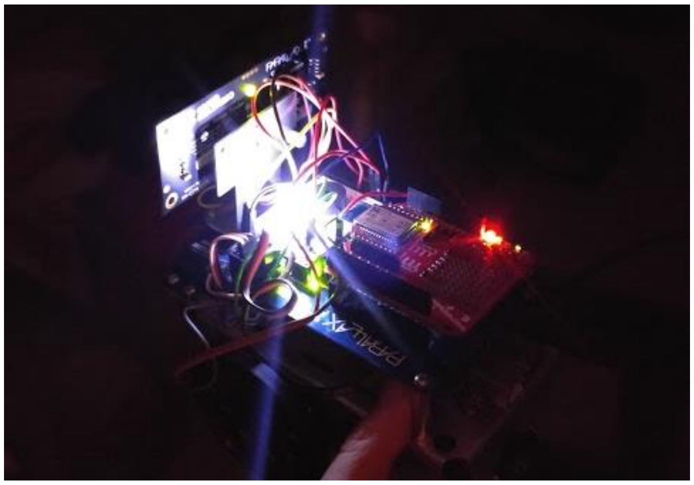
Objective: Test the functionality of the Arduino Software to be able to turn on an LED if the Arduino Robot is put into darkness.

Description: The Arduino Software shall be able to turn on an LED if the level of darkness gets high enough.

Test Conditions: Robot shall be placed in a room with a light that can be switched from on and off.

Expected Results: The LED light will turn on when the level of darkness if high enough.

Result Test Case ID: 05
Who Ran Test: Julia Roscher
Environment: Arduino Studio
Pass/Fail: Pass



LED On While Robotic Car is in Darkness:

<u>Test Case ID:</u> 06, corresponding to requirement 1 of the Android App Requirements

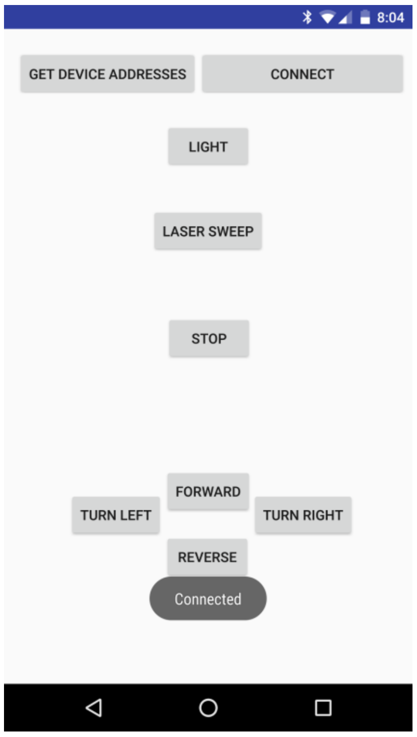
<u>Objective:</u> Test the functionality of the Android Software to be able to connect with local Bluetooth access points.

<u>Description:</u> The Android Software will be able to connect to the Bluetooth module on the Arduino Robot.

<u>Test Conditions:</u> Phone's Bluetooth will be on, as well as the Arduino Robot.

<u>Expected Results:</u> The Android Software shall be able to connect to the Arduino Robot, displaying the word "Connected" on the Android Software.

<u>Result Test Case ID:</u> 06
<u>Who Ran Test:</u> Matthew Reilly
<u>Environment:</u> Android 7.1.1
<u>Pass/Fail:</u> Pass



Screenshot of "Connected" on Android Phone:

<u>Test Case ID:</u> 07, corresponding to requirement 2 of the Android App Requirements

<u>Objective:</u> Test the functionality of the Android Software to be able to control the Arduino

Robot's direction.

Description: The Android Software will be able to send commands to the Arduino Robot to control the direction of the Robot.

Test Conditions: Phone shall be connected to the Arduino Robot, and the Arduino Robot will be on.

Expected Results: The Android Software shall be able to send directional commands to the Arduino Robot and will successfully perform actions.

Result Test Case ID: 07
Who Ran Test: Matthew Reilly
Environment: Android 7.1.1
Pass/Fail: Pass

Test Case ID: 08, corresponding to requirement 3 of the Android App Requirements

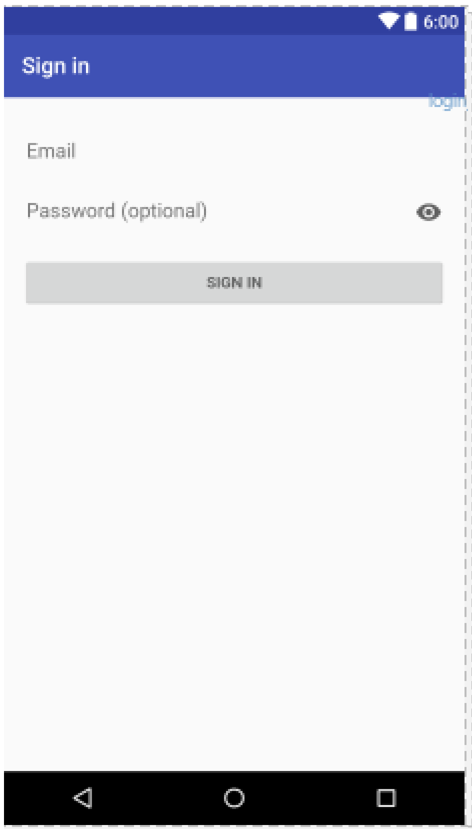
Objective: Test the ability of the Android Software to have a clean UI that enhances User experience.

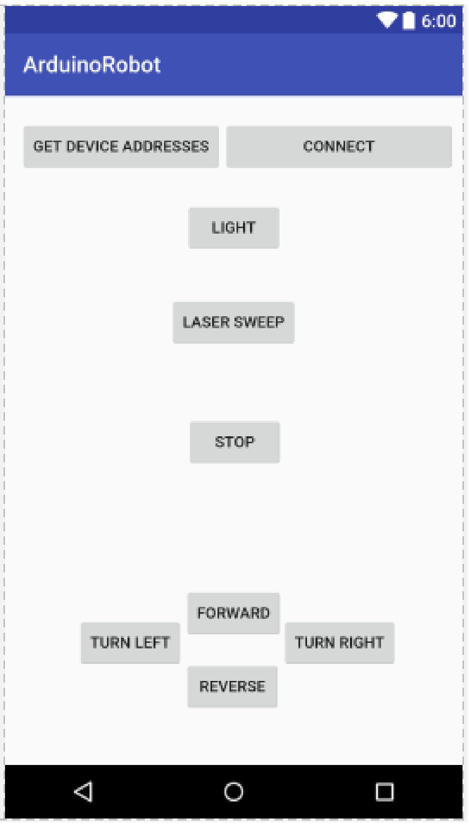
Description: The Android Software shall be able to give the User a crisp and clean experience while operating the software.

Test Conditions: Android Software will be running on the phone.

Expected Results: The Android Software will be able to give the User a crisp clean UI experience, letting the user to be able to navigate well through the software.

Result Test Case ID: 08
Who Ran Test: Julia Roscher
Environment: Android 7.1.1
Pass/Fail: Pass





Screenshot of Login: Screenshot of Main Page:

Test Case ID: 09, corresponding to requirement 4 of the Android App Requirements

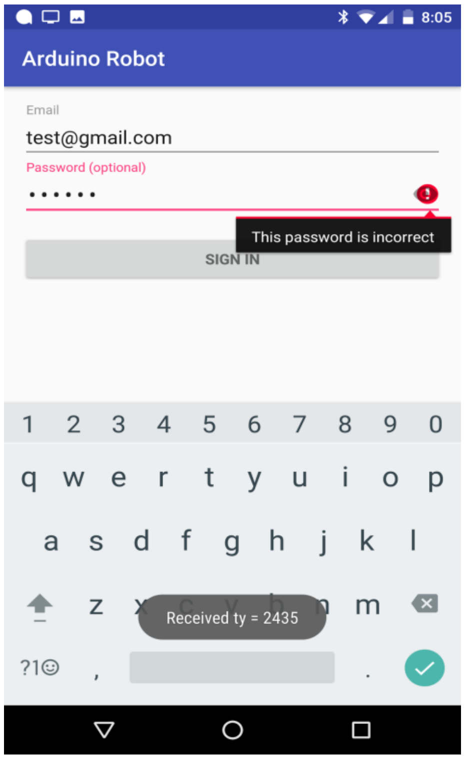
Objective: Test the ability of the Android Software to only allow the usage of the Software to authorized users.

Description: The Android Software will prompt a user for their email and password, and allow access if specified login is correct.

Test Conditions: Android Software will be running on the phone.

Expected Results: The Android Software shall be able to give access to the right email and password combination.

Result Test Case ID: 09
Who Ran Test: Matthew Reilly
Environment: Android 7.1.1
Pass/Fail: Pass



test@gmail.com

This password is incorrect

Received ty = 2435

Screenshot of Wrong Email and Password Combination:

Test Case ID: 10, Referring to requirements number 1, of the Design Constraints.

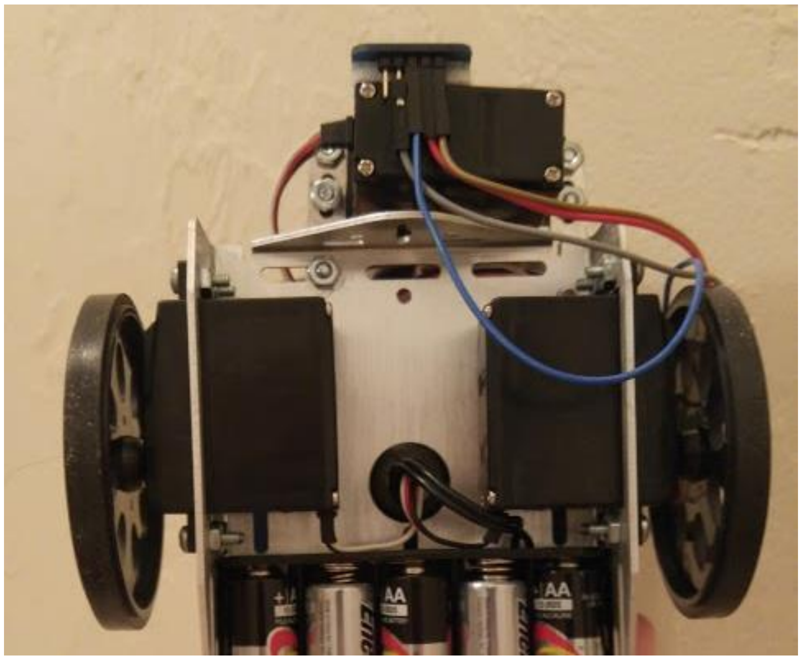
Objective: Test the functionality of the Robot Car's servo motors.

Description: Program the wheels to move forward and backwards and turn left and right

Test Conditions: Robot shall be placed on a flat surface, away from objects it can run into

Expected Results: The Robot Car shall move forwards and backwards and move left and right.

Result Test Case ID: 10
Who Ran Test: Matthew Reilly
Environment: Arduino Studio
Pass/Fail: Pass



Picture of Robotic Car's Servo Motors:

Test Case ID: 11, Referring to requirement 2 of the Design Constraints

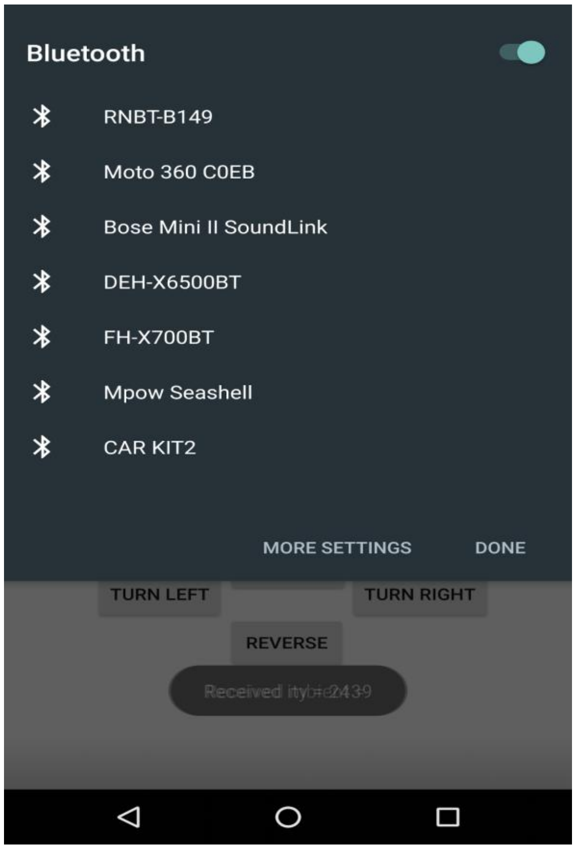
Objective: Test to make sure Arduino Robot has Bluetooth Connectivity.

Description: The Arduino Robot shall allow connection via Bluetooth.

Test Conditions: Arduino Robot shall be turned on.

Expected Results: The Arduino Robot shall allow for Bluetooth connection.

Result Test Case ID: 11
Who Ran Test: Julia Roscher
Environment: Arduino Robot
Pass/Fail: Pass



Screenshot of the Robotic Car's Bluetooth displaying on Android Phone Available Bluetooth Connections:

Test Case ID: 12, Referring to requirement 3 of the Design Constraints

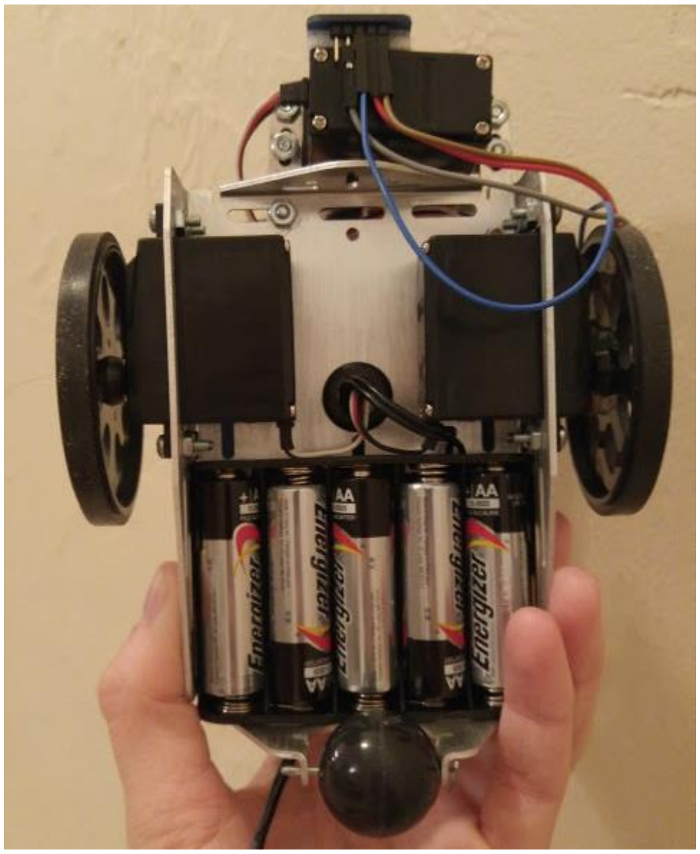
Objective: The Arduino Robot must have charged batteries in order to run.

Description: The Arduino Robot will need to have charged batteries or the Arduino Robot will not function properly.

Test Conditions: Arduino Robot shall have 5 AA batteries equipped on itself.

Expected Results: The Arduino Robot shall be able to turn on due to the charged batteries.

Result Test Case ID: 12
Who Ran Test: Matthew Reilly
Environment: Arduino Robot
Pass/Fail: Pass



Batteries Properly Equipped:

Test Case ID: 13, Referring to requirement 4 of the Design Constraints

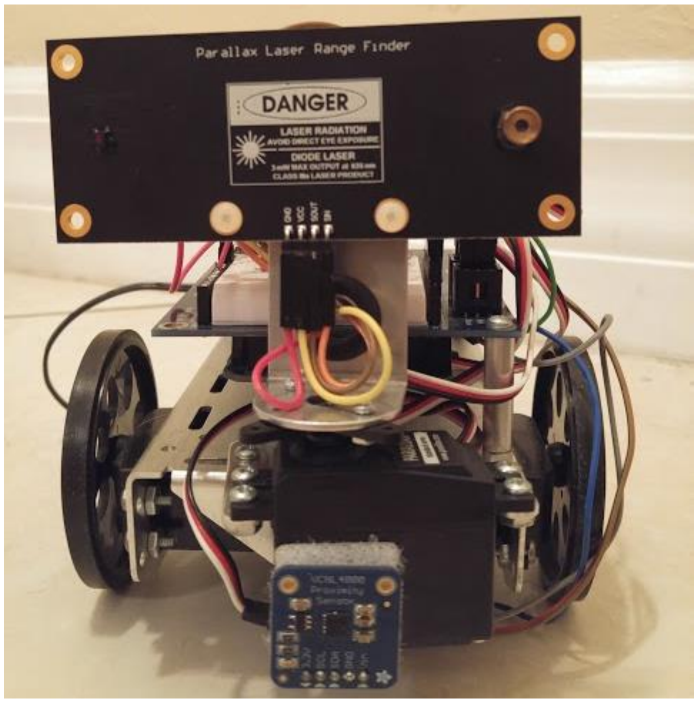
Objective: Test the Arduino Robot for the addition of Laser Range Finder.

Description: The Arduino Robot shall be equipped with a Laser Range Finder.

Test Conditions: Arduino Robot shall have the Parallax Laser Range Finder attached to body of the Robot.

Expected Results: The Arduino Robot will have the Parallax Laser Range Finder equipped.

Result Test Case ID: 13
Who Ran Test: Matthew Reilly
Environment: Arduino Robot
Pass/Fail: Pass


Parallax Laser Range Finder
DANGER
LASER RADIATION
AVOID DIRECT EYE EXPOSURE
DIODE LASER
CLASS IIIa LASER PRODUCT
VCNL4000 Proximity Sensor

Picture of Laser Range Finder:

Test Case ID: 14, Referring to requirement 5 of the Design Constraints

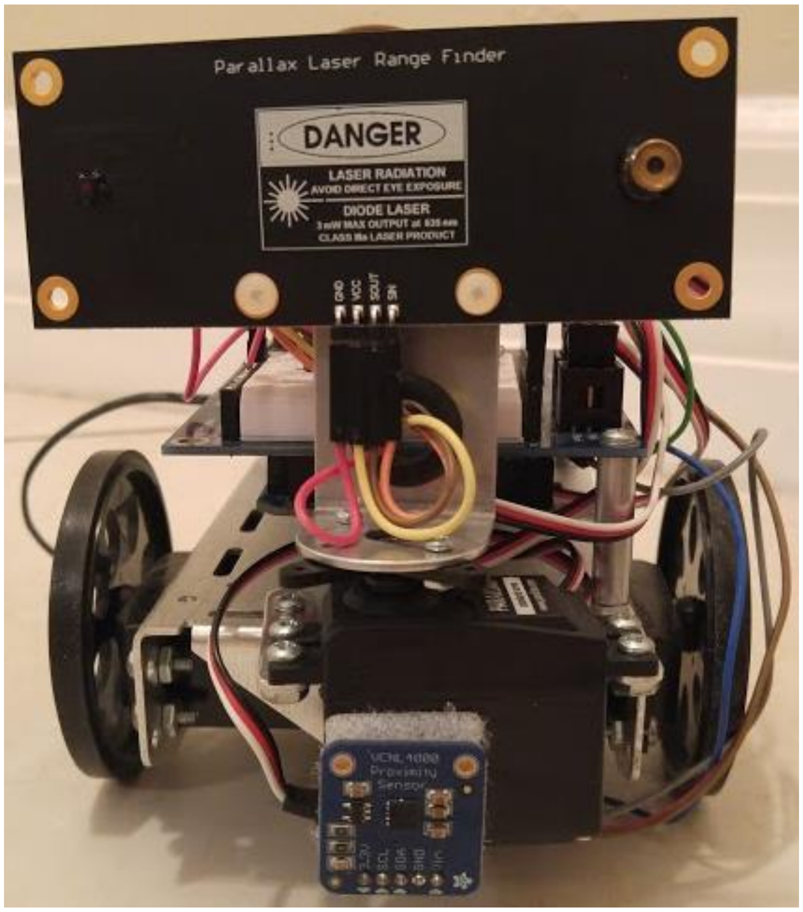
Objective: Test the Arduino Robot for the addition of the VCNL4000 Proximity Sensor.

Description: The Arduino Robot shall be equipped with a VCNL4000 Proximity Sensor.

Test Conditions: Arduino Robot shall have the VCNL4000 Proximity Sensor attached to body of the Robot.

Expected Results: The Arduino Robot will have the VCNL4000 Proximity Sensor equipped.

Result Test Case ID: 14
Who Ran Test: Matthew Reilly
Environment: Arduino Robot
Pass/Fail: Pass



Picture of VCNL4000 Proximity Sensor:

Test Case ID: 15, Referring to requirement 6 of the Design Constraints

Objective: Test the ability of the Android Software to be able to run on Android 4.4 and newer.

Description: The Android Software shall be able to run on Android 4.4 or newer.

Test Conditions: Android Phone that Android Software is on, shall have a version of Android 4.4 or newer.

Expected Results: The Android Software shall be able to run on expected versions.

Result Test Case ID: 15
Who Ran Test: Julia Roscher
Environment: Android 7.1.1
Pass/Fail: Pass

Test Case ID: 16, Referring to requirement 7 of the Design Constraints

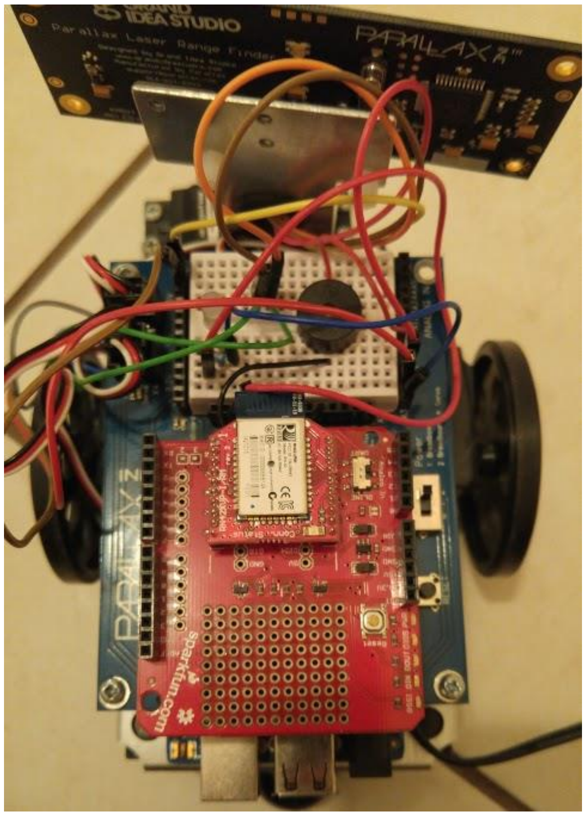
Objective: Test that the Bluetooth Module is attached correctly to the Arduino Board.

Description: The Bluetooth Module shall be attached onto the Arduino Board in order to allow correct communication between to too.

Test Conditions: Arduino Mega ADK attached to the XBee Shield Bluetooth Module

Expected Results: The Arduino Board will be able to communicate correctly with the Bluetooth Module.

Result Test Case ID: 16
Who Ran Test: Matthew Reilly
Environment: Arduino Mega ADK
Pass/Fail: Pass



Picture of Bluetooth Module:

Test Case ID: 17, Referring to requirement 8 of the Design Constraints

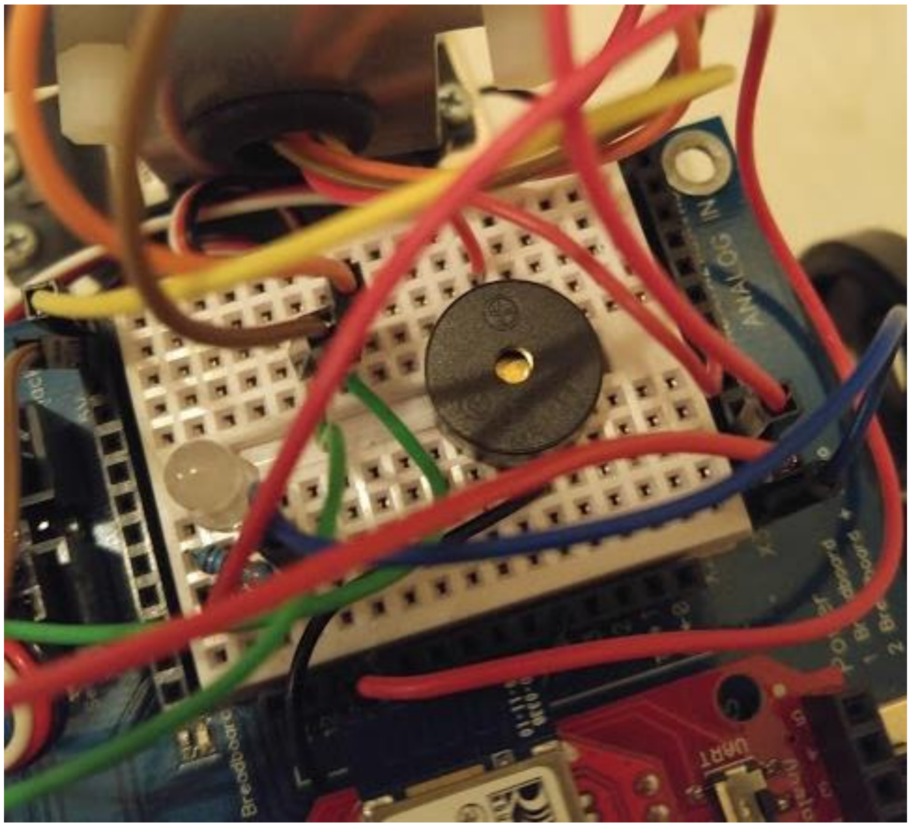
Objective: Test that there is correct wiring between the breadboard and controllers.

Description: The breadboard shall be wired and correctly attached to the appropriate controllers in order for correct functionality.

Test Conditions: Parallax Board with breadboard wired to appropriate controllers.

Expected Results: The Parallax Board will be able to properly communicate between it and the Arduino Robot's controllers.

Result Test Case ID: 17
Who Ran Test: Julia Roscher
Environment: Parallax Board of Education Shield
Pass/Fail: Pass



Picture of Wired BreadBoard:

5. Conclusion

5.1 Summary

The objective of this project was designing and creating a Remote Controlled Robotic Car with an Arduino Board through Bluetooth connectivity. The Robotic Car has the ability to be controlled by the User’s Android device. In this password protected app, the User can instruct the car to go forward, backward, left, right, or stop. The User can also instruct the car to perform a “laser sweep” with its laser range finder to determine if any object obscure its path or instruct the car to turn on an LED light. This LED light is connected to the Proximity Sensor which continuously takes in reading about ambient light levels; so the LED will turn on automatically in the dark. Lastly, the Robotic Car stops before running into large object thanks to the Proximity Sensor which will stop the motors automatically if a large object obstructs the car's path.

5.2 Difficulties

During the course of the project difficulties and problems arose that had to be resolved. These problems ranged from basic functionality to esthetics.

One of the biggest problems encountered when working with the Robotic Car was getting the code from the desktop program “Arduino” to upload onto the Arduino Mega ADK. The frustrating part was that the code would compile correctly on the desktop program but not upload to the physical Arduino. After intensive deduction through trial and error, it was discovered that the difficulties in uploading the code to the Arduino had to do with the SparkFun XBee Shield and Bluetooth Module connected through the Parallax Board of Education Shield to the Arduino. Having the XBee Shield connected to the Robot Car, when trying to upload code to it, would cause the connection to timeout because the XBee Shield was blocking full communication to the Board. To fix the problem, the XBee Shield with the Bluetooth Module have to be removed from the Robotic Car while uploading new code to the Arduino. The hardware can promptly be reattached after the upload is complete.

Another problem was enabling the Arduino to communicate with the Bluetooth Module. At first it was thought that the communication had to be assisted to the Bluetooth Module through a specialized serial port established to the Bluetooth through pins RX and TX. It was later discovered that such a specialized port actually hinders communication. To enable full constant communication, the whole XBee Shield, with the Bluetooth Module on top, had to be attracted to the pins on top on the Parallax Board of Education so all data could flow through it.

Also, a considerable problem encountered was that of making the Android Application email and password protected. Upon initial tries to add a page to the main page of the application, the whole app would crash after successfully logging in. In order to fix this issue, the entire project had to be remade in order to randomly generate another `Android_Manifest.xml`. When this was done, the Android Application was then able to work in the way it was intended to work.

5.3 Further Developments

Future developments could be completed to further enhance the Remote Controlled Robotic Car. These enhancements would increase the usability of the project. One such idea, would be to create a way for the Robotic Car to remember past devices it has been paired with and automatically connect to those devices when the app is active on the user's phone. That way the Robotic Car would not have to be manually connected every time the app is opened. Also, enhancements could be made to what happens when the sensors tell the car that something is obstructing its path. At the moment the car simply stops, but a good addition would be to instruct the car to stop, turn to a side, and continue moving. This way the car could be autonomous. Additionally, adding more proximity sensors would increase the functionality of the Robotic Car. By adding sensors to the sides and back of the car, the car will have a higher success rate of not running into objects. Lastly, usability enhancements could be made to the Android App's login section. These enhancements could include; having the app remember the email and password so that the User does not have to login new each time and to add a "Create New Account" feature so new users can be added,

6. References

[1] Davis, A, Hall, P, and Nafziger, A *Wireless Control of Autonomous Robot Maintenance Report.* Florida Gulf Coast University April 27, 2016

[2] Parallax INC, *Parallax Laser Range Finder (#28044)*. Parallax INC November 6, 2016

[3] Dizdarevic, S, VCNL400 - Integrated Proximity and Ambient Light Sensor, hackster.io, November 24, 2016, https://www.hackster.io/sdizdarevic/vcnl4000-integrated-proximity-and-ambient-light-sensor-d187bf

[4] Vishay, *Designing VCNL4000 into an Application.* Vishay, November 24, 2016