# Table of Contents

Abstract 2
Acknowledgments 2
Introduction 3
Customer Requirements and Engineering Specifications 4
Functional Decomposition 6
Time and Cost Estimates 11
Design 15
  Vehicle Hardware 15
  Vehicle Embedded Software 18
  iOS Application Software 21
Testing 23
Conclusion and Future Work 25
Annotated List of References 26
Appendix A: Senior Project Analysis 27
Appendix B: Links to Datasheets and Resources 33
Appendix C: Pictures of RC Car 34
Appendix D: Arduino Embedded Software 38
Appendix E: Apple Swift Software 45
Abstract

The smartphone-controlled RC (remote-controlled) car is an inexpensive remote-controlled car designed to be fast and portable. Instead of manufacturing, packaging, and shipping a separate controller, the remote control is implemented in a phone application, which saves time and money in both the design process and the manufacturing process. Utilizing the user’s smartphone is more cost-effective since mobile devices are a common recurrence, and packaging fewer devices results in overall better portability of the product.

This smartphone-controlled car is speedy and intuitive to learn for typical smartphone users. The user can change the car’s speed and direction wirelessly using their phone; the speed and direction also change in real-time. The wireless connection between the smartphone and the car is maintained for a reasonable distance with little to no delay in RC car movement. The RC car is small enough to be comfortably carried in a backpack or in one hand, but is still large enough to fit the internal hardware. The car has software embedded into the hardware to wirelessly communicate with the user. Based on the input data transmitted from the smartphone, the embedded software also sends signals on the hardware to change wheel speeds and directions.

Acknowledgments

I would like to take a moment to acknowledge that this project and my time here at Cal Poly would not have been successful without the help of many people. First, I would like to thank my advisor, Dr. Bridget Benson, for taking me on as a student, giving me the tools and resources I needed for learning, and being flexible in supporting this project. I would like to thank my former professors, Dr. Vladimir Prodanov and Professor Steve Dunton for helping me with my initial drafts of this report and teaching me about the design process in theory and application. I would also like to thank Dr. Paul Hummel for providing opportunities and encouraging me to pursue this field of electrical and computer engineering where I feel challenged, yet at home. Finally, I would like to thank my friends, my academic peers and mentors, and my family for their endless love, support, and wisdom. Thank you all! Although this is a solo project, I could not have completed this project without this entire community’s support.
Introduction

The phone-controlled RC car was first conceptualized as a way to provide greater accessibility to people that enjoyed cars. Many of these toys can be expensive to buy, and even the “cheap” ones can be expensive to manufacture, especially the cars that required controllers only designed for one type of car. Few companies sold these cheap versions capable of control via smartphone. Currently, the younger generation tends to be more fluent with smartphones. Smartphone control provides a business opportunity to offset the product cost by advertising to users or implementing subscription models; however, the scope of this project does not include this implementation.

Many high-end or high-performance RC cars can be upwards of $1000 USD [1], but most of the same enjoyment can be achieved with much less expensive cars. Companies sell cheap remote-controlled vehicles with comparable performance at typically $50 USD [1].

The target audience consists of mostly young males that have mobile devices, or more specifically smartphones. The typical age range would fall between 7-years-old and 15-years-old; however, any customer with prior smartphone knowledge should be able to use and have fun with this remote-controlled vehicle. Young males are most likely to buy and play with RC cars, and many similar companies and cars within the same price range target the same audience.

Many children in the typical age range physically go to school, and those children typically love to share or show off their toys. The RC car needs to be small enough to fit inside backpacks or carried in one hand for easy transportation between school or the park and the car’s place of storage. This portability is another reason that a separately-manufactured controller would not have been ideal for this design: carrying two objects is more demanding than carrying one object.

The smartphone-controlled RC car is intended to be a new standard and a role model for increasing accessibility to cheap and fun products. The remote-controlled vehicle is inexpensive to manufacture, convenient to transport, and simple to use.
## Customer Requirements and Engineering Specifications

### TABLE I: CUSTOMER REQUIREMENTS AND ENGINEERING SPECIFICATIONS

<table>
<thead>
<tr>
<th>Customer Requirements</th>
<th>Engineering Specification</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b</td>
<td>1. Top speed &gt; 3 miles per hour. Record various distances and time to travel those distances.</td>
<td>The car needs to be slightly faster than typical humans walk to be perceived as fast.</td>
</tr>
<tr>
<td>b, c, f</td>
<td>2. Time to learn how to use &lt; 2 minutes. A brief guide or tutorial is crucial.</td>
<td>Younger people tend to get distracted after a shorter period of time. It needs to be easy-to-use for users to not lose interest.</td>
</tr>
<tr>
<td>d, e</td>
<td>3. Overall cost of parts &lt; $50.</td>
<td>Overall cost needs to be cheaper than the competition for RC cars on the market. Similar RC cars typically sell for $40 to $50 [1].</td>
</tr>
<tr>
<td>a, b, c, f</td>
<td>4. Delay between input and movement &lt; 100 milliseconds.</td>
<td>Perception of what is “instant” to humans is less than 100ms [2].</td>
</tr>
<tr>
<td>d, e</td>
<td>5. Overall car size &lt; 1 cubic foot.</td>
<td>The user should be able to carry it in one hand.</td>
</tr>
<tr>
<td>b, f</td>
<td>6. Battery life &gt; 30 minutes.</td>
<td>The car needs to be able to maintain communication and movement for a reasonable timeframe.</td>
</tr>
<tr>
<td>a, b, f</td>
<td>7. Consistent wireless connection &gt; 25 feet.</td>
<td>The user should have the option to stay in one place while moving the car around it. The car needs decent range for connectivity to be responsive for reliable use.</td>
</tr>
</tbody>
</table>

Customer Requirements

a) Fast - The vehicle must be quick.
b) Fun - The vehicle must be fun to use and watch.
c) Intuitive - The phone application must be simple and easy to use.
d) Inexpensive - The overall cost of the parts required for the car should be cheap.
e) Portable - The car must be relatively small
f) Responsive - The user’s input must update car movements in real-time
### TABLE II: ENGINEERING SPECIFICATIONS

<table>
<thead>
<tr>
<th>Spec. #</th>
<th>Parameter</th>
<th>Target (units)</th>
<th>Tolerance</th>
<th>Risk (H, M, L)</th>
<th>Compliance (A, T, S, I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Top speed of car</td>
<td>10 mph</td>
<td>3 mph to 20 mph</td>
<td>H</td>
<td>T, S</td>
</tr>
<tr>
<td>2</td>
<td>Time to learn how to use</td>
<td>1 min</td>
<td>0 min. to 2 min.</td>
<td>L</td>
<td>T, I</td>
</tr>
<tr>
<td>3</td>
<td>Overall cost of parts</td>
<td>$30 USD</td>
<td>$10 to $50</td>
<td>M</td>
<td>S, I</td>
</tr>
<tr>
<td>4</td>
<td>Delay between input and movement</td>
<td>50 ms</td>
<td>10 ms to 100 ms</td>
<td>H</td>
<td>A, T, I</td>
</tr>
<tr>
<td>5</td>
<td>Overall car size</td>
<td>0.7 cubic ft</td>
<td>0.3 cubic ft to 1.0 cubic ft</td>
<td>M</td>
<td>A, S, I</td>
</tr>
<tr>
<td>6</td>
<td>Battery life</td>
<td>60 min.</td>
<td>30 min. to 120 min.</td>
<td>L</td>
<td>T, S</td>
</tr>
<tr>
<td>7</td>
<td>Consistent wireless connection</td>
<td>30 ft</td>
<td>25 ft to 40 ft</td>
<td>M</td>
<td>A, T</td>
</tr>
</tbody>
</table>
TABLE III:
LEVEL 0 FUNCTIONAL DECOMPOSITION INFORMATION

<table>
<thead>
<tr>
<th>Module</th>
<th>Smartphone-Controlled RC Car</th>
</tr>
</thead>
</table>
| Input(s)                                         | ● User Input - information on how to move car  
  ● Battery - energy storage for powering integrated circuits in RC car |
| Output(s)                                        | ● Rotational Energy - rotating the tires for movement  
  ● Mechanical Energy - steering vehicle or tires  
  ● Thermal Energy - heat dissipation |
| Functionality                                    | ● This system dictates communication between the smartphone controller and the remote-control car.  
  ● The phone wirelessly controls the car's direction and speed  
  ● The car wirelessly drives around based on user input |

Figure 1: Level 0 Diagram

Figure 2: Level 1 Diagram
### TABLE IV: LEVEL 1 FUNCTIONAL DECOMPOSITION INFORMATION

<table>
<thead>
<tr>
<th>1.1</th>
<th>Module</th>
<th>Smartphone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input(s)</td>
<td>● User Input - information on how to move car</td>
<td></td>
</tr>
<tr>
<td>Output(s)</td>
<td>● Electromagnetic Waves - radio waves transmitting car movement information</td>
<td></td>
</tr>
</tbody>
</table>
| Functionality | ● Converts the user input into movement data  
                   ● Transmit data to car |

<table>
<thead>
<tr>
<th>1.2</th>
<th>Module</th>
<th>Remote-Control Car</th>
</tr>
</thead>
</table>
| Input(s) | ● Electromagnetic Waves - radio waves transmitting car movement information  
         ● Battery - energy storage for powering integrated circuits in RC car |
| Output(s) | ● Rotational Energy - rotating the tires for movement  
         ● Mechanical Energy - steering vehicle or tires  
         ● Thermal Energy - heat dissipation |
| Functionality | ● Converts radio waves to movement data  
                   ● Converts movement data into control signals to power wheel movement and steering |

---

**Figure 3: Level 2 Diagram**
## TABLE V:
LEVEL 2 FUNCTIONAL DECOMPOSITION INFORMATION

<table>
<thead>
<tr>
<th>1.1.1</th>
<th>Module</th>
<th>Convert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input(s)</td>
<td>● User Input - information on how to move car</td>
<td></td>
</tr>
</tbody>
</table>
| Output(s) | ● Binary Data - data to transmit to car  
               ● Binary Data - data to display on smartphone screen for user feedback |
| Functionality | ● Convert user input to data for car movement and data to display feedback on smartphone screen |

<table>
<thead>
<tr>
<th>1.1.2</th>
<th>Module</th>
<th>Modulate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input(s)</td>
<td>● Binary Data - data to transmit to car</td>
<td></td>
</tr>
<tr>
<td>Output(s)</td>
<td>● Voltage Signal - signal carrying car movement data</td>
<td></td>
</tr>
<tr>
<td>Functionality</td>
<td>● Modulate car movement data into a voltage signal</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1.1.3</th>
<th>Module</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input(s)</td>
<td>● Binary Data - data to display on smartphone screen for user feedback</td>
<td></td>
</tr>
<tr>
<td>Output(s)</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Functionality</td>
<td>● Display movement feedback on screen for user</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1.1.4</th>
<th>Module</th>
<th>Convert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input(s)</td>
<td>● Voltage Signal - signal carrying car movement data</td>
<td></td>
</tr>
<tr>
<td>Output(s)</td>
<td>● Electromagnetic waves - radio waves carrying car movement data</td>
<td></td>
</tr>
<tr>
<td>Functionality</td>
<td>● Convert and transmit data to car from smartphone</td>
<td></td>
</tr>
<tr>
<td>Module</td>
<td>Convert</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>Input(s)</td>
<td>Electromagnetic waves - radio waves carrying car movement data</td>
<td></td>
</tr>
<tr>
<td>Output(s)</td>
<td>Voltage Signal - signal carrying car movement data</td>
<td></td>
</tr>
<tr>
<td>Functionality</td>
<td>Receive radio waves from smartphone</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Convert radio waves into voltage signal carrying movement data</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input(s)</td>
<td>Battery - energy storage for powering integrated circuits in RC car</td>
</tr>
<tr>
<td>Output(s)</td>
<td>VDC - DC Voltage for powering integrated circuits</td>
</tr>
<tr>
<td>Functionality</td>
<td>Storing power for RC car</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module</th>
<th>Demodulate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input(s)</td>
<td>Voltage Signal - signal carrying car movement data</td>
</tr>
<tr>
<td>Output(s)</td>
<td>Binary Data - car movement data</td>
</tr>
<tr>
<td>Functionality</td>
<td>Demodulate voltage signal into readable binary data for controller</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module</th>
<th>Demultiplex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input(s)</td>
<td>Binary Data - car movement data</td>
</tr>
<tr>
<td>Output(s)</td>
<td>Binary Data - tire rotational movement data</td>
</tr>
<tr>
<td></td>
<td>Binary Data - tire direction data</td>
</tr>
<tr>
<td>Functionality</td>
<td>Convert binary data received from smartphone into binary data to control tire rotation speeds and directions</td>
</tr>
<tr>
<td>1.2.5</td>
<td><strong>Module</strong></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><strong>Input(s)</strong></td>
<td>● Binary Data - tire rotational movement data</td>
</tr>
<tr>
<td><strong>Output(s)</strong></td>
<td>● Voltage Signal - a signal to power tire rotation</td>
</tr>
<tr>
<td><strong>Functionality</strong></td>
<td>● Modulate rotational data into signal to power tire-rotating mechanism</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1.2.6</th>
<th><strong>Module</strong></th>
<th>Modulate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input(s)</strong></td>
<td>● Binary Data - tire direction data</td>
<td></td>
</tr>
<tr>
<td><strong>Output(s)</strong></td>
<td>● Voltage Signal - a signal to adjust and maintain direction</td>
<td></td>
</tr>
<tr>
<td><strong>Functionality</strong></td>
<td>● Modulate the direction data into signal to power direction mechanism</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1.2.7</th>
<th><strong>Module</strong></th>
<th>Convert</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input(s)</strong></td>
<td>● Voltage Signal - a signal to power tire rotation</td>
<td></td>
</tr>
<tr>
<td><strong>Output(s)</strong></td>
<td>● Rotational Energy - tire rotation</td>
<td></td>
</tr>
<tr>
<td><strong>Functionality</strong></td>
<td>● Convert voltage signal into rotational energy to move the car with the desired speed</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1.2.8</th>
<th><strong>Module</strong></th>
<th>Convert</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input(s)</strong></td>
<td>● Voltage Signal - a signal to adjust and maintain direction</td>
<td></td>
</tr>
<tr>
<td><strong>Output(s)</strong></td>
<td>● Mechanical Energy - mechanism changing car’s direction</td>
<td></td>
</tr>
<tr>
<td><strong>Functionality</strong></td>
<td>● Convert voltage signal into mechanical energy to adjust and maintain the car’s direction</td>
<td></td>
</tr>
</tbody>
</table>
## Time Estimates

### EE 460 Gantt Chart

<table>
<thead>
<tr>
<th></th>
<th>September</th>
<th>October</th>
<th>November</th>
<th>December</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Week 1</td>
<td>Week 2</td>
<td>Week 3</td>
<td>Week 4</td>
</tr>
<tr>
<td>Labor Hours</td>
<td>20</td>
<td>27</td>
<td>4</td>
<td>11</td>
</tr>
</tbody>
</table>

**Project Plan**

- Abstract and Introduction: 5
- Requirements and Specifications: 8
- Functional Decomposition: 6
- Literature search: 12
- Gantt Chart: 8
- Cost Estimates: 3
- ABET Sr. Project Analysis: 8
- Report V1: 8
- Report V2: 8
- Report V3: 8
- Final Report: 10

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EE 460</td>
<td>Total</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4: EE 460 Gantt Chart
<table>
<thead>
<tr>
<th>EE 461 Gantt Chart</th>
<th>January</th>
<th>February</th>
<th>March</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EE 461</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Design V1</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware Design</td>
<td>3</td>
<td>10</td>
<td>17</td>
</tr>
<tr>
<td>Hardware Simulation</td>
<td>24</td>
<td>31</td>
<td>7</td>
</tr>
<tr>
<td>Microcontroller Software</td>
<td>31</td>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>Smartphone Software</td>
<td>7</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td><strong>Build and Test V1</strong></td>
<td>4</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>Assembling Hardware</td>
<td>2</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Debugging Software</td>
<td>4</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>Hardware Testing</td>
<td>3</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>Software Testing</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>System Testing</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Design V2</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware Design</td>
<td>4</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>Hardware Simulation</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Microcontroller Software</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Smartphone Software</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Build and Test V2</strong></td>
<td>4</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>Assembling Hardware</td>
<td>4</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>Debugging Software</td>
<td>8</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Hardware Testing</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Software Testing</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>System Testing</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

| EE 461 Total | 60 |

Figure 5: EE 461 Gantt Chart
<table>
<thead>
<tr>
<th>EE 462 Gantt Chart</th>
<th>March</th>
<th>April</th>
<th>May</th>
<th>June</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Week 1</td>
<td>Week 2</td>
<td>Week 3</td>
<td>Week 4</td>
</tr>
<tr>
<td>EE 462</td>
<td>28</td>
<td>4</td>
<td>11</td>
<td>18</td>
</tr>
<tr>
<td>Final Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware Simulation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microcontroller Software</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smartphone Software</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assembling Hardware</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Debugging Software</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware Testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Demonstrations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EE 462 Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6: EE 462 Gantt Chart
### Cost Estimates

#### TABLE VI: COST OF MATERIALS

<table>
<thead>
<tr>
<th>Part (Quantity)</th>
<th>Description</th>
<th>Unit Cost</th>
<th>Total Cost</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smartphone (1)</td>
<td>User input device that communicates movement information wirelessly to RC car</td>
<td>$0</td>
<td>$0</td>
<td>User provides smartphone. Needs to have available wireless chip installed onboard to communicate with RC car.</td>
</tr>
<tr>
<td>Low-Voltage Motor Wheels (4)</td>
<td>Powered by modulated signals from microcontroller, speed and direction adjusts</td>
<td>$3</td>
<td>$12</td>
<td>Will be used to turn the car and move the car forward and backward. Speed and direction adjust based on power to individual wheels.</td>
</tr>
<tr>
<td>Microcontroller (1)</td>
<td>A controller with embedded code to control binary data coming in and out, then sending signals controlling other peripherals</td>
<td>$15</td>
<td>$15</td>
<td>Will be used to convert binary data from smartphone into movement data. Based on movement data, will modulate and send signals to low-voltage motor wheels.</td>
</tr>
<tr>
<td>Wireless Receiver Chip (1)</td>
<td>Receives signals from the smartphone and converts into binary data</td>
<td>$5</td>
<td>$5</td>
<td>Will be used for receiving user input and converting into binary data. Necessary for sending to microcontroller.</td>
</tr>
<tr>
<td>Batteries (2)</td>
<td>Provides power using a dedicated voltage and current output</td>
<td>$20</td>
<td>$40</td>
<td>Will be used to power the microcontroller and circuitry that rotates the wheels of the car.</td>
</tr>
<tr>
<td>Wiring, Breadboards, and Car Frame</td>
<td>Materials to hold car together, connect modules, and protect circuit components</td>
<td>$8</td>
<td>$8</td>
<td>Frame is necessary for car aesthetic and circuit protection, will help with stable car movement. Wiring needed to connect microcontroller to motor wheels and wireless receiver chip.</td>
</tr>
</tbody>
</table>

**Total Cost of Materials:** $80

Based on an average salary of $82,000 for an entry-level electrical engineer in California, plus an additional 35% to employee benefits, this brings cost to $110,700 per year. Assuming the average work time of 48 weeks per year at 40 hours a week, this brings the hourly cost of labor to $57.65 per hour. The estimated number of labor hours is 180 hours at $57.65 per hour.

**Total Cost of Labor:** $10,377

**Total Cost of Labor and Materials:** $10,377 + $80 = $10,457
Design

Vehicle Hardware

Integrated Circuits:
IC0 Microcontroller = Arduino Nano 33 IoT from Arduino
IC1 Motor Driver = L298N Dual H-Bridge Motor Driver from Handson Technology

One of the first steps in building this car was deciding the kind of motor that would be used to rotate the wheels. The responsibility falls on a set of 4 H-Bridge DC voltage motors, 1 for each wheel [8]. Sending a negative voltage means the motors will rotate in the opposite direction. With a microcontroller that is only capable of max voltage or 0 volts, this requires using two different pins, one for positive voltage control and one for negative voltage control. Since the rotational speed increases as the magnitude of the voltage increases, it was also easy to adjust the speed of each individual wheel through pulse-width modulation with a basic square wave. In order to turn, the car adjusts the speeds of the wheels so that the inner wheels are moving slower than the outer wheels. For example, on a left turn, the left wheels move slower than the right wheels so that the entire body of the car rotates left. Different speeds of each side are achieved by sending lower average voltages to the inner side via adjusting the duty cycle of the PWM signal. Adjusting the duty cycle was an easy logical process using a microcontroller where the controller simply turned on or off a pin’s output to send a square wave. This signal needed to be amplified to provide enough power to each wheel, so this PWM became a control signal by passing through a separate motor driver instead of directly from the microcontroller.
This motor driver acts as a switch and current amplifier. Whenever one of the 4 logic input pins are high, the driver passes through the input voltage connected to the 12V pin to the corresponding output pin, which is eventually connected to the wheels. The microcontroller will control the 4 signal input pins according to desired direction and speed. Since the 12V input is being used to power the driver, the 5V pin with the jumper enabled acts as an output and powers the microcontroller at its VIN pin. The motor driver is powered by two 7.4V batteries [9] capable of 2200mAh. In series, this makes it a 14.8V source, or approximately a voltage source of 15V. This voltage will be passed through with a high current whenever one of the outputs is activated by the relative logical input.

The smartphone is provided by the user: it requires Bluetooth capabilities and a touchscreen. This project was developed for iOS so only iPhones or iPads would work with this RC car. However, writing the software for devices that use Android would be the next step in providing more accessibility to these cars.
The Arduino Nano 33 IoT is the microcontroller in charge of adjusting the duty cycle of the motor control signal. This Arduino contains support circuitry for sending PWM signals using the built-in function `analogWrite`, and it supports Bluetooth and WiFi through an on-board wireless communication chip. Once set, the microcontroller will continue sending that PWM signal while getting updated values from the smartphone via Bluetooth, so the car does not need to stop in order to retrieve updated speeds. The microcontroller will turn on the LED on pin D13 whenever there is an established connection between the microcontroller and the smartphone. In order to get two different variable PWM signals, the outputs are connected to pins that are on two different PWM timers, PA and PB. The pins capable of `analogWrite` are marked by the “~” character.
The microcontroller is the most important in the process: it is the translator between the smartphone and the circuitry. After setting pins as outputs, establishing a bluetooth peripheral service with characteristics, and advertising that service to other devices, it needs to connect to a central device. In this case, the central device is the smartphone that will be writing over the values in the characteristics determining speed and steering.
Once the smartphone (central device) connects to the Arduino (peripheral device), the Arduino’s orange led on pin D13 will light on to indicate the connection is established. Now, the smartphone is able to write to the speed and steering characteristic values. Typically writing to a peripheral’s posted characteristics is done either with or without a response. To speed up reaction time of the car, writing without a response proved way faster and more efficient since lots of bluetooth data with minor changes would be sent quickly. The microcontroller will check if it is still connected, and then if one or both of the values have been overwritten, then it updates both values and then updates the movement of the wheels on both the left and right side of the vehicle. It continues of this loop of checking to see it is still connected, checking and reading recently written values, and updating the motor speeds until the bluetooth disconnects. While in this loop, the user is able to send control signals to move forward, backward, turn left or right while moving forward or backward, and stopping. If the bluetooth connection is broken, the vehicle stops all motors and starts looking for a central device to repeat the process again, in case connection is lost over a large distance or the phone severs the bluetooth connection. [5]

Since the Arduino Nano 33 IoT has the built-in bluetooth, the Arduino sets up the local name, the service, the characteristics of speed and steering for that service, and starts advertising. A central device can then connect and start giving values for speed and steering if it knows how to do so.
Figure 12: PWM Signal Setup and Control

Speed is set as an integer that ranges on a scale from -255 to +255, negative meaning reverse speed and positive meaning forward speed. Steering is a percentage represented by an integer on a scale ranging from -100% to +100%, representing turning left (-100%) and turning right (+100%). Both have a set deadzone of 17%, so if the user is in that deadzone close to the center, it will set the corresponding value to 0.

Each time the Arduino reads the bluetooth values, it sends that speed and steering values to other functions that break down that data into PWM bytes for forward or backward control pins. If the values are not in the deadzone, then the steering percentage is used to adjust the inner wheel speed. For example, if the car is meant to be turning left with a turning ratio of 80%, then the inner wheels (left side if turning left) will be adjusted to rotate at 100 - 80 = 20% of the input speed, and the outer wheels (right side if turning left) will still be moving at the given input speed. Once the speed of the left wheels and the right wheels have been determined, the microcontroller sets up the PWM for the pins. The built-in analogWrite PWM output is based on 0 to 255. If the speed value is positive, the forward motors are given that speed and the backward motors are set to 0. If the speed value is negative, the backward motors are given the absolute value of that speed and the forward motors are set to 0.
The main work that’s done for the smartphone application is setting up the layout, taking touch input from the user, and connecting to the microcontroller through Bluetooth. After laying out the sliders and frames in the desired positions on the screen, the smartphone will detect any dragging gestures the user does on the slider. This allows the user to move two different sliders in different directions simultaneously, but only if it is within the slider’s box. Each slider can only move along one axis, the other axis has a fixed value. Whenever the user provides the touch screen a dragging or “panning” motion within the slider’s box, the slider’s position is moved and updated. To do so in code, it requires attaching a UIPanGestureRecognizer to the sliders [7]. Whenever those sliders are dragged across the screen, the recognizer calls a function that updates the bluetooth characteristics and animates the slider moving to a new position if it is still in the designated box.
This figure shows the layout for the user interface. The blue slider on the left dictates speed. Moving it to the top of the yellow box is full speed forward and the bottom of the yellow box is full speed backward. The blue slider on the right dictates steering, sliding it to the left turns the vehicle left and sliding it to the right turns the vehicle right. The sliders are limited to the boxes on their relative axis and can only be moved by dragging the slider inside the box. This layout is similar to many remote controllers for toy cars, planes, and helicopters so it is more intuitive for users.

This screen layout is only helpful if it can write to the microcontroller from the phone. It does so by establishing bluetooth connection. It starts by scanning for peripheral devices nearby, if it finds a match for a service and name, it connects and checks for matching characteristics. If the characteristics for speed and steering are both present, then it enables the function that contains bluetooth writing. The bluetooth data is only written if the characteristics are present and writeable.
TABLE VII:
ENGINEERING SPECIFICATIONS: OUTCOME AND ACHIEVEMENTS

<table>
<thead>
<tr>
<th>Spec. #</th>
<th>Parameter</th>
<th>Target (units)</th>
<th>Tolerance</th>
<th>Risk (H, M, L)</th>
<th>Actual</th>
<th>Spec. Achieved?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Top speed of car</td>
<td>10 mph</td>
<td>3 mph to 20 mph</td>
<td>H</td>
<td>3.6 mph</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Time to learn how to use</td>
<td>1 min</td>
<td>0 min. to 2 min.</td>
<td>L</td>
<td>0.5 min.</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>Overall cost of parts</td>
<td>$30 USD</td>
<td>$10 to $50</td>
<td>M</td>
<td>$80</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Delay between input and movement</td>
<td>50 ms</td>
<td>10 ms to 100 ms</td>
<td>H</td>
<td>30 ms</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>Overall car size</td>
<td>0.7 cubic ft</td>
<td>0.3 cubic ft to 1.0 cubic ft</td>
<td>M</td>
<td>0.165 cubic ft</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>Battery life</td>
<td>60 min.</td>
<td>30 min. to 120 min.</td>
<td>L</td>
<td>50 min.</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>Consistent wireless connection</td>
<td>30 ft</td>
<td>25 ft to 40 ft</td>
<td>M</td>
<td>29 ft</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The table above shows the expectations and targets for each specification provided at the beginning as well as the actual outcome and whether or not each specification was met.

With a wheel that is 2.7 inches in diameter, the fastest measured rotational speed was approximately 450 rpm, which translates to about 3817 inches per minute. At 229,000 inches per hour, the top speed of the car was calculated to have a top speed of about 3.6 miles per hour. Although it is well below the target, the specification was still met and it is still fast enough to be fun at 3 to 4 mph. The benefit of not having it go 10 mph is any human can run and catch up to it if the car or the phone malfunctions. The car maintained above those 3mph speeds for 50 minutes during project demonstrations, which fit the format of typical use of stopping, going, speeding up, slowing down, not going full speed all the time but still relatively fast. After about 90 minutes, the car was moving 2 mph or less at full throttle, where it was no longer considered a “fun” fast speed. For the sake of deeming the battery life, the 50-minute time segment seems to be more accurate and fitting since it is still able to work at high performance towards the end of that timeframe. In either case, both the battery life and the top speed specifications were met.

In terms of the latency between input movement and output behavior, it ended up being very difficult to define a good starting point that was also accurately measurable. Initially setting up the bluetooth to write with response, any updates had to be confirmed, so no updates were thrown away, even if they were virtually the same. This caused a huge backlog where it took 1-5 seconds for the proper behavior to show depending on how long the user had been dragging their fingers across the sliders. Once it was changed to enable writing without a response, all data updates were sent and only some samples were accepted and written. The Arduino printing log with timestamps showed it updated 2 sets of values approximately every 30ms; however, sometimes the values would be the same due to miniscule movements on the sliders on the phone.
In demonstrations and testing with people, many stated it was “very responsive” and most picked up the controls within about 10 seconds. At the Project Expo, the longest it took for one of the passerbys to get moving with it was 30 seconds, and it was partially due to a thick screen protector that made the smartphone less responsive to touch.

The furthest distance with consistent bluetooth connection achieved came out to be 29ft. Using a hallway, the car was turned slightly one way or the other while going forward, alternating going left and right until it would not respond to new input within a second. Since the bluetooth is set up for writing without a response, it is possible that the bluetooth waves may have reflected off surfaces within the hallway, potentially causing helpful or destructive interference, which could affect the car’s ability to consistently send and receive information.

The vehicle officially came out to 10 in. X 7 in. X 4 in. which was less than a foot in every direction, so it is a small enough size for someone to carry it with one hand. However, the parts were more expensive than intended. Initially the voltage source was just a typical 9V battery intended for low-power electronics. However, the motors were so current-hungry that the battery was drained from 9V to 5V in a matter of 20 minutes and the motors stopped rotating even at full throttle, which was not meeting speed or battery life expectations. When one 7.4V battery was implemented, it moved around but it was not meeting the speed standard, which was a more important factor for the vehicle’s success.

Although the hopeful cost of materials specification was not met, it is much more feasible to get the cost down now that there is a working prototype to compare and minimize. Putting more parts together in-house rather than preassembled by others reduces price. Typically the cost also scales down by buying parts in bulk, which would likely be done in a manufacturing scenario.

Figure 16: QR Code for Demonstrations (left); Picture of RC Car (right)
QR Code Link: https://youtu.be/7Mts0Jup8cI
Other pictures available in Appendix
Conclusion and Future Work

Given more time, this project could be a great product. If the vehicle could be scaled down to the size of a matchbox car, it could fit in kids pockets and would be very easy to manufacture and take on the go. Even going 1-2 mph, the smaller size relative to its environment makes 2 mph look very fast for an RC car that tiny. Another interesting concept might be swapping the radio controller for a bluetooth controller on in-store RC cars to maybe revolutionize the remote-control industry by manufacturing less controllers. It would also be relatively easy to add a circuit with a switch or a button to turn the whole system off or on, rather than plugging in and unplugging the system’s voltage source.

If I was able to spend more time on this project personally, I would like to spend more time with the user-interface. A more polished look with labels would prove very effective and helpful. With screen protectors, some users might struggle using the app since it depends on dragging, so swapping the code to respond to just touch in those boxes instead of dragging motions might prove more accessible and easier to use. Some peers and mentors introduced WiFi as an alternative and it might be interesting to implement that way, or even just using the smartphone as the radio instead of a separately manufactured radio controller. During testing and demonstrations, users notably had a hard time knowing whether their fingers were still in the box and dragging the sliders since they don’t look at the screen. They focus on the car. Haptic feedback in the direction the user is driving would significantly improve confidence and overall experience.

I found this project most exciting for building my first iOS app and learning about bluetooth. The last thing that might be interesting and recommended is implementing this for Android users as well. There are programs online that provide helpful resources to code the interfaces for Android and iOS simultaneously. However, bluetooth is a hardware-specific implementation. Even if there is one language that could code the user interface and experience for both easily, it cannot address the bluetooth chips specifically without swapping to the native coding language for that device family. Developing this bluetooth application for other operating systems like Windows, Linux, and MacOS would show flexibility to this market and encourage a future with maybe less waste for the same products.
Annotated List of References


Appendix A: Senior Project Analysis

1. Summary of Functional Requirements

The smartphone-controlled RC (remote-controlled) car is an inexpensive remote-controlled car designed to be fast and portable. Instead of manufacturing, packaging, and shipping a separate controller, the remote control is implemented in a phone application, which saves time and money in both the design process and the manufacturing process. Utilizing the user’s smartphone is more cost-effective since mobile devices are a common recurrence, and packaging fewer devices results in overall better portability of the product.

This smartphone-controlled car is speedy and intuitive to learn for typical smartphone users. The user can change the car’s speed and direction wirelessly using their phone; the speed and direction also change in real-time. The wireless connection between the smartphone and the car is maintained for a reasonable distance with little to no delay in RC car movement. The RC car is small enough to be comfortably carried in a backpack or in one hand, but is still large enough to fit the internal hardware. The car has software embedded into the hardware to wirelessly communicate with the user. Based on the input data transmitted from the smartphone, the embedded software also sends signals on the hardware to change wheel speeds and directions.

2. Primary Constraints

The smartphone-controlled RC car is a more modern approach to the classic RC car. The most challenging aspect of this project was developing the smartphone application to control the car. This was the most critical part because the ease of the user-interface and its responsiveness was a key factor in overall product satisfaction.

For the car to respond quickly to changes in user-input, a main goal of this project was minimizing the time difference between user-input and car movement. This meant optimizing the wireless communication interface between the smartphone and the car as well as determining which calculations were done by either the smartphone or the RC car microcontroller. Reducing the overall cost of parts per car was also difficult. It required trading manufacturability for performance in some areas, such as rotational speed of the wheels.
3. Economic

The greatest economic impact this remote-controlled car has is improved manufacturability. The overhead cost of running the assembly line via robots as well as human workers can be expensive. Since this product only requires building the car, it allows for a larger number of products to be output in the same amount of time, meaning the cost of manufacturing goes down as efficiency goes up. Very little human capital is required for assembly since it only consists of essentially connecting the correct wires. In order to manufacture many units for later sale, significant financial capital may be required for initial manufacturing as well as for marketing. For example, if 100 cars needed to be manufactured, an upfront cost at $40 per RC car would require $4000; however, in order to sell those cars to new customers, marketing costs of (again as an example) $1000 would be added to that $4000. This example is based on the constraint of a maximum of $40 for all the parts for a single RC car.

Since the product is relatively cheap, customers with a product that breaks will likely replace it rather than paying half the initial cost to repair it. This means more sales, but also greater waste of internal components like the battery and the frame of the car. The product is built to have a life cycle of multiple years of intermittent use; it is not dependent on the life cycle of the controller since the user can reconnect the car to another smartphone. However, poor care and improper or dangerous use of the product could reduce the lifespan significantly. The car was designed to run into walls and flip over, but constant damage to the car will result in a shorter lifespan.

Since this project required significant upfront investment in financial resources spent on development, likely thousands of RC cars will need to be sold to make profits. Assuming $10,000 of development costs, covering these development costs would require selling 1000 RC cars at a price of $50, a profit of $10 on each $40 car. More cars will need to be sold as the retail price approaches the manufacturing price with thinner profit margins. A large profit margin requires selling less cars but at a higher price that would attract less customers.

4. If manufactured on a commercial basis:

The smartphone-controlled RC car requires large-scale manufacturing to be profitable due to low cost of the item. With effective marketing targeted towards boys, young men, or parents of these boys, approximately 10,000 smartphone-controlled RC cars can be sold per year. Each car costs $40 in manufacturing expenses, including parts and labor in assembly, and each car sells for $50. Each car will profit $10, so the estimated profit for 10,000 cars sold per year is $100,000 per year.

The only costs for the user to operate will be costs of batteries and smartphone-usage, which can be approximated to a few pennies per hour of use since smartphones are used for many hours and batteries are very cheap. The user will perceive this as virtually “free” to operate since both batteries and smartphones are a common necessity amongst anticipated customers in the total addressable market.
5. **Environmental**

The smartphone-controlled RC car provides consumers an alternative that is less-damaging to the environment compared to typical RC cars that come packaged with a plastic controller. Cars that require separate controllers often become waste if the controller no longer works, however, our product can simply be reconnected to another smartphone. This saves waste that can affect the environment in regards to unnecessary waste of the car due to bad controllers as well as reducing pollution from manufacturing RC cars. Assuming this product is an alternative to typical car-controller combinations, our product is less harmful to the environment compared to competitors. However, from the perspective of just the product, it can be of some harm.

The user will be providing batteries, however, those batteries contribute to pollution of natural resources caused by electronic waste. Components used in assembly are made of metals, plastics, and rubber, which do not easily break down and could be harmful to animals that attempt to ingest components individually. In the manufacturing process, a highly-pollutant factory building our product could reduce the air quality and accelerate global warming.

6. **Manufacturability**

The smartphone-controlled RC car requires a plastic shell, a metal frame, copper wires, pre-built microcontrollers, and rubber wheels connected to pre-built motors. Potential difficulties include manufacturing reliant on the sale of other components being manufactured; however, the price can be reduced by selecting more widely available components. For example, selecting a very popular cheap microcontroller in design results in a consistent and reliable stream of microcontrollers due to high demand and resulting high supply. Many of the parts required are highly popular and well-supplied. The only parts that could cause delays would be the custom plastic shell and metal frame for holding the electrical components.
7. **Sustainability**

The smartphone-controlled RC car will be wasteful to the environment once it is no longer able to be used. To offset this, designing this vehicle to have a long life cycle for use was vital. Designing this RC car to be controlled via smartphone is the key to improving the sustainability compared to competitors that also manufacture controllers. This RC car is more sustainable than competitors because manufacturing separate controllers requires more plastic, rubber, and copper waste. If either the controller or the car is malfunctioning, then both are disposed of compared to our product, which can simply be reconnected to a different smartphone if the smartphone the user provides malfunctions. These RC cars require manufacturing on a large scale, which provides work to factories and factory workers as well as companies selling the individual components. Due to the portability, smartphone-controlled RC cars are easy to share and bring to other destinations, which will provide more opportunities for RC car enthusiasts to bond with other enthusiasts, classmates, friends, and family, essentially building greater community via entertainment and fun.

8. **Ethical**

The ethical impact of this project falls mostly in the hands of the user, but will likely cause few issues. Misuse of the product like driving this product into other people can cause minor injury, but could potentially seriously injure small animals if the user drives over them. The small nature of the car reduces the potential for physical harm, and the ability to share and show others is more likely to build community rather than negatively impact it.

In fairness, the user needs to provide a smartphone, which has many barriers to entry and ethical concerns in itself. Since people that buy this product likely have other luxuries, this product could amplify the gap in quality of life between people that have many luxuries and people that have few. If parts of this project were to be manufactured at a large scale in factories, sub-optimal factories could be underpaying or mistreating workers and polluting the local environment’s water or air quality.
9. **Health and Safety**

The smartphone-controlled RC car has very few concerns regarding safety. Misuse of the product or eating components could result in injury. The RC car is designed to be sturdy, so breaking the product and using it as a weapon is not a concern. Driving the car into other people or fragile obstacles could result in minor damage or injuries, but due to its small size it will likely cause minimal damage like a simple cut or bruise. The car ideally will bring people together and build community, however, building community can sometimes result in outsiders. If users are not kind to other people regarding utilizing or owning the product, this could be an issue for mental health of people that do not use the product.

The manufacturing of this product could have potential health implications if manufactured in factories with unhealthy working conditions. If the factories have high pollution, then manufacturing the car would be contributing to bad air quality and potentially global warming issues. A more likely health concern is related to the disposal of plastics or batteries used in the product, which contribute to mass electronic waste. High electronic waste causes environmental issues that often pollute resources used.

10. **Social and Political**

In order to provide a more sustainable and better quality alternative to competitors, this RC car is controlled by a smartphone instead of a traditional controller manufactured separately. The issue arises with the smartphone, or more specifically, who has access to the smartphone. Although smart cellular devices are becoming more widely used and popular, there are social inequalities between those that can and cannot afford a smartphone in order to buy and use this device. Requiring the smartphone as a barrier to use can further amplify those gaps and issues since not everyone will be able to use the product. However, there are alternatives offered by competitors that provide solutions to this inequity by manufacturing the controllers rather than requiring them for use. Providing more work for factories that have poor working conditions and high pollution could also be providing them financial power to lobby government officials and alter their perspectives on agendas that could reduce the factories profit margins.

With the portability and entertainment this device provides, it can be used to build community amongst those that can use smartphones, which is a large majority amongst the total addressable market. Smartphones and cars are easy to share temporarily, which can bring users together as well as provide fun entertainment to those that simply watch. Using smartphones instead of controllers provides easier portability to increase the efficiency of community-building, but it also supports the use of smartphones. Companies that develop and sell those smartphones can have negative political and social impacts in how they design their products, however, customers with those products were likely to get the item regardless of the influence of our product due to the versatility of smartphones.
11. Development

Developing the smartphone-controlled RC car required learning how to develop a smartphone application that could wirelessly connect and communicate with another wireless piece of equipment. It required learning the proper coding language for the application, documenting and applying for the program to be downloadable on the smartphone app store, and testing with different components and even other products with wireless functionality. It was also educational on the process of maintaining signal integrity and the benefits of different kinds of wireless communication in regards to speed, regulation, and stability.
Appendix B: Links to Datasheets and Resources

QR Code Smartphone-Controlled RC Car Demonstrations:
  https://youtu.be/7MtsoJup8cI

General Info:
  IoT: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7571181/

Bluetooth Debugging:
  Bluetooth Sniffing: https://www.bluetooth.com/blog/4-essential-tools-for-every-bluetooth-low-energy-developer/

Swift Debugging:

Arduino Debugging:
  PWM Help: https://docs.arduino.cc/tutorials/generic/secrets-of-arduino-pwm
  Nano 33 Stuff: https://docs.arduino.cc/hardware/nano-33-iot
  Nano 33 Datasheet: https://docs.arduino.cc/static/d7fb28381119e10c821e7169ff9fb9b/ABX00027-datasheet.pdf
  Internal Bluetooth Datasheet: https://content.arduino.cc/assets/Arduino_NINA-W10_DataSheet_%28UBX-17065507%29.pdf

Motor Driver:
  Driver Datasheet(s):
    https://components101.com/modules/l293n-motor-driver-module
    https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf

Motors:
  DC Motor Datasheet:
    https://media.digikey.com/pdf/Data%20Sheets/Adafruit%20PDFs/3777_Web.pdf?_gl=1*1vx66en*_ga*NjM5OTUyMDQuMTY0Mzg0MDQyOQ..*_ga_WRTVM74KVB*MTY0Mzg0MDQvOC4xLjEuMTY0Mzg0MDYxOC4w
  7.4V 2200mAh Batteries:
    Battery Datasheet: https://cdn.sparkfun.com/assets/8/6/4/a/f/Lithium_Ion_Battery_MSDS.pdf
    https://www.sparkfun.com/products/11856
Appendix C: Pictures of RC Car
Appendix D: Arduino Embedded Software

// Arduino Bluetooth Controlled by iOS for RC Car
// setting up arduino as BLE peripheral to be controlled by central device (phone)

//------------------BLUETOOTH HEADERS/GLOBAL VARS---------------------
#include <ArduinoBLE.h>

BLEDevice phone_central;

BLEService motorService("1111"); //create service for phone to read
BLEIntCharacteristic leftMotorChar("1110", BLERead | BLEWrite |
BLEWriteWithoutResponse);
BLEIntCharacteristic rightSteering("1112", BLERead | BLEWrite |
BLEWriteWithoutResponse);

int LED = 13; // Most Arduino boards have an onboard LED on pin 13

//------------------MOVEMENT HEADERS/GLOBAL VARS---------------------

//-------- WHEEL PIN SELECTION --------
//only ~ on data sheet means PWM possible, make sure to use two on different PA/PB timers
//pull one high/pwm and one low for that direction
const int rightMotorForward = 2; //D02 -> pos out1 //yellow-n-orange
const int rightMotorBackward = 3; //D03 -> neg out1
const int leftMotorForward = 16; //D16 -> pos out2 //green-n-blue
const int leftMotorBackward = 17; //D17 -> neg out2

//NOTE: When 16 and 9/17 and 10 in sync, lots of issues. but 16 and 10? No issues.
//throw in 25ms delay between onemotor, 16 and 9 positive in sync, but then 17 and 10 issues

//-------- MOVEMENT VARIABLES --------
const int deadzone_percent = 17; //deadzone for steering AND throttle
const int maxSteerPossible = 100;
const int maxSteeringRatio = (maxSteerPossible - 2) - deadzone_percent; // -100 to 100, if near max hardcodes deadzone limit
const int minSteeringRatio = deadzone_percent + 2;

const int maxAnalogDutyCycle = 255; // 0 to 255 for analogWrite forward, -1 to -255 for reverse
const int throttle_deadzone = (255*deadzone_percent)/100; // same % deadzone for steering and throttle

int steering_ratio = 0; // -100 to 100 (left to right)
int speed_input = 0; // -255 to 255, backwards to forwards
int left_speed = 0; // -255 to 255, backwards to forwards
int right_speed = 0; // -255 to 255, backwards to forwards

//-------- MOVEMENT FUNCTIONS ----------------

void moveMotor(int speed_ratio, int motor_forward_pin, int motor_backward_pin)
{
  //positive speed_ratio is forward, negative is backward.
  //motor_forward_pin and motor_backward_pin dictates which motor it is
  //refer to init at top for pin_number
  //forward pin and backward pin should stay same in main
  //direction dictated by speed_ratio
  if (speed_ratio > throttle_deadzone)
  {
    //forward

    if (speed_ratio > (maxAnalogDutyCycle - throttle_deadzone))
    {
      //full speed, timer doesn't do 0-10% and 90-100% effectively
      analogWrite(motor_forward_pin, maxAnalogDutyCycle);
      digitalWrite(motor_forward_pin, HIGH);
      //NOTE!!! using digitalWrite sometimes messes with PWM pins
      // and leaves it stuck at last known PWM, use analog max and 0
      analogWrite(motor_backward_pin, 0);
    }
  }
}
else
{
  //not in top, 0, or bottom deadzones
  analogWrite(motor_forward_pin, speed_ratio); //voltage difference in
  forward direction
  analogWrite(motor_backward_pin, 0);
}
//digitalWrite(motor_backward_pin, LOW);
}
else if (speed_ratio < (0-throttle_deadzone)) //may need to be 0-deadzone
{
  //backward

  if (speed_ratio < (throttle_deadzone - maxAnalogDutyCycle))
  {
    //full speed backward
    analogWrite(motor_backward_pin, maxAnalogDutyCycle);
    //digitalWrite(motor_backward_pin, HIGH);
    analogWrite(motor_forward_pin, 0);
  }
  else
  {
    //not in top, 0, or bottom deadzones
    analogWrite(motor_backward_pin, (abs(speed_ratio)) ); //voltage difference
    in backward direction
    analogWrite(motor_forward_pin, 0);
  }
  //digitalWrite(motor_forward_pin, LOW);
}
else
{
  //in 0 deadzone so stop
  analogWrite(motor_forward_pin, 0);
  analogWrite(motor_backward_pin, 0);
  //digitalWrite(motor_forward_pin, LOW);
  //digitalWrite(motor_backward_pin, LOW);
}
//Serial.print(";   PWM-byte: ");
//Serial.println(speed_ratio);
void stopAllMotors()
{
    // turn off all motors, no V-diff for no movement
    analogWrite(rightMotorForward, 0);
    analogWrite(rightMotorBackward, 0);
    analogWrite(leftMotorForward, 0);
    analogWrite(leftMotorBackward, 0);
}

int turningSpeed(int max_speed, int turning_ratio, bool left_or_right)
{
    // left_or_right is false(left) or true(right)
    // turning_ratio is -100(left) to 100(right)
    // max_speed is "max speed" possible
    int output_speed = max_speed;

    // limits turning_ratio if out of bounds
    if(turning_ratio > maxSteeringRatio)
    {
        turning_ratio = maxSteeringRatio;  
    }
    else if(turning_ratio < (0-maxSteeringRatio))
    {
        turning_ratio = 0-maxSteeringRatio;  
    }

    if(left_or_right) // right motors
    {
        // if(turning_ratio > deadzone_percent)
        // turning right, right side, so "max speed" is passed through
        if(turning_ratio < (0-deadzone_percent)) // turning left, right side
        {
            output_speed = maxSteerPossible - abs(turning_ratio);
            // turning left w/ 80% turning_ratio means only 20% right
            // 100-(80%) = 20% for next calculation.
            output_speed = (max_speed*output_speed)/maxSteerPossible;
        }
    }
    else // left motors
    {
    }
if(turning_ratio > deadzone_percent) //turning right, left side
{
    output_speed = maxSteerPossible - abs(turning_ratio);
    //turning right w/ 80% turning_ratio means only 20% left
    //100-(80%) = 20% for next calculation.
    output_speed = (max_speed*output_speed)/maxSteerPossible;
}
//if(turning_ratio < (0-deadzone_percent))
    //turning left, left side, so "max speed" is passed through

return(output_speed); //output_speed returns same input speed if no change or in deadzone

//------------------------------------------------------------------------------ SETUP AND RUNTIME --

void setup() // Called only once per startup
{
    //-------BLUETOOTH SETUP START------
    Serial.begin(9600); // Setup the serial port at 9600 bps, default baud rate
    //while(!Serial); //waits to establish serial connection with computer, don't use if on battery
    pinMode(LED, OUTPUT); // Set pin as an output to show connected
    //Serial.println("Starting bluetooth");
    
    if (!BLE.begin()) //start up bluetooth
    {
        //Serial.println("couldn't start BLE");
        while(1); //needs to be reset
    }
    BLE.setLocalName("RC Arduino"); //set name of local device (peripheral)
    BLE.setAdvertisedService(motorService); //set UUID for the service

    //add motor values to service (service is bulletin for others to check/change)
    motorService.addCharacteristic(leftMotorChar);
motorService.addCharacteristic(rightSteering);
BLE.addService(motorService);

leftMotorChar.writeValue(0); //initialize no motor movement
rightSteering.writeValue(0); //initialize no steering

BLE.advertise(); //put on bulletin board
digitalWrite(LED, LOW); //LED off to show not connected

//------BLUETOOTH SETUP END------
//------MOVEMENT SETUP START------
Serial.println("Deadzones: ");
Serial.println((maxAnalogDutyCycle - throttle_deadzone));
Serial.println(throttle_deadzone);
Serial.println(throttle_deadzone - maxAnalogDutyCycle);

pinMode(rightMotorForward, OUTPUT); // output pin setup
pinMode(rightMotorBackward, OUTPUT);
pinMode(leftMotorForward, OUTPUT);
pinMode(leftMotorBackward, OUTPUT);
//Serial.begin(9600); //for printing, done in bluetooth setup
stopAllMotors(); //make sure car is off and not moving

//------MOVEMENT SETUP END------

}

void loop() // Continuous loop
{

  phone_central = BLE.central(); //try to connect to a central

  if (phone_central)
  {
    //phone connected!
    //Serial.print("Connected Central Address: ");
    //Serial.println(phone_central.address());
    digitalWrite(LED, HIGH); //LED on to show connected

    while(phone_central.connected()) //continue while connected
    {

if (leftMotorChar.written() || rightSteering.written())
{

    //motor vals have been updated
    speed_input = leftMotorChar.value();
    //Serial.print("Speed: "); Serial.print(speed_input);
    steering_ratio = rightSteering.value();
    //Serial.print(";   Steer: "); Serial.println(steering_ratio);

    //
    left_speed = turningSpeed(speed_input, steering_ratio, false); //false is left
    right_speed = turningSpeed(speed_input, steering_ratio, true); //true is right
    //Serial.print("Speed (Left): "); Serial.print(left_speed);
    //Serial.print(";   Speed (Right): "); Serial.println(right_speed);
}
//delay(75); //wait 75ms before changing
moveMotor(left_speed, leftMotorForward, leftMotorBackward);
moveMotor(right_speed, rightMotorForward, rightMotorBackward);

//Serial.println("Disconnecting from device");

left_speed = 0;
right_speed = 0;
steering_ratio = 0;
stopAllMotors(); //stop if disconnected so car doesn't drive away
digitalWrite(LED, LOW); //LED off to show disconnected
//BLE.poll(); //update BLE events
Appendix E: Apple Swift Software

-----Info.plist-----------------------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>UISupportedInterfaceOrientations~iphone</key>
  <array>
    <string>UIInterfaceOrientationLandscapeLeft</string>
    <string>UIInterfaceOrientationLandscapeRight</string>
  </array>
  <key>UIRequiredDeviceCapabilities</key>
  <array>
    <string>bluetooth-le</string>
  </array>
  <key>UIRequiresFullScreen</key>
  <true/>
  <key>UISupportedInterfaceOrientations</key>
  <array>
    <string>UIInterfaceOrientationLandscapeLeft</string>
    <string>UIInterfaceOrientationLandscapeRight</string>
  </array>
  <key>NSBluetoothAlwaysUsageDescription</key>
  <string>Bluetooth to connect to RC car for control</string>
  <key>NSBluetoothPeripheralUsageDescription</key>
  <string>Bluetooth to connect to RC car for control</string>
  <key>CFBundleIcons</key>
  <dict/>
  <key>CFBundleIcons~ipad</key>
  <dict/>
  <key>UIApplicationSceneManifest</key>
  <dict>
    <key>UIApplicationSupportsMultipleScenes</key>
    <false/>
    <key>UISceneConfigurations</key>
    <dict>
      <key>UIApplicationSupportsMultipleScenes</key>
      <false/>
      <key>UISceneConfigurations</key>
      <dict>
        
    </dict>
  </dict>
<key>UIWindowSceneSessionRoleApplication</key>
<array>
  <dict>
    <key>UISceneConfigurationName</key>
    <string>Default Configuration</string>
    <key>UISceneDelegateClassName</key>
    <string>$(PRODUCT_MODULE_NAME).SceneDelegate</string>
    <key>UISceneStoryboardFile</key>
    <string>Main</string>
  </dict>
</array>
</dict>
</dict>
</dict>
</plist>
import UIKit
import CoreBluetooth

class ViewController: UIViewController, CBCentralManagerDelegate, CBPeripheralDelegate {

    var manager:CBCentralManager!
    var peripheral:CBPeripheral!

    let peripheral_NAME = "RC Arduino"
    let motorService_UUID = CBUUID(string: "1111") //"1111"
    let leftMotorChar_UUID = CBUUID(string: "1110") //"1110"
    let rightSteer_UUID = CBUUID(string: "1112") //"1112"

    func centralManagerDidUpdateState(_ central: CBCentralManager) {
        // starts scanning for Arduino if bluetooth enabled and in Central state
        if central.state == CBManagerState.poweredOn {
            central.scanForPeripherals(withServices: [motorService_UUID])
        } else {
            print("Arduino Not Found via Bluetooth")
        }
    }

    func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber) {
        //if peripheral discovered, connect if desired local name
        let potential_periph = (advertisementData as NSDictionary).object(forKey: CBAAdvertisementDataServiceLocalNameKey) as? NSString
        if (potential_periph?.contains(peripheral_NAME) == true){
            // connect to the peripheral
            peripheral = peripheral!
            manager = manager!
            print("Connected to peripheral")
            // connect to the service
            let service = manager!
            let leftMotorChar = service!
            let char Num = leftMotorChar!
            // connect to the character
            let char = char Num!
            let charNum: UInt8 = char!
            // send data to the character
            char.writeData(to: data: [100, 200], isFormat: false)
        }
    }
}

fun centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber) {
    //if peripheral discovered, connect if desired local name
    let potential_periph = (advertisementData as NSDictionary).object(forKey: CBAAdvertisementDataServiceLocalNameKey) as? NSString
    if (potential_periph?.contains(peripheral_NAME) == true){
        // connect to the peripheral
        peripheral = peripheral!
        manager = manager!
        print("Connected to peripheral")
        // connect to the service
        let service = manager!
        let leftMotorChar = service!
        let char Num = leftMotorChar!
        // connect to the character
        let char = char Num!
        let charNum: UInt8 = char!
        // send data to the character
        char.writeData(to: data: [100, 200], isFormat: false)
    }
}
// if device name matches, connect to this peripheral
self.manager.stopScan() // stop looking for others
self.peripheral = peripheral
peripheral.delegate = self
manager.connect(peripheral) // connect to arduino
print("connecting to arduino successful!")
// peripheral.discoverServices([motorService_UUID])
// needs to wait for connection
}

func centralManager(_ central: CBCentralManager, didConnect peripheral: CBPeripheral) {
    peripheral.discoverServices(nil)
}

func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error: Error?) {
    // get service(s) offered by device
    print("checking for services...")
    for service in peripheral.services! {
        let this_service = service as CBService
        if (this_service.uuid == motorService_UUID) {
            print("found the right service! Going to check for characteristics...")
            peripheral.discoverCharacteristics(nil, for: this_service)
        } // if it matches the arduino service is discovered, get the characteristics
    }
}

var motorCharsPresent : Bool = false; // if left and right motor chars available
var leftMotorCharacteristic : CBCharacteristic? = nil;
var rightMotorCharacteristic : CBCharacteristic? = nil;

func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService, error: Error?) {
    // check to see if motor characteristics present
    var leftMotor = false;
    var rightMotor = false;
    print("Entered discovering characteristics")
    for current_characteristic in service.characteristics! {
        let this_char = current_characteristic as CBCharacteristic
        print(current_characteristic.uuid, "; check if correct uuid")
if this_char.uuid == leftMotorChar_UUID
{
    print("left confirmed")
    leftMotor = true;
    leftMotorCharacteristic = current_characteristic
}
if this_char.uuid == rightSteer_UUID
{
    print("right confirmed")
    rightMotor = true;
    rightMotorCharacteristic = current_characteristic
}
}
motorCharsPresent = leftMotor && rightMotor
print("motorChars: ", leftMotor, rightMotor)
}

func centralManager(_: CBCentralManager, didDisconnectPeripheral peripheral: CBPeripheral, error: Error?) {
    //if disconnected, start rescanning again
    print("Disconnected, rescanning")
motorCharsPresent = false;
    central.scanForPeripherals(withServices: [motorService_UUID])
        //put in services for reconnecting to specific kind, nil if just general
}
var leftSpeed : Int = 0
var rightSteer : Int = 0

func updateMotors()
{
    //only opens if both motor characteristics present and connected
    
    //Data.init(base64Encoded: Data(leftSpeed))
    let dataL = NSMutableData.init(capacity: 0) //
    let dataR = NSMutableData.init(capacity: 0) //
    dataL?.append(&leftSpeed, length: 4)
    dataR?.append(&rightSteer, length: 4)
    print("dataL: ", dataL as Any, ";  dataR: ", dataR as Any)

    if (leftMotorCharacteristic != nil && rightMotorCharacteristic != nil)
    {
        print("\nWriting Values")
    peripheral.writeValue(dataL! as Data, for: leftMotorCharacteristic!, type: .withoutResponse)
    peripheral.writeValue(dataR! as Data, for: rightMotorCharacteristic!, type: .withoutResponse)
}

private let screenSize : CGRect = UIScreen.main.bounds //const var is let

private var throttleBoxView : UIView = {
    let throttleBoxView = UIView()
    throttleBoxView.contentMode = .scaleAspectFill
    throttleBoxView.backgroundColor = .systemYellow
    throttleBoxView.autoresizesSubviews = true
    return throttleBoxView
}()}

private var throttleView : UIView = {
    let throttleView = UIView()
    throttleView.contentMode = .scaleAspectFill
    throttleView.backgroundColor = .systemBlue
    throttleView.autoresizesSubviews = true

    return throttleView
}()

private var steeringBoxView : UIView = {
    let steeringBoxView = UIView()
    steeringBoxView.contentMode = .scaleAspectFill
    steeringBoxView.backgroundColor = .systemYellow
    steeringBoxView.autoresizesSubviews = true

    return steeringBoxView
}()

private var steeringView : UIView = {
    let steeringView = UIView()
    steeringView.contentMode = .scaleAspectFill
    steeringView.backgroundColor = .systemBlue
    steeringView.autoresizesSubviews = true
private var throttleCenter = CGPoint(x: 0, y: 0)
private var steeringCenter = CGPoint(x: 0, y: 0)
private var userPosition = CGPoint(x: 0, y: 0)

@objc func throttle_drag(sender: UIPanGestureRecognizer){
    switch sender.state{
    case .changed:
        userPosition = sender.location(in: view)

        if(self.throttleBoxView.frame.contains(userPosition)) { //move in throttle
            UIView.animate(withDuration: 0.15, delay: 0, options: .curveEaseInOut,
                           animations: { self.throttleView.center.y = self.userPosition.y }
             ) //movement in 1/4 sec, 0s delay,
        }

    if motorCharsPresent {
        //speed is left, -255 to 255 (left to right)
        //positive val is turn right, negative val is turn left
        //throttleBoxView is outside frame/bounds, throttleView is joystick
        let left_motor = self.throttleBoxView.center.y - self.throttleView.center.y
        //print(right_motor, "", self.steeringView, " centerx: ",
        self.steeringBoxView.center.x)
        letSpeed = Int(
            (left_motor*2*255) / self.throttleBoxView.frame.height 
        )

            updateMotors();
        }

    default:
        break
    }
}

@objc func steering_drag(sender: UIPanGestureRecogni

return steeringView
}()
switch sender.state{
    case .changed:
        userPosition = sender.location(in: view)

        if (self.steeringBoxView.frame.contains(userPosition)) {
            // move in steering
            UIView.animate(withDuration: 0.15, delay: 0, options: .curveEaseInOut, animations: {
                self.steeringView.center.x = self.userPosition.x
            })
            // movement in 1/4 sec, 0s delay,
        }

        if motorCharsPresent {
            // steering is right, -100 to 100 (left to right)
            // positive val is turn right, negative val is turn left
            let right_steering = self.steeringView.center.x - self.steeringBoxView.center.x
            // print(right_steering, "", self.steeringView, " centerx: ", self.steeringBoxView.center.x)
            rightSteer = Int((right_steering * 2 * 100) / self.steeringBoxView.frame.width)
            updateMotors();
        }

    default:
        break
}

// Set the shouldAutorotate to False
override open var shouldAutorotate: Bool {
    return true
}

// set orientation
override open var supportedInterfaceOrientations: UIInterfaceOrientationMask{
    return .landscape
}

// override func viewWillAppear(_ animated: Bool) {
//    super.viewWillAppear(animated)
//    //view.layoutSubviews()
func setJoystickCenters(throttleBoxView: UIView, steeringBoxView: UIView, throttleView: UIView, steeringView: UIView){

    throttleBoxView.frame = CGRect(x: 0, y: 0, width: 125, height: 290)
    throttleBoxView.center = view.center
    throttleBoxView.center.x -= 5*screenSize.midX/8 //center x at 3/16 screen
    //throttleBoxView.center.y += screenSize.midY/4 //center y at 5/8 screen
    self.throttleCenter = throttleBoxView.center

    throttleView.frame = CGRect(x: 0, y: 0, width: 80, height: 20)
    throttleView.center = self.throttleCenter

    steeringBoxView.frame = CGRect(x: 0, y: 0, width: 290, height: 125)
    steeringBoxView.center = view.center
    steeringBoxView.center.x += screenSize.midX/2 //center x at 3/4 screen
    //steeringBoxView.center.y += screenSize.midY/4 //center y at 5/8 screen
    self.steeringCenter = steeringBoxView.center

    steeringView.frame = CGRect(x: 0, y: 0, width: 20, height: 80)
    steeringView.center = self.steeringCenter
}

override func viewDidLoad() { super.viewDidLoad()
    manager = CBCentralManager(delegate: self, queue: nil)
    //establish view controller as Central Device Management

    //timer = Timer.scheduledTimer(withTimeInterval: 0.05, repeats: true, block: { _ in self.updateMotors() }) //should update motors every 50ms (1 is 1 second)

    let value = UIInterfaceOrientation.landscapeLeft.rawValue
    UIDevice.current.setValue(value, forKey: "orientation")
    //viewWillLayoutSubviews()
// view.layoutSubviews()

view.backgroundColor = .systemGreen

// x is short side in portrait (width), y is long side in portrait (height)
view.addSubview(throttleBoxView)
view.addSubview(steeringBoxView)
view.addSubview(throttleView)
view.addSubview(steeringView)

// set colors and centers
view.isMultipleTouchEnabled = true;
let throttlePan = UIPanGesture Recognizer(target: self, action:
#selector(throttle_drag))
let steeringPan = UIPanGesture Recognizer(target: self, action:
#selector(steering_drag))

throttlePan.maximumNumberOfTouches = 10
steeringPan.maximumNumberOfTouches = 10

throttleBoxView.addGestureRecognizer(throttlePan)
steeringBoxView.addGestureRecognizer(steeringPan)

setJoystickCenters(throttleBoxView: throttleBoxView, steeringBoxView:
steeringBoxView, throttleView: throttleView, steeringView: steeringView)

// steeringBoxView.addGestureRecognizer(
//    UIPanGesture Recognizer(target: self, action: #selector(steering_drag))
//)
// updateMotors()