KENNESAW STATE UNIVERSITY

CPE 4850 - Senior Project Design, Fall 2023

Final Project Report

MightyMover

CacheMoney

Grant Burke, Hanson Chaney, Cooper Newlin, Chris Turner

Date: December 8, 2023

Electrical and Computer Engineering

Kennesaw State University

Faculty: Dr. Jeffrey L Yiin

Table of Contents

I. Project Summary

Join MightyMover in revolutionizing the autonomous market with a seamless internet voided experience. Our goal is to create a product that will assist its users in pushing and pulling heavy loads. All components operate from one button, a tag, and a Bluetooth connection from our mobile app. With these, you can walk, and the mover will follow.

Our autonomously following mover is designed to follow you anywhere. It operates on object detection, Bluetooth, and a rechargeable 9.3V lithium-ion battery. Along with the app, the user can access data such as Bluetooth strength, servo angle and Mover-stop functions. This design is simplistic to allow a large audience ease of access without the need to read a large manual or quick start guide to begin moving. Simply press the power on the electronic speed controller and wait for successful connection on the app, the mover will do the rest.

Market Analysis

Our product has multiple use cases in various scenarios such as personal gardening, outdoor landscaping, farming, textile or warehouse industries, we move it all! The MightyMover will save you time and reduce stress. You will be able to move things around your property efficiently without risking injury. The MightyMover allows all to work without outside help, those who may be physically challenged will no longer need a second hand to assist them in doing the things they love. This product may be used by all ages as a safer alternative to manual hauling.

Competition would consist of wagons, push carts, and dollies. These all require pushing or pulling which can be strenuous on the user. There are also electric pallet jacks and similar products, but these are mostly used in commercial settings which are not our

focused market. Other competitors closely related are the food delivery robot (Hall et al., 2022) and heavy-duty push carts, but these are a niche market for food service and personal use. From what we have researched, our product is the first of its kind to target a consumer market dedicated to outside use.

II. Product Description

The MightyMover makes use of 4 major features for its operation: steering, object detection, a battery management system, and a mobile app. The user can simply turn on the power button and the mover will begin its operation.

Steering Sub System

The steering system makes use of a Bluetooth module that provides accurate angle readings. This was chosen because it is highly accurate and dependable. With the angle measurements from the module, the mover is able to correctly orient itself towards the user.

Object Detection Sub System

The object detection system utilizes an artificial intelligence model to accurately depict people via a camera. The mover will only proceed towards the user if it is sufficiently far away, and there are no other people or objects in the way. Using this method prevents the mover from colliding with the user and causing injury.

Battery Management Sub System

The battery management system, made with a custom PCB and a buck converter, has multiple functions. Firstly, it provides a safe amount of power to the components in the mover. This ensures that no components in the mover catch fire or

stop functioning unexpectedly. Secondly, it indicates to the user that the mover's battery is sufficiently charged via a green LED, or in need of charging via a red LED.

Mobile Application Sub System

The mobile app is an optional monitoring tool that displays things like angle and signal strength. It makes use of Bluetooth, so the user can connect to the mover at any location, regardless of Wi-Fi availability. It also contains an emergency stop function via the 'Disconnect' button, which allows the user to stop the mover immediately if something goes wrong.

Chassis Sub System

Additionally, the mover has a durable, weather-proof encasement that allows it to be used indoors or outdoors. It contains a space on top for placing objects, so it can haul objects for the user.

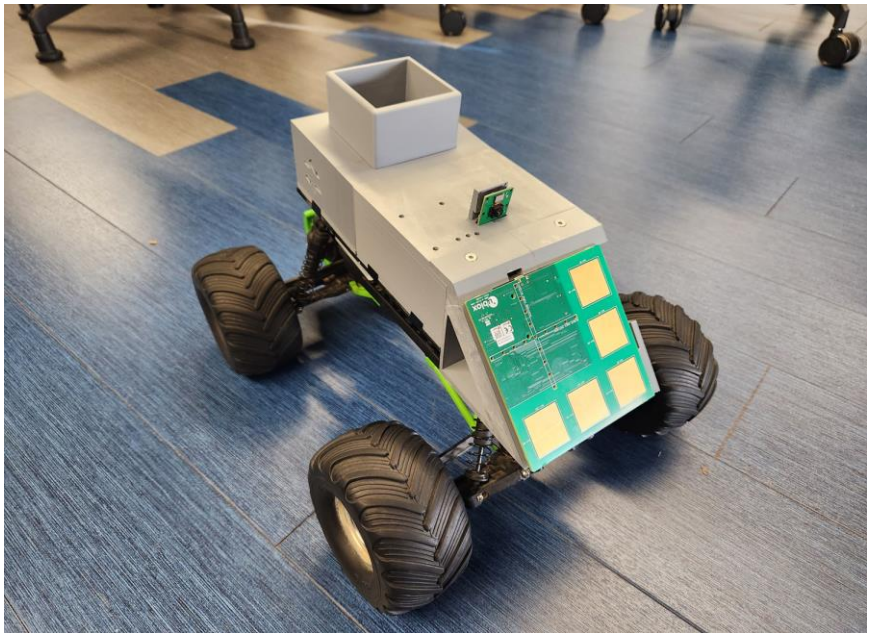

Figure 1 – The MightyMover

III. Hardware System Design

Raspberry Pi #1 and Bluetooth Module

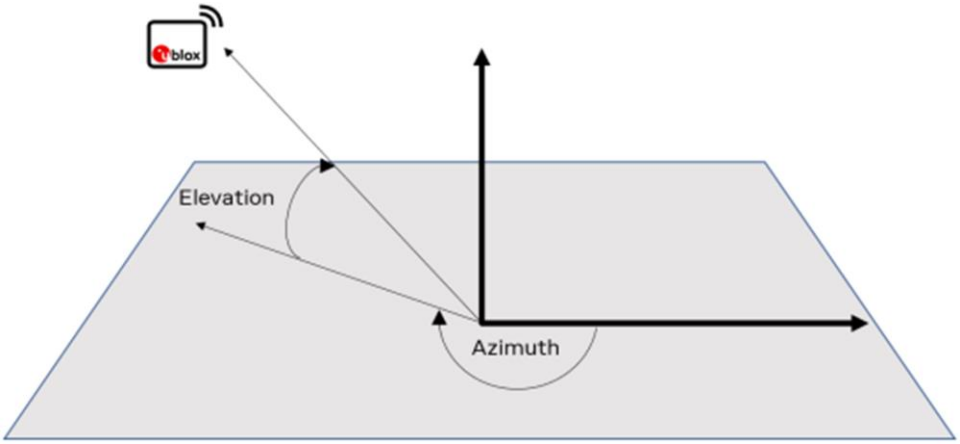

Figure 2 – Bluetooth Angle Finding

The Bluetooth anchor node uses Bluetooth Direction Finding technology to calculate a relative angle between itself and any Bluetooth tag in its range. This is done by calculating the phase shift of the incoming signal from the tag, between each of the anchor's five antennas. As the antennas are laid out on two axes, this angle can be reported in two directions, referred to as the Azimuth and Elevation angles. For our purposes, only the Azimuth angle will be used.

The anchor delivers the angle data to a listening host over either USB or the UART pins. For development purposes, we used the USB connection, although in a future design it would be more practical to use the UART connector.

Battery Management PCB

The PCB accomplished the main functions that we needed for this project. One main key factor of the PCB board was to distribute power to the whole system when a button on the ESC was pushed to start the mover. This allowed an easy to start method for people to use our project without having to calibrate or do multiple functions. The second goal that our PCB accomplished was to show if the mover was on. Another element that we added to the mover was to tell us if the battery needed charging. A green light was shown that the mover was on, and a red light was to be the low battery

indicator. This was accomplished through LEDs, resistors, transistors, connectors, and an op amp.

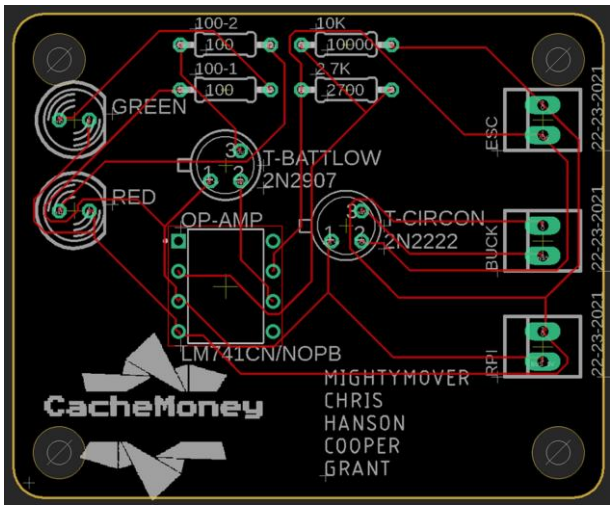

Figure 3 – PCB Design

IV. Encasing Design

Our goals for the 3D design were to enclose all of the electronics, other than the Bluetooth anchor node which needed to be exposed, and to possess a compartment for carrying a small object on the back.

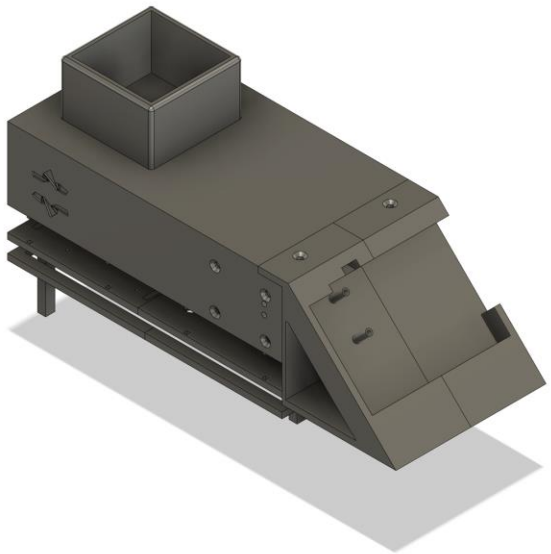

Figure 4 – 3D Render of Chassis

It also provides mounting holes for the electronics on the base of the chassis and for the PCB on the top. The individual components all connect either by screws or permanently attached with epoxy glue. This was due to limitations in the size of our 3D printer, if we had access to a larger printer, the top and bottom pieces could have been printed as a total of two components.

The front of the mover chassis has an angled surface for mounting the Bluetooth anchor node with screws. It was positioned at an angle to best enable accurately detecting the location of the user since it should be pointed towards the middle of their body in most cases.

V. Software System Design

Bluetooth Angle Finding

The script for interfacing the Bluetooth Direction Finding module with the Raspberry Pi had the responsibility of receiving the angle data from the anchor node and reporting those values to the Arduino to control the steering servo as well as to the mobile app to be displayed on the screen. This was implemented by writing a simple Python script which read from the serial port corresponding to the anchor node, parsed the incoming message for the angle values, and wrote the angles back out of the other serial port corresponding to the Arduino.

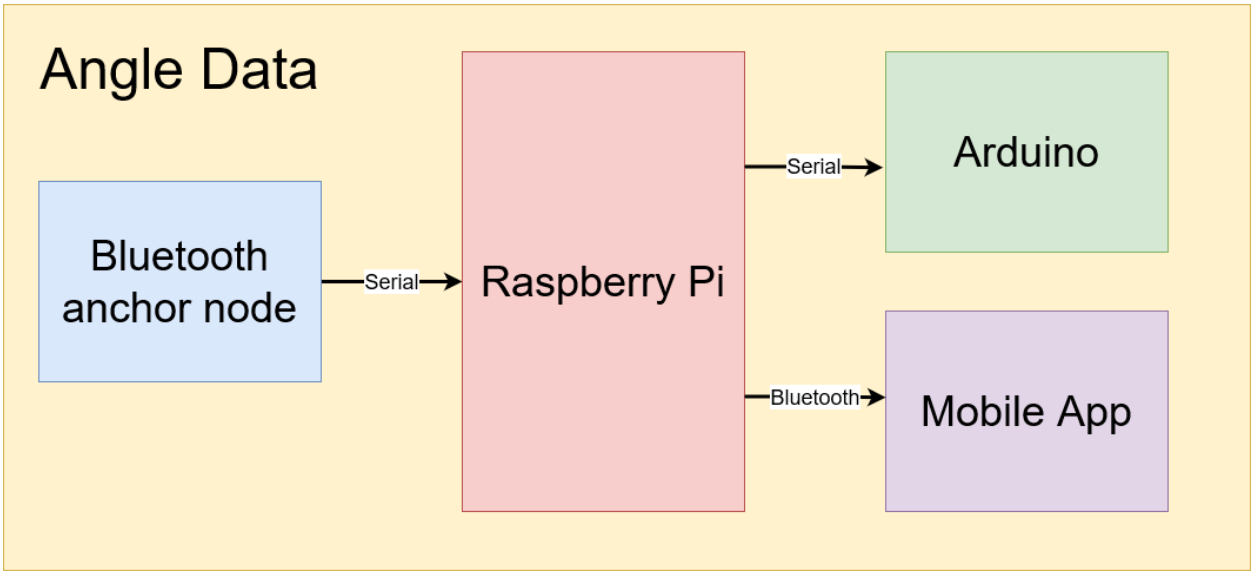

Figure 5 – System Communication

Object Detection

Our object detection system had two functions: track the person in front of the "RC" car and detect if there was an object in between the person and our project. The implementation of this system included taking a pre-trained model named Common Objects in Context (COCO) model. Two manipulations that we did for the raspberry pi was to use a TensorFlow lite model, so the raspberry pi would not have trouble running AI models. We also limited the number of labels that the camera needed to detect using if statements in the implementation code.

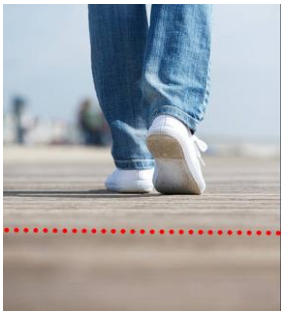

Figure 6 – Detection Threshold

The only scenario that the project would move if it detected a person by their legs and if the person was far away. Instead of having to find the person's distance or depth on the camera, we drew an imaginary line on the picture and if the person's detected

border was over this threshold, then the car was able to move forward. Otherwise, the car would stop. Also, if the camera detected an object such as a dog, chair, or another person then the car would stop.

Arduino Code

The Arduino code had the purpose of receiving angle data from one raspberry pi (Bluetooth module and app) and GPIO output from the other raspberry pi (object detection) to output a PWM signal to the steering servo based on the received angle data. It also outputs a PWM signal to the RC car's ESC to control the movement of the rear motors based on the object detection code. The ESC also needed calibration which the microcontroller handled.

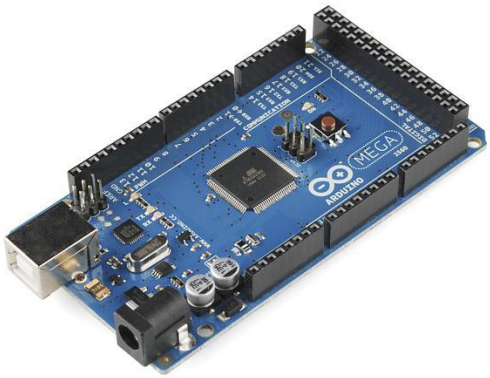

Figure 7 – Arduino MEGA 2560

App Design and Communication

The app was created in MIT App Inventor, a block-based app creation tool. It allows quick creation of decent looking apps, but also lacks some functionality like fast processing. Apart from basic design elements like buttons and labels, the major function of the app was to communicate via Bluetooth. MIT App Inventor has built in Bluetooth functionality, and it was very easy to work with.

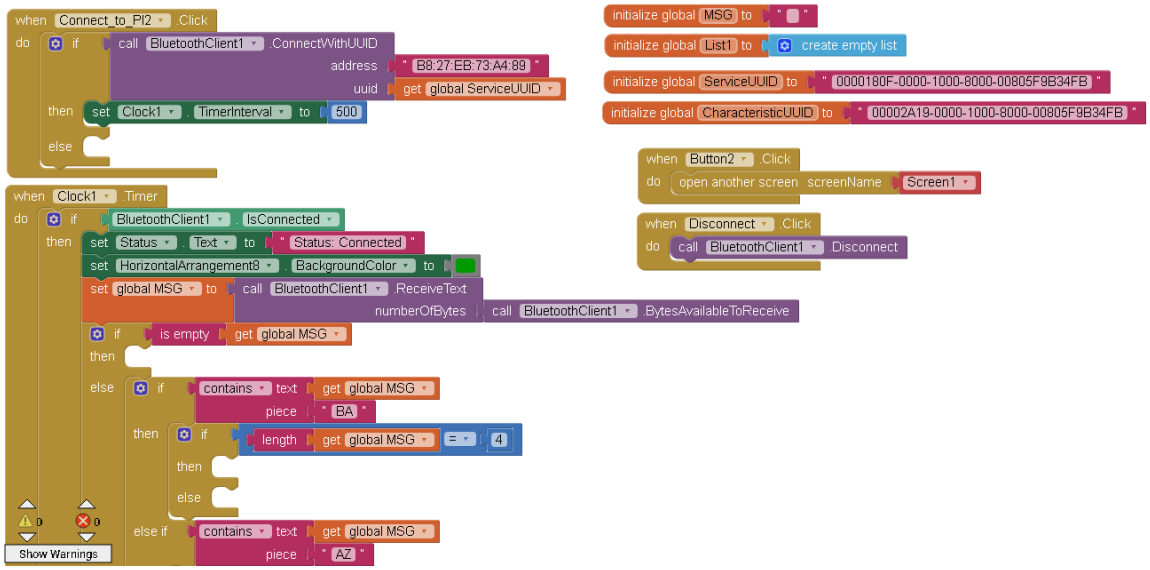

Figure 8 – App Blocks

On the Raspberry Pi, a script that establishes connection with the app is executed. The app uses a specific UUID that is replicated on the Pi to establish connection. Once the connection is established, every second specific data is sent to the app so it can be displayed. This script required several libraries to be executed, namely PyBluez and Sockets. It imports the angle readings and signal strength readings from a separate script used for the Bluetooth angle finding.

Both this script and the angle finding script are designed to run on startup, through the use of Cron. This way, the user does not have to tinker with code. Within 60 seconds of the Pi being powered from the battery, both scripts are functional, and the app can be connected to the Pi to receive data.

```
pi@cnewlin:~/seniorproject $ sudo python test3.py
Waiting for a connection on RFCOMM channel 1...
Accepted connection from ('98:09:CF:92:D3:CD', 1)
Accepted connection from ('98:09:CF:92:D3:CD', 1)
Sent data: BA80
Sent data: AN50
Sent data: -50
```

Figure 9 – Bluetooth Script Execution

VI. User Application Design

Initial Concept

The first concept for the app mimicked the function of the Bluetooth tag the mover currently uses. It was supposed to provide the user's location via GPS, so the mover could navigate towards them. Additionally, it would provide the mover's location to the user, so if the user lost the device, they could still locate it on the app.

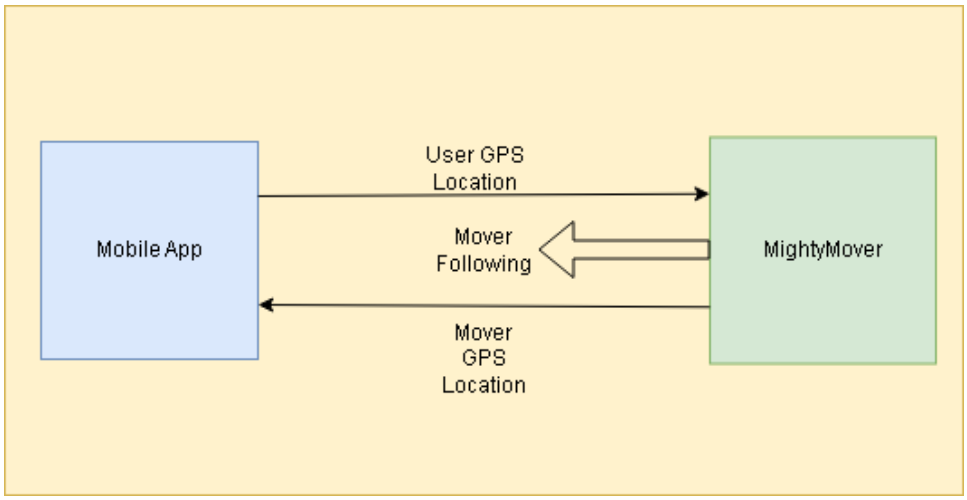

Figure 10 – Initial App Diagram

Initial Function

For the app to provide the user's location to the mover, it had to be able to receive the GPS location of the mobile device. There are several ways to accomplish this for an independent application, but the easiest is to make use of Google's free API, Fused Location Provider. Implementing this into the application provided very accurate GPS readings.

For the Mover's GPS, a proprietary GPS module was used. It provided semi-accurate GPS readings, and those readings could be transferred into a device capable of transmitting the data, such as a Raspberry Pi.

With these two devices providing GPS location, the data would be sent bi-directionally via Bluetooth. Both the Raspberry Pi and mobile device have Bluetooth capability. With all these features together, a circular system of transmitting and receiving GPS locations for the Mover and app to use could be created.

First Design

The first design of the app was a simple interface that provided the user a way to see the various data points being collected from the API. It also included a map button, so that the user could see their location on a map of the earth, but this idea wasn't developed further because it costs money to use the google maps API. The design wasn't overly user friendly but was functional.

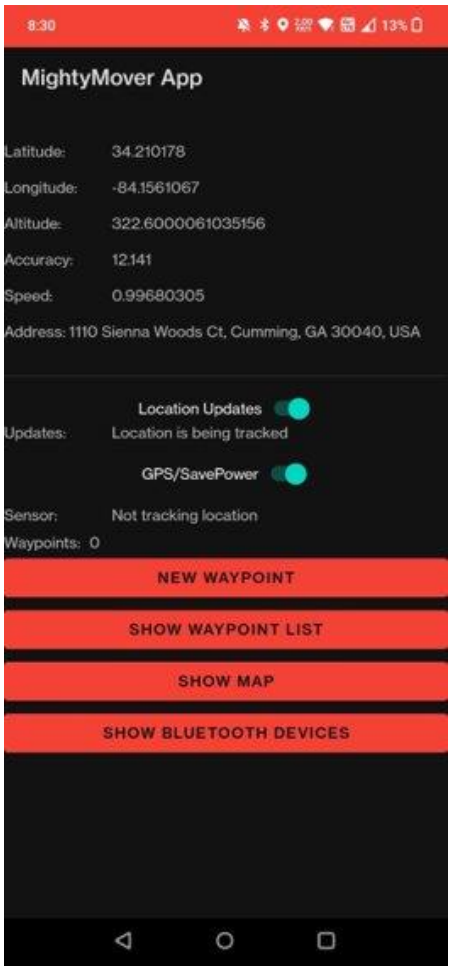

Figure 11 – Mobile App Home Page

As seen in the image above, the Google API is able to provide accurate latitude and longitude, as well as altitude. The app also had a power saving mode, because constantly polling GPS consumes energy. The waypoint buttons were unused but could have been a feature that allowed the user to define their own path for the mover to travel on. The ‘Show Map’ button allows the user to view their location on a map of the earth.

Figure 12 – Map Screen

Lastly, the initial design contained a screen for connecting Bluetooth devices. This would allow the transmission of GPS data back and forth.

Figure 13 – Devices Screen

Initial Results

The accuracy of the GPS readings on the mobile device were accurate, but the proprietary module that was used on the mover was providing wildly inaccurate and slow readings. This led us to look for another solution. The maps API costs $2 per 1000 requests, so that was an additional cost for a feature that wasn't necessary. Additionally, this app was created in Android Studio, which had a massive learning curve. It was simply too costly in terms of time to continue with this version of the mobile application, and a new approach was needed.

Pivoting

The new approach to the mover's navigation was to use a Bluetooth module and tag, which bypassed the need for connecting to a satellite to get GPS data. This also changed the function of the app, which was no longer a central tool for navigation. Instead, it became a monitoring tool, which could display various types of data provided by the mover. Things like angle, signal strength, connection security and more could be sent to the app for the user to read. Additionally, a new tool for app creation, called MIT app inventor, allowed for much easier app development and Bluetooth implementation. It also had a feature to sever the connection to the devices on the mover, so this could be used as an emergency stop feature.

Final Design

The final design included a home page, an additional page to display the data, as well as three buttons to go back to the home screen, connect to the mover via Bluetooth, or disconnect.

Figure 14 – Home Screen

Figure 15 – Connected

Figure 16 - Disconnected

Results

The final product is fully functional and easy to use. It provides the user with various types of data to monitor. Additionally, the emergency stop feature was quick and could stop the mover almost instantly. A limitation of MIT app inventor meant that the Bluetooth data was not able to be updated faster than every 1 second, but the user would probably not need it updated faster than that anyway. The app isn't essential to the functioning of the mover but contains helpful information and can be expanded upon as detailed in the future work section.

VII. Product Requirement Analysis

Table 1. Original Product Requirements

ID-Name	MOVER-SYS-001: Phone to MightyMover
Description	This sub system is the process of the phone and the MightyMover communicating. The user will have feedback from MightyMover showing the battery percentage, location monitoring, and the current mode of the Mover. The app will start with a mode selection which would be following or location mode. Then prompts will display with the user putting in further information to help MightyMover get instructions. The app shall always show an emergency stop button, so when the user sees a situation, they can wirelessly shut the Mover down.
Key Components	Mobile App, Bluetooth Module, Raspberry Pi
Interfaces	Xcode, Android Studio, Linux OS
Requirements	PRD-SYS-001 – Must have Bluetooth 5.0 Connectivity. PRD-SYS-002 – Must Receive Bluetooth Signal up to 10 meters away from module. PRD-SYS-003 – Must transfer data at a speed of at least 2 Mbit/s.
ID-Name	MOVER-SYS-002: MightyMover Motor Control System
Description	The motors determine where the MightyMover moves. The motors will be controlled by a Raspberry Pi PWM GPIO port. The Raspberry Pi will make decisions based on the user's phone location and all the sensors on the Mover.
Key Components	Raspberry Pi, LiDAR sensor, GPS module, mobile app, Bluetooth Module. Motors,
Interfaces	Linux OS, Speed Controller

Requirements	PRD-SYS-004 – Motor must consume no more than 12V. PRD-SYS-005 – Motor must possess 30 Watts of horsepower. PRD-SYS-006 – Motor must be smaller than 2x2x3in.
ID-Name	MOVER-SYS-003: MightyMover LED Control System
Description	The LEDs show which mode the cart is currently in along with the power on LED. These LEDs will be controlled by the Raspberry Pi which will get the mode it's currently in from the user's phone. This will help the user understand what the cart is doing and will help the team in debugging.
Key Components	LEDs, Raspberry Pi, Mobile App
Interfaces	Linux OS, Xcode, Android Studio, Raspberry Pi GPIO
Requirements	PRD-SYS-007- The LEDs must operate in accordance with the RGB color model. PRD-SYS-008- The LEDs shall be always on, even during motion. PRD-SYS-009- The LEDs shall use no more than 20mA of current.
ID-Name	MOVER-SYS-004: LiDAR Subsystem
Description	The LiDAR system will include a microcontroller to process the LiDAR data to convert into serial communication for the Raspberry Pi. We are currently looking into ways that we can use the LiDAR system to get more accurate location than a GPS module. This system is very important because it will help determine object detection, help create a path in memory, and make sure we are following the user. We are thinking that the microcontroller may also be able to run the object detection code that we initially have on the Raspberry Pi. For now, we plan to have the raspberry pi running the code for simplicity.
Key Components	LiDAR sensor, Raspberry Pi
Interfaces	Linux OS, Raspberry Pi GPIO

Requirements	PRD-SYS-010- The LiDAR sensor must be capable of sensing objects within 12 meters of the device. PRD-SYS-011- The LiDAR sensor must have a sample rate of 8000 Times. PRD-SYS-012- The LiDAR sensor must weigh no more than 1 pound.
ID-Name	MOVER-SYS-005: Software Hub
Description	The Raspberry Pi is the decision maker of the Mover. The RPi shall take in information from the mobile app via Bluetooth signal, GPS location, and LIDAR input from its microcontroller. The app data will be used to determine which mode that the Mover will be in as well as determine if the Mover needs to stop (emergency stop). The GPS location will be used to determine the next point that the Mover needs to move to. Finally, the LiDAR sub-system will run object detection and help with PWM outputs to the motor. These inputs will feed into many files when they are executed. The emergency stop, lost signal, and object detection command files will hold higher precedence over the location and following command file because these are the safety features to the Mover.
Key Components	Raspberry Pi, GPS Module, LiDAR Module, Bluetooth Module
Interfaces	Linux OS
Requirements	PRD-SYS-013- System must start all applications within one minute of turning on. PRD-SYS-014- System must accept all data sent from sensors. PRD-SYS-015- System must determine mode from app signal.
ID-Name	MOVER-SYS-006: Battery Management System
Description	The battery management system sends the power to mostly every component of the Mover. We are still in the process of picking parts out, so most of these materials are still being determined, but this is the idea so far. If things change to a low voltage system, we may need a different DC to DC converter.
Key Components	Battery, DC-DC Converter

Interfaces	Raspberry Pi
Requirements	PRD-SYS-016- Battery must have a capacity of 12 Amp Hours. PRD-SYS-017- Battery must weigh less than 4 pounds.

Due to pivoting in the design, some systems were eliminated/replaced, such as LiDAR and GPS being replaced by the Bluetooth Direction Finding module. The requirements for the phone-to-mover subsystem were met. The requirements for the motor ended up not being very relevant to our final design, however they were technically met. The power LEDs also fulfilled the requirements as written; however we would have liked to have a more informative indicator of the remaining battery level. The Software Hub was able to fulfill requirements PRD-SYS-014 and PRD-SYS-015, however it required the use of an additional Raspberry Pi to run the object detection code. Additionally, it took longer than one minute to start all the applications from bootup, so PRD-SYS-013 is a failure.

Finally, the battery management system: the specs for the battery surpassed the requirements due to the capacity being more than double what was required at ~28Ah and it also fell under the weight limit. It was not discussed in the original document, but it is relevant here, the battery management PCB did what we intended it to do by supplying power for a Raspberry Pi from the RC car's battery and providing the circuitry for the power LED. However, it was insufficient to power the additional electronics that were needed.

VIII. Lessons Learned

Many skills were learned, starting from our market analysis and ending with our presentation skills at the Senior Expo. We struggled as a team to finalize our project ideas in the early stages but soon we found a product we would all be proud to work on. Forming a market analysis and a product requirement document gave us a skillset we all never built in our early college careers. Deciding on our requirements meant making a budget with a bill of materials for each item needed for a successful proof of concept.

Once diving into hardware and software implementation, we soon learned the complexity of this project. Doctor Billy Kihei met with us early in the project lifecycle to discuss with us just how complex our project would become, and he was completely correct. Dropping GPS and LIDAR for Bluetooth and object detection was the best move to avoid any more setbacks, however we were still unsuccessful in a fully functioning mover that would work on one button start. Each component had its issues that caused a waterfall effect that caused implementation to be a headache, then once battery load was considered, we realized we had too many electronics for our single PCB designed for just one Pi and some accessories.

In terms of hardware, we learned the importance of diligent simulation work before sending designs for production; the PCB is a great example of that. Having a few tweaks to the LTSpice model and a second draft of the EagleCAD board was the best thing we could have done to avoid any issues with our battery circuit when implementing hardware. This project also taught the importance of taking your time when soldering on boards or troubleshooting wiring, if any part was rushed you could risk losing an entire board or worse, an entire module to the mover.

When creating the app, we faced multiple challenges. The first was the app's functionality. At first, we decided to use Android Studio. That meant we had to hardcode all our functions, and none of us had experience doing that beforehand. That meant it was massively time consuming to learn how to code the app. Secondly, we had to learn how Bluetooth worked. We wanted to send data back and forth but had no idea how a connection was established. After finding MIT app inventor and it's built in Bluetooth Client functionality, we were able to proceed with the app.

IX. Future Work

Through showing our project, we found many applications for different scenarios. We found that auto following would be nice for outdoor activities such as camping and gardening. We also learned that there could be a market for having a collapsible wagon for the beach would be a nice market as well as strollers following runners. On top of

residential uses, there was mention of an auto-following wagon being good for construction sites. Through this, we also found assumptions that we could sell at least a thousand units for $800-850 which shows a profit of $800,000 to $850,000 minus the cost of making one. The cost of implementing our current system was around $500, and we could probably bring costs down, which would add to our future work.

Figure 17 – The Future MightyMover

In the future, we hope to take our working system on the auto-following “RC” car and implement it on a wagon as shown above. We would also have to improve many of our systems to be a more reliable product. One thing would be to add more sensors for safety features and add bigger motors for better torque for going up and down hills. The app can be made more user friendly by including a help button/user manual. More types of data can be made available through further development, like battery life in a percentage, whether it is currently moving or not etc.

CacheMoney could also go back to the original idea of having lidar or GPS with a base station to have smart wagons follow pre-determined paths for the user. This would open a new market for commercial use cases.

X. Appendix

MightyMover GitHub

Above shows the GitHub for all of the code used in our final project. This GitHub is public for all to see.

Kai Morich Bluetooth Source Code

This GitHub was used as a reference for the Bluetooth code that we have for the phone app, and we want to make sure that the owner gets recognition for it.

MIT App Inventor

MIT App Inventor was used to make the phone app and was super useful to use due to its easy-to-use layout of coding.

Google Maps API

This API was used in the initial phone app to display location data of the phone itself.

Digikey Tensorflow Lite Model on Raspberry Pi

This link was used to import TensorFlow lite onto the raspberry pi to build and implement the current lightweight model.