

March 17th, 2023

Dr. Mike Hegedus
School of Engineering Science
Simon Fraser University
British Columbia, V5A 1S6

RE: ENSC 405W Design Specification for Company 2

Dear Dr. Mike Hegedus,

The attached document is the design specifications for ScootPilot, the advanced driver assistance system for e-scooters proposed by ADAScooter. Our company is committed to introducing a modern safety solution to electric-scooter riders and nearby road users through the use of a modular ADAS device.

ScootPilot will contain a radar sensor which monitors the distance between the scooter and other vehicles or obstacles on the road. If the system predicts an impending collision, it will automatically apply the brakes to avoid the collision altogether or reduce the likelihood of injury for any party involved. Additionally, if the sensor detects the surrounding area is too crowded with slower-moving people or vehicles, the system will restrict the maximum speed to 12km/hr.

The information laid out in this document goes over the software, hardware and mechanical design aspects of ScootPilot and how they relate to the requirement specifications of this project. Additionally, this document covers such aspects as the pseudocode for the decision-making algorithm, design alternatives and test plans.

ADAScooter can be contacted about any questions and concerns regarding this document through our CCO, Alejandro Lorenzo-Luaces, who can be reached at alorenzo@sfu.ca.

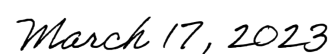
Enclosure: Design Specifications Document

Sincerely,

Alejandro Lorenzo-Luaces, CCO



Signature of CCO



Date



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

Design Specifications

Modular E-Scooter Advanced Driving Assistance System (ADAS)



CEO	Robert Respicio (rrespici@sfu.ca)
CFO	Makan Mousavifard (makanm@sfu.ca)
CCO	Alejandro Lorenzo-Luaces (alorenzo@sfu.ca)
CLO	Pouria Khodabakhsh (pkhodaba@sfu.ca)
CTO	Chris Tesar (ctesar@sfu.ca)
COO	Gregory Sheppard (gjsheppa@sfu.ca)

March. 17, 2023

Abstract

This document will outline the design specifications for the ScootPilot to ensure that e-scooter riders are safe by throttling the speed when the user is in a risk-prone riding environment or the rider is about to crash into an object. In particularly dangerous scenarios, our device will apply an emergency brake. To ensure a safe riding experience for the user, the device uses a radar sensor to calculate the distance and relative velocity of an object in front of the user. Depending on the distance and velocity, the ScootPilot system will throttle the motor safely and notify the user audibly, avoiding collisions. To accomplish this, a Raspberry Pi 4B will interface with the radar sensor and scooter motor controller to throttle the motor. The Raspberry Pi will also signal the mechanical braking mechanism to stop the scooter. By creating this system, e-scooter riders will benefit from a safer riding experience, especially within a crowded, city setting.

Table of Contents

Abstract	i
Table of Contents	ii
List of Figures	iii
List of Tables	iv
Glossary	v
Version History	v
Approvals	v
1. Introduction	1
1.1 Scope	1
1.2 Challenges	1
1.3 Updates on Feedback	2
1.4 Design Specification Classification	2
2. System Overview	2
3. ScootPilot Design	6
3.1 Mechanical Design Specifications	6
3.1.1 Mechanical Design Calculations	6
3.1.2 Mechanical Design	10
3.2 Software Design	14
3.2.1 Software Design Specifications	14
3.2.2 Software State Controller	14
3.2.3 Radar Software	17
3.2.4 GUI Software and Libraries	19
3.2.5 Raspberry Pi GPIO ports	19
3.3 Electrical and Hardware Design	20
3.3.1 Processor and Sensor	20
3.3.2 Speaker Wiring and Operation	25
3.3.3 Battery	26
3.3.4 E-scooter Controller	26
3.3.5 Display	28
3.3.6 E-Brake Motor Controller	29
4. Conclusion	30
5. References	31
Appendix A: Test Plan	33
A.1 Proof of Concept Deliverables	33
A.2 Testing Procedures	33
Appendix B: Alternative Design	38
B.1 Auto-Braking Motor Mount	38
B.2 Velocity measurements	38

List of Figures

- Figure 1:** Simple Block Diagram of ScootPilot
- Figure 2:** Block diagram for the whole system
- Figure 3:** Isometric View of the Whole System
- Figure 4:** Display and Speaker System Mounted on Handlebars
- Figure 5:** Motor for Pulling the Brake Lever
- Figure 6:** Free Body Diagram of the Wheel Connected to the Brake Pads
- Figure 7:** Mechanical Advantage of the brake lever L1/L2
- Figure 8:** Cytron 12V 75RPM Spur Gearmotor
- Figure 9:** Appearance size of the Cytron 12V 75RPM Spur Gearmotor
- Figure 10:** Motor Mount for Handlebar
- Figure 11:** Display and Speaker on Mount
- Figure 12:** CAD of Display and Speaker Mount
- Figure 13:** Radar, Processing Unit and Battery Unit Enclosure
- Figure 14:** CAD of Radar, Processing Unit and Battery Unit Enclosure
- Figure 15:** System Control Loop from Software Perspective
- Figure 16:** Look up Table for States
- Figure 17:** State controller pseudo code
- Figure 18:** Acconeer Exploration Tool Interface
- Figure 19:** Plotting magnitude of complex Sparse IQ data
- Figure 20:** Pseudo code for velocity measurements
- Figure 21:** External GPIO ports on the Acconeer XE-121
- Figure 22:** Raspberry Pi 4B Critical Ports and Dimension
- Figure 23:** Raspberry Pi 4B GPIO Pin Layout
- Figure 24:** Acconeer XE-121 Front View with the Dimensions
- Figure 25:** Acconeer XE-121 Connection to the Raspberry Pi Pin Layout
- Figure 26:** Odseven Mini External USB Stereo Speaker
- Figure 27:** *Circuitry used for controlling the throttle.*
- Figure 28:** *5-inch Touchscreen Displays with Dimensions and Connection to Raspberry Pi .*
- Figure 29:** Description of throttle motor controller

List of Tables

Table 1.4.1: Design Specification Encoding Classification

Table 2.0.1: Bill of Materials

Table 3.1.1: Mechanical Design Specifications

Table 3.2.1.1: Software Design Requirements

Table 3.3.1.1: Microprocessor and Sensor Design Specification

Table 3.3.1.2: Pin Description of Acconeer XE-121 Connection to the Raspberry Pi

Table 3.3.2.1: Speaker Design Specification

Table 3.3.3.1: Battery Design Specification

Table 3.3.4.1: Design Specification for E-Scooter Controller

Table 3.3.5.1: Design Specification for Display

Table 3.3.6.1: Design Specification for Motor Controller

Glossary

Term	Definition
ADAS	Advanced Driver Assistance Systems
AUX	Auxiliary
CAD	Computer-Aided Design
E-scooter	Electronically propelled kick-Scooter
LCD	Liquid Crystal Display
USB	Universal Serial Bus
LUT	Look Up Table

Version History

Document Version	Date
1	March 11, 2023
2	March 15, 2023
3	March 17, 2023

Approvals

The design specification for ScootPilot safety device, which is currently in development, has been reviewed and approved by ADAScooter company.

1. Introduction

In British Columbia, there is currently a pilot program in place for electric scooters, which has raised concerns regarding the safety of riders and pedestrians alike. To address these concerns, ADAScooter is working to develop a safety device, ScootPilot utilising ADAS for e-scooters. By implementing this device, we aim to substantially reduce the occurrence of accidents involving e-scooters and improve the overall safety of individuals using this mode of transportation. By constantly monitoring the surroundings for potential collisions, ADAScooter will be able to promptly adjust and/or apply the brakes to the e-scooter, thereby mitigating the risk of accidents. To provide a clear overview of the safety device being developed, Figure 1 illustrates a simple block diagram showcasing the basic components of the system.

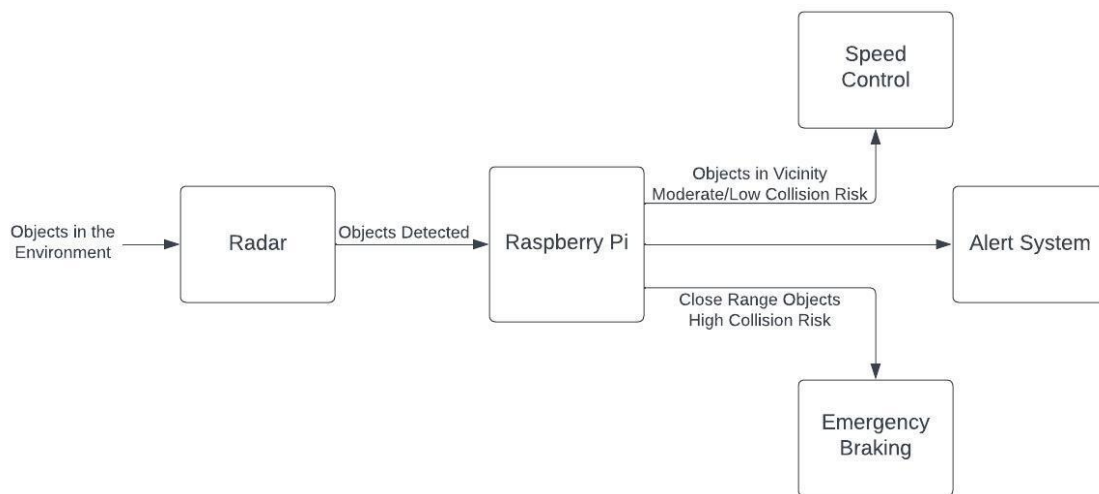


Figure 1: Simple Block Diagram of ScootPilot

1.1 Scope

The purpose of this document is to outline the design decisions of ADAScooter in its endeavour to create a modular Advanced Driver Assistance System (ADAS) for electric scooters. The document comprises a comprehensive account of the mechanical, software, and electrical preferences, with each decision being substantiated by detailed justifications. Additionally, the document is accompanied by two appendices, namely, Appendix A which outlines the test plan and Appendix B which presents an alternative design.

1.2 Challenges

Below is a list of challenges that ADAScooter may face during the design of the device. However, additional challenges may occur later in the completion of the project.

- Insufficient/excessive torque for the motor to apply the brake
- Radar not detecting objects
- High power consumption from peripheral devices, resulting in low battery life

- Software not properly sending signals to stop the scooter
- Creating a UI that is easy to navigate

1.3 Updates on Feedback

Upon receiving feedback from Dr. Mike Hegedus, Yalda Foroutan, and Usman Ahmed, the company (ADAScooter) has made the following changes to the design to ScootPilot:

1. The device will slow down when object presence is detected to maximize the safety of the user. The braking system will be used only as a last resort.
2. Designing and implementing a control system to be added to an e-scooter for speed reduction.

1.4 Design Specification Classification

The following convention has been used in the design specification document.

Des {Encoding [A-C]}-{Section}.{Subsection}.{Requirement Number}

The requirement specification document is listed in order of priority. The example below provides the used convention alongside the table for the categories:

Des A.1.1.1 - Primary design specification for section 1.1.

Des A.2.1.1 - Primary design specification for section 2.1.

Encoding	Stage of Development
A	Proof-of-Concept
B	Engineering Prototype
C	Production Version

Table 1.4.1: Design Specification Encoding Classification

2. System Overview

In this section, we present a detailed overview of the safety module device that has been specifically designed for ScootPilot. The device is equipped with an XE121 Acconeer AB radar that continuously scans the surrounding area for potential obstacles and alerts both the driver and nearby individuals in the event of a high risk of collision. This alert system consists of both audio and visual warnings. An audio speaker alerts the driver and pedestrians of an approaching e-scooter, while a small display screen is used as a user interface. Moreover, in crowded areas, the device can automatically adjust the speed of the e-scooter to half, to reduce the risk of collisions. In case of emergency situations, the device activates the e-scooter's built-in braking system and automatically cuts off the throttle and applies a brake, preventing any potential accidents. The Raspberry Pi 4 is responsible for receiving and interpreting analog signals from the sensor to determine whether an obstacle

has been detected, enabling the device to take the necessary actions. The entire device is powered by a rechargeable Lion battery, designed to provide sufficient power to run for the duration of a single charge of an e-scooter battery, ensuring uninterrupted operation and reliability. The implementation of this device is of utmost importance in ensuring the safety of both e-scooter riders and those in the vicinity. Figure 2 shows the overview block diagram of the system.

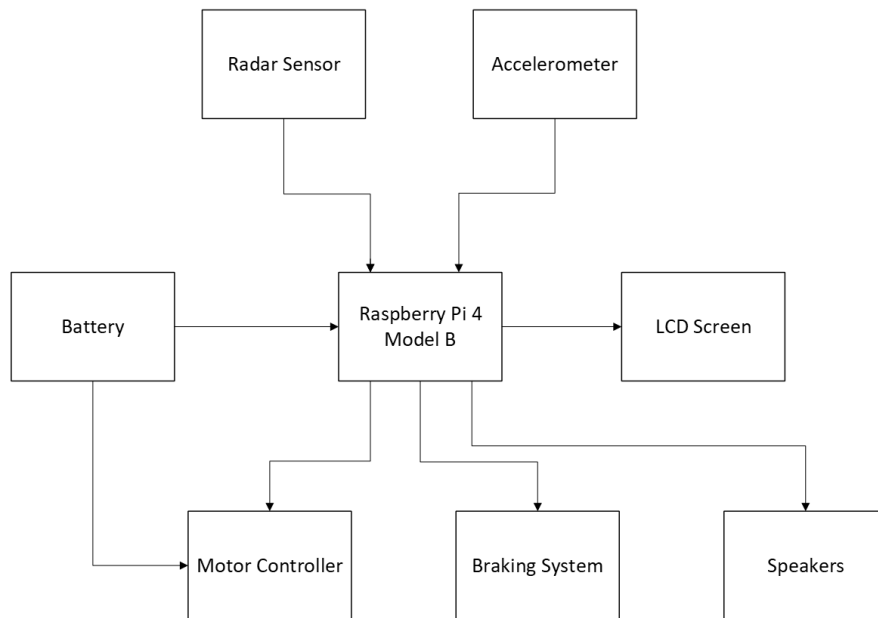


Figure 2: Block diagram for the whole system

During the proof-of-concept (PoC) stage, the radar system implemented in the e-scooter will detect stationary objects such as a wall or a person, and transmit the information to the Raspberry Pi processor. Raspberry Pi will interpret the data and send signals to the speed controller and braking system for necessary decision-making, while simultaneously activating alerts through the speaker and LCD. All of these components will be powered by a battery, allowing for a comprehensive and efficient detection and response system.

The Figures 3,4, and 5 below illustrate the attachment of the various components of the ScootPilot device to an e-scooter. The radar and Raspberry Pi are housed inside a case that is securely affixed to the headset of the e-scooter. The LCD display and speaker are mounted on the handlebar for easy visibility and audibility. The braking system is attached to the left handle where the lever is located, while the speed controller is positioned adjacent to the e-scooter's main controller. This configuration ensures that all components of the ScootPilot system are optimally positioned for efficient and effective operation.



Figure 3: Isometric View of the Whole System

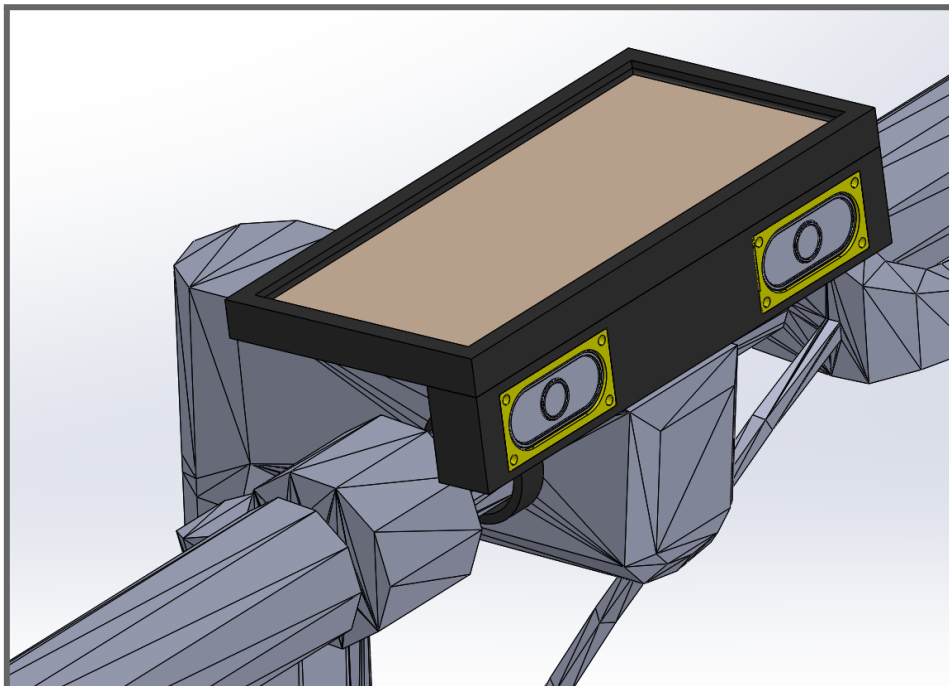


Figure 4: Display and Speaker System Mounted on Handlebars

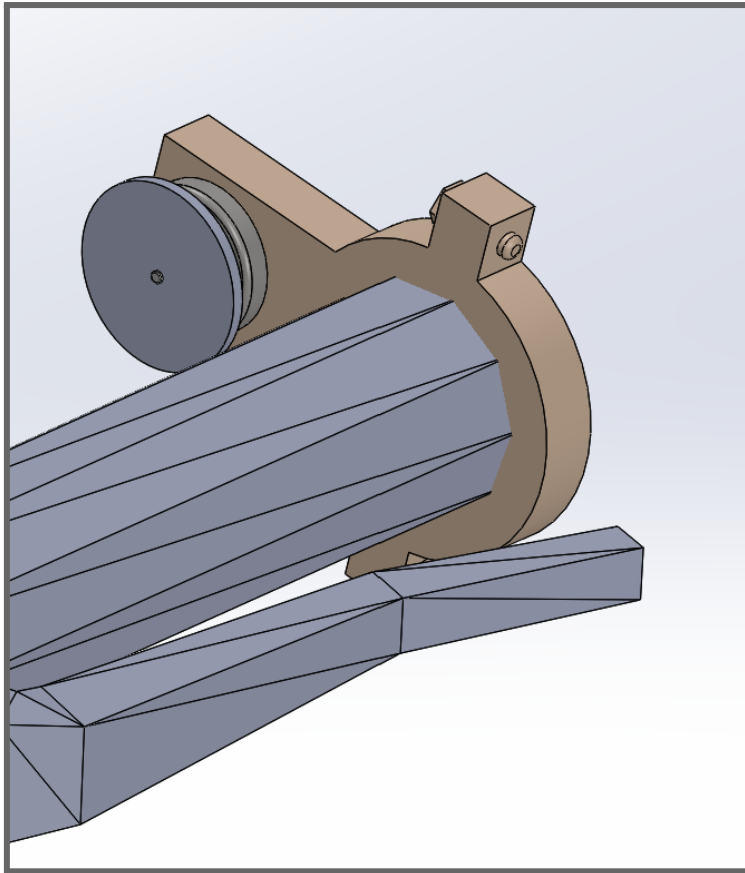


Figure 5: Motor for Pulling the Brake Lever

Table 2.0.1 provides a comprehensive bill of materials, detailing the cost of all components necessary for constructing the ScootPilot device.

Material	Unit Cost (\$)	Quantity	Total Cost (\$)
Raspberry Pi 4 Model B	66.13 [1]	1	66.13
XE121 Acconeer AB	236.48 [2]	1	236.48
Waveshare 5" Capacitive LCD Display	71.95 [3]	1	71.95
Odseven USB Mini Speakers	11.25 [4]	1	11.25
Charmast 10400 Power Bank	42.99 [5]	1	42.99
DF Robot Dual Motor Controller	19.22 [6]	1	19.22
Cytron 12V 75RPM Spur Gearmotor	19.43 [7]	1	19.43
		Total Cost	458.94

Table 2.0.1: Bill of Materials

3. ScootPilot Design

3.1 Mechanical Design Specifications

Design Specification ID	Description	Expected Changes for Future Design	Related Requirement Specifications
<i>Des A.3.1.1</i>	The motor connected to the brake lever provides 5.5 kg.cm of torque	None	Req A-4.1.1 Req A-4.1.3
<i>Des A.3.1.2</i>	The selected motor is rated for 75 RPM	None	Req A-4.1.2 Req A-4.1.3
<i>Des A.3.1.3</i>	The designed motor mount minimally interferes with the user's ability to operate the scooter	Injected moulded plastic to be used for the production version	Req B-3.2.8 Req B-6.1.2
<i>Des B.3.1.4</i>	The motor mount will be 3D printed for prototype purposes	Production version would be smaller dimensions and aluminum alloy	Req B-6.1.2
<i>Des B.3.1.5</i>	The enclosure for the radar, raspberry pi, and battery will be 3D printed.	The production would be injection moulded plastic	Req B-6.1.2 Req B-6.1.3
<i>Des B.3.1.6</i>	The mount for the LCD and speaker will be 3D printed	The production would be injection moulded plastic	Req B-6.1.2 Req B-6.1.3

Table 3.1.1: Mechanical Design Specifications

3.1.1 Mechanical Design Calculations

The design of the braking system as outlined by the requirements specification, is related to the translation of the actuated force at the lever onto the disc. As the automated emergency braking aims to stop the scooter from a maximum speed of 24km/h within 9m from the point of braking, the required linear deceleration is simply calculated as below:

$$a = \frac{v^2 - v_o^2}{2(d)} = \frac{0 - 6.67^2}{2(9)} = - 2.47m/s^2 \text{ (Eq. 1)}$$

At the contact point of the brake pad on the disc, the required amount of force to bring the e-scooter to a full stop can be found with the use of the free body diagram (Figure 6) and the Kinetic energy formula:

$$KE = \frac{1}{2}mv^2 + \frac{1}{2}Iw^2 \quad (\text{Eq. 2})$$

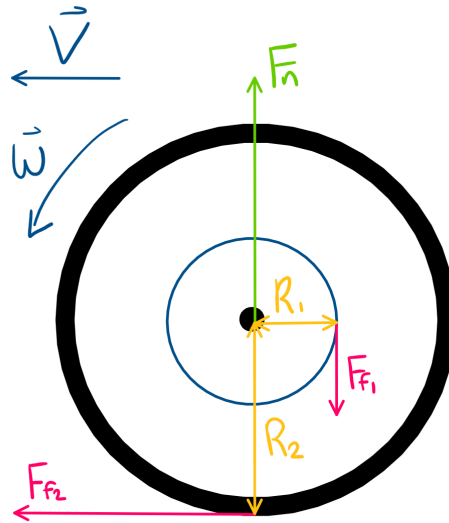


Figure 6: Free Body Diagram of the Wheel Connected to the Brake Pads where M_{Total} is the total mass of the moving object, v is the linear velocity, $I = M_{\text{Wheel}}R^2$ is the moment of inertia of the wheel, and w is the angular velocity.

The rotational kinetic energy about the centre of the wheel, using the sum of the moments, W_{net} is then found as shown below:

$$W_{\text{net}} = \tau_{\text{net}} \theta = [(F1 \cdot R1) + (F2 \cdot R2)] \cdot \frac{9}{R2} \quad (\text{Eq. 3})$$

By equating this to Eq. 2 we can then solve for the required force to bring the e-scooter's kinetic energy to zero, or from 24km/h (KE initial) to a full stop (KE final).

$$[(F1 \cdot R1) + (F2 \cdot R2)] \cdot \frac{9}{R2} = 0 - (\frac{1}{2}mv^2 + \frac{1}{2}Iw^2) \quad (\text{Eq. 4})$$

Then by isolating and solving for F1, we find the required force at the brake pads:

$$F1 = \frac{1}{R1} \cdot \left[\frac{R2}{9} (\frac{1}{2}mv^2 + \frac{1}{2}Iw^2) - F2 \cdot R2 \right] \quad (\text{Eq. 5})$$

Where $F2 = mgCrr$, $Crr = \text{rolling resistance coefficient}$ (Eq. 6)

$I = mr^2$, $I = \text{moment of inertia of the wheel}$ (Eq. 7)

$w = v/R2$, $w = \text{angular velocity of the wheel}$ (Eq. 8)

The table below provides the values of the parameters in the equations above

Parameter	Value	Unit
R1 (disc)	0.055	m
R2 (wheel)	0.136	m
m (wheel)	15	kg
m (scooter + load)	112	kg
Crr	0.001	
v	6.67	m/s

Table 3.1.1.1: Values and Units of the Parameters Used in Force Calculation

Using these values, F1 as shown in equations 5 to 8 is calculated as shown below:

$$\begin{aligned}
 F1 &= \frac{1}{(0.055)} \cdot \left[\frac{0.136}{9} \left(\frac{1}{2} (112)(6.67)^2 + \frac{1}{2} (15)(0.136^2)(6.67/0.136)^2 \right) - (112)(9.81)(0.001) \cdot (0.136) \right] \\
 &= 18.18 \overline{[0.01511(2491.3784 + 333.6667) - 0.149426]} \approx 773.457 \text{ N} \quad (\text{Eq. 9})
 \end{aligned}$$

At the next step, to translate this to the force required at the lever, the factor of the mechanical advantage of the braking system is considered. While the actual mechanical advantage is the ratio of the output force to the input force, it is possible to approximate a mechanical advantage figure with the ratio of the displacement of the input force over the displacement of the output force. The following sample calculation is found by measuring the amount of cable movement at the lever as well as the caliper.

$$\text{Mechanical Advantage} = \frac{\text{Output Force}}{\text{Input Force}} \approx \frac{\text{input displacement}}{\text{output displacement}} = \frac{8 \text{ mm (brake cable movement)}}{1 \text{ mm (brake pad movement)}} = 8$$

(Eq. 10)

Similarly the mechanical advantage of the brake lever at the handlebar is found by the ratio of L1/L2 as shown in the figure below:

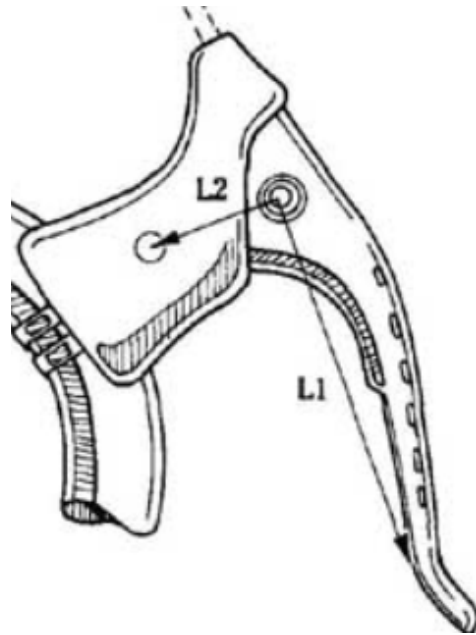


Figure 7: Mechanical Advantage of the Brake Lever $L1/L2$ [8]

This ratio $L1/L2$ according to our measurements is about 3, which once multiplied by the previously found mechanical advantage at the disc gives an estimation of that of the entire system to be around 24. The $F1$ as found previously is then divided by 24 to find the total required force at the brake lever. This $F1$ value is around 32.23Newtons.

The calculated value used various estimations and neglected friction, as well as the resistive spring force at the lever to set it back to zero position; therefore, a factor of safety of 2 is considered to determine the total amount of force required to pull the lever. This then allows us to find a suitable motor for this purpose.

Below is the selected motor to actuate emergency braking.



Figure 8: Cytron 12V 75RPM Spur Gearmotor

This motor is capable of generating 539mN.m which is sufficient considering the pulley design which would be mounted to the rotor shaft with a 3cm diameter. In addition, the extent of the lever displacement from the zero position is measured to be around 42mm, which is about a 160.5 degrees rotation of the pulley mounted onto the motor shaft. This allowed us to determine the required speed of the motor in revolutions per minute (rpm), which is very well covered by our choice of motor that rates at 56 rpm. Calculations below summarise this conclusion:

$$\frac{\theta}{360} = \frac{4.2}{2\pi r} \rightarrow \frac{\theta}{360} = \frac{4.2}{9.42}, \theta = 160.51^\circ \quad (\text{Eq. 11})$$

Aiming to pull the lever in full in half a second, we then get the following rpm:

$$\frac{160.51}{360} \div \frac{0.5}{60} = 53.5 \text{ rpm} \quad (\text{Eq. 12})$$

Figure below provides the sizing specifications of the selected motor:

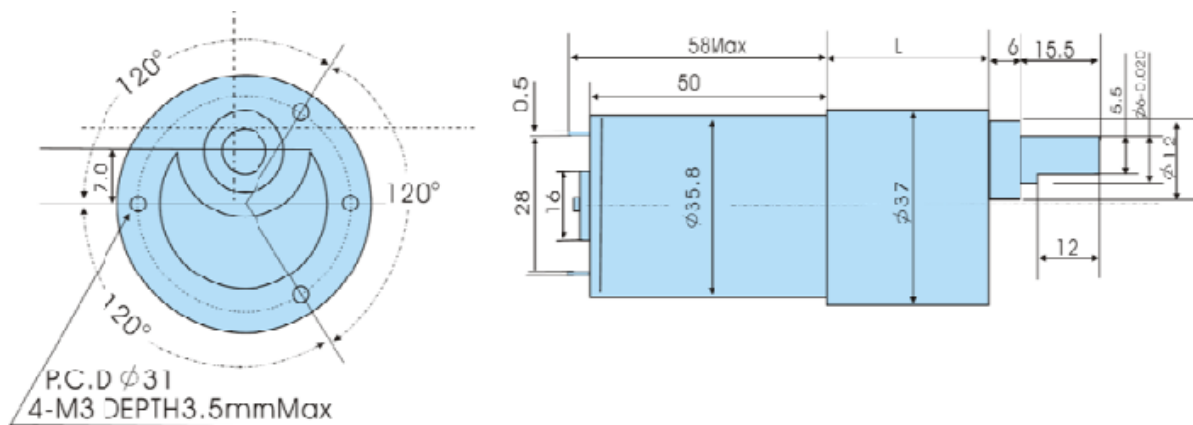


Figure 9: Appearance size of the Cytron 12V 75RPM Spur Gearmotor

3.1.2 Mechanical Design

The main mechanical aspects of ScootPilot involve the automatic braking feature and the housing of all components. As can be seen in Figure 3 above, ScootPilot has 3 points of contact with an e-scooter which has it installed. These 3 points are the motor mount for automatic braking, a display screen with speakers mounted on the handlebars, and a radar sensor with a processing unit attached to the headset of the scooter.

The motor mount which is fixed onto the standard-sized handlebar will be clamped via 2 7mm pinch bolts as seen in Figure 10 below. Spacers can be used to mount onto smaller handlebars. The electric motor will be fixed to the mount through the 35.8mm cylindrical opening. For prototyping purposes, this mount will first be made by 3D printing techniques, but the production model would be made of an aluminium alloy for longevity.

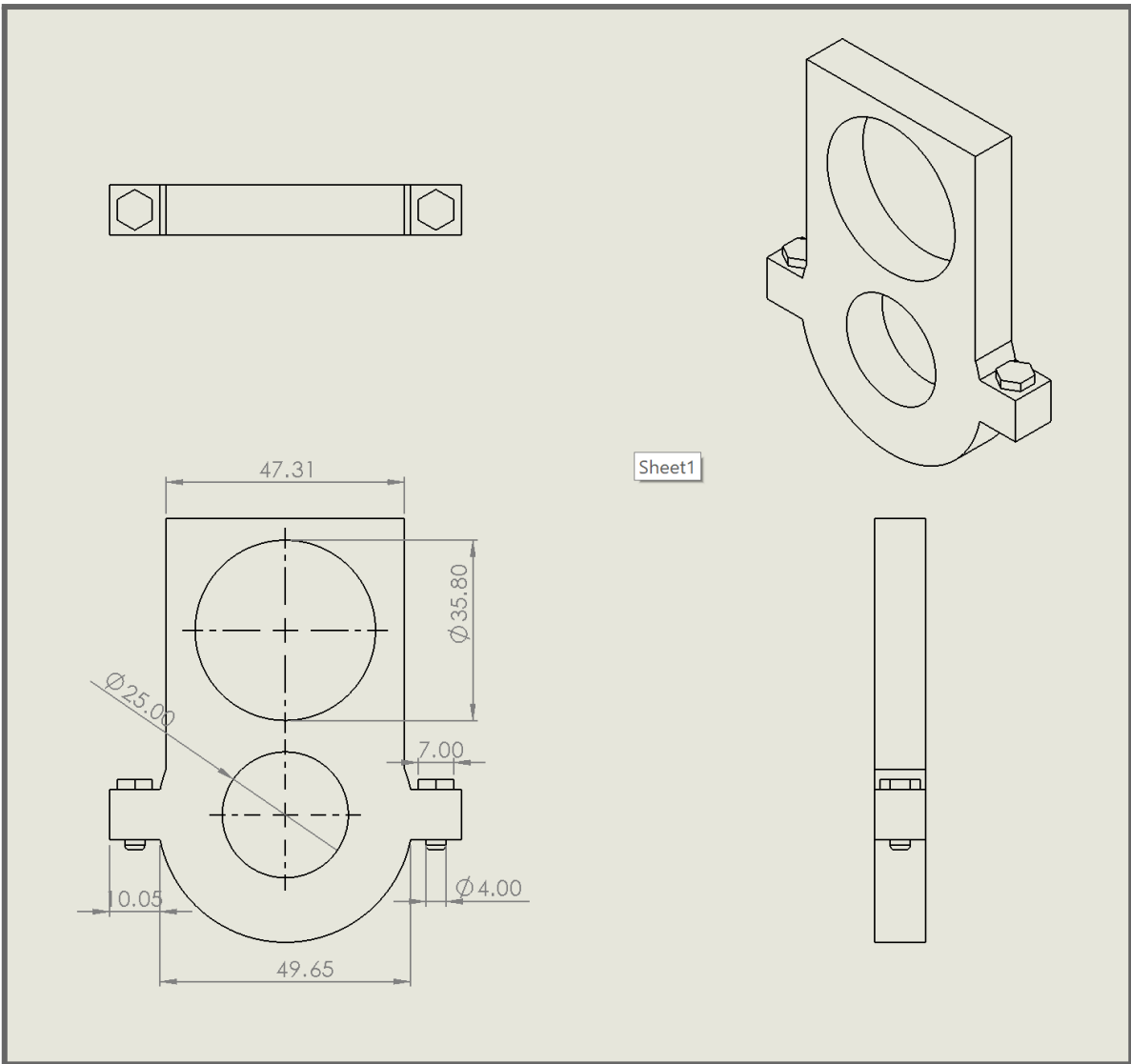


Figure 10: Motor Mount for Handlebar

The speaker and display system will mount to the handlebars with a very similar design to that of the motor mount. The mount uses 2 clamps that go around the handlebars and centre the display. The mount also has room to add the chosen speakers to the system. Similarly to the motor mount, this piece will also be 3D printed for prototype purposes while the production model would be made from injection moulded plastic.

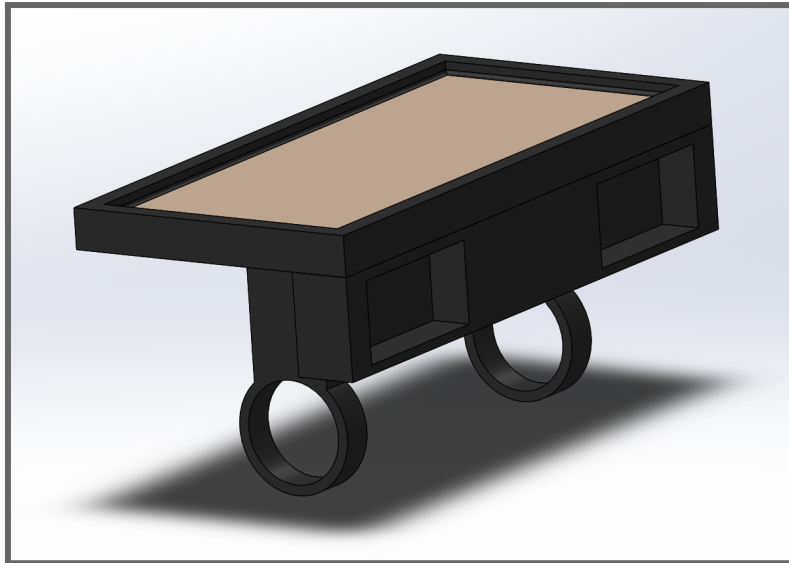


Figure 11: Display and Speaker on Mount

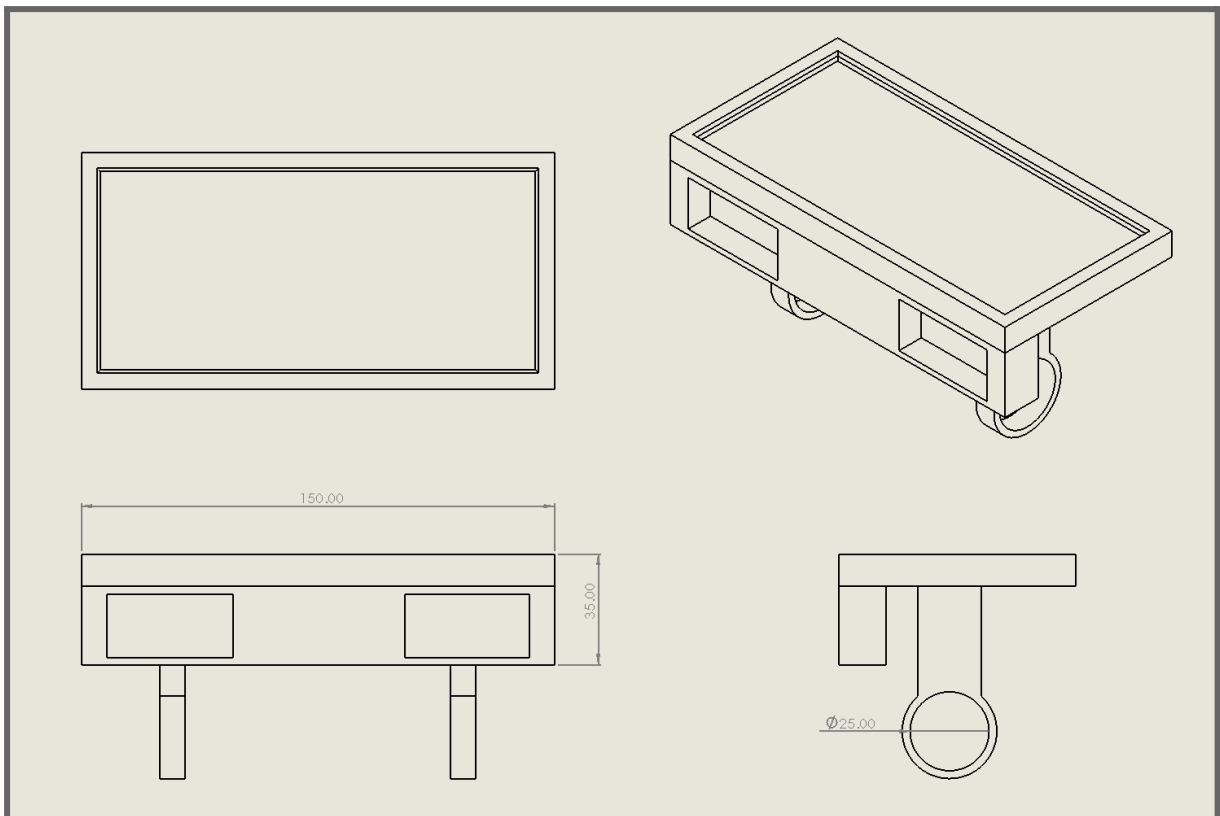


Figure 12: CAD of Display and Speaker Mount

The radar sensor, processing unit and system battery will all be housed in the same enclosure near the bottom of the e-scooter headset. For similar reasons as above, the prototype model would be made from 3D printing of the CAD models while the production version will be made from an injection moulded plastic. The clamp around the headset will squeeze tight into place as to not change its orientation in relation to the front of the scooter

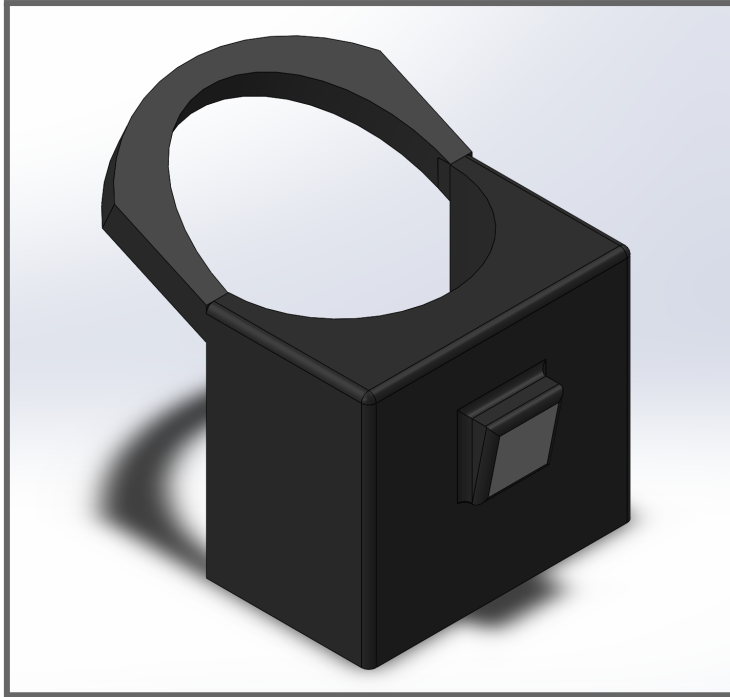


Figure 13: Radar, Processing Unit and Battery Unit Enclosure

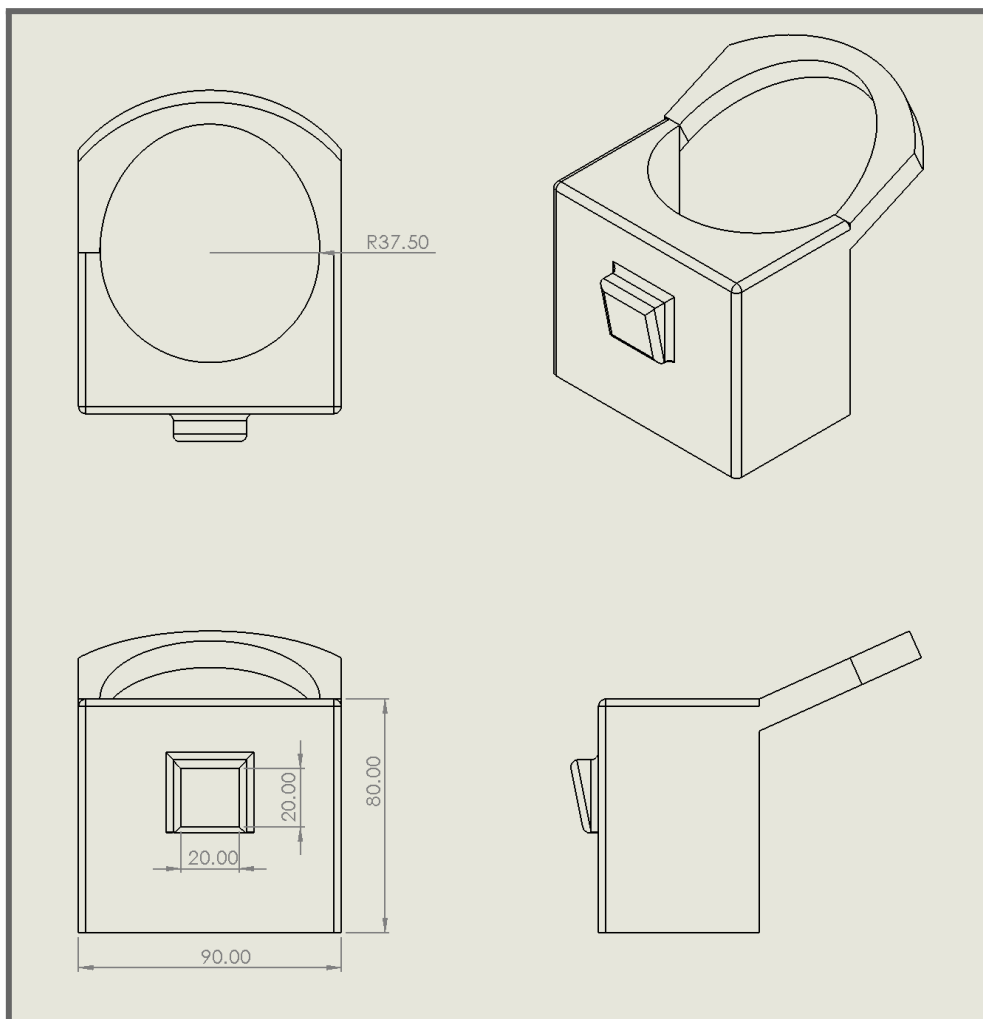


Figure 14: CAD of Radar, Processing Unit and Battery Unit Enclosure

3.2 Software Design

3.2.1 Software Design Specifications

Design Specification ID	Description	Expected Changes for Future Design	Related Requirement Specifications
<i>Des A.3.2.1</i>	Distance measurements will be taken at a rate of 20 Hz or higher.	None	Req A-5.1.2
<i>Des A.3.2.2</i>	The system software will be available at boot up.	None	Req A-5.1.5
<i>Des A.3.2.3</i>	The system will have a self calibration function at start up and through user input.	None	Req A-5.1.6
<i>Des A.3.2.5</i>	The user will be able to interface with the system through a touchscreen GUI.	None	Req A-5.1.5
<i>Des A.3.2.6</i>	The Raspberry Pi 4B will control the motor controller through SPI and/or GPIO ports.	None	Req A-5.1.3
<i>Des B.3.2.7</i>	System will dynamically send signals to adjust motor power based on distance and velocity measurements.	Possibly get speed data from hall effect sensors on the scooter.	Req A-5.1.2
<i>Des B.3.2.8</i>	System will play sounds according to the velocity and distance to the object.	None	Req A-5.1.5

Table 3.2.1.1: Software Design Requirements

3.2.2 Software State Controller

To meet the outlined software design requirements, a state controller onboard the Raspberry Pi 4 is used. Below, Figure 15, outlines the main system control loop for the state controller.

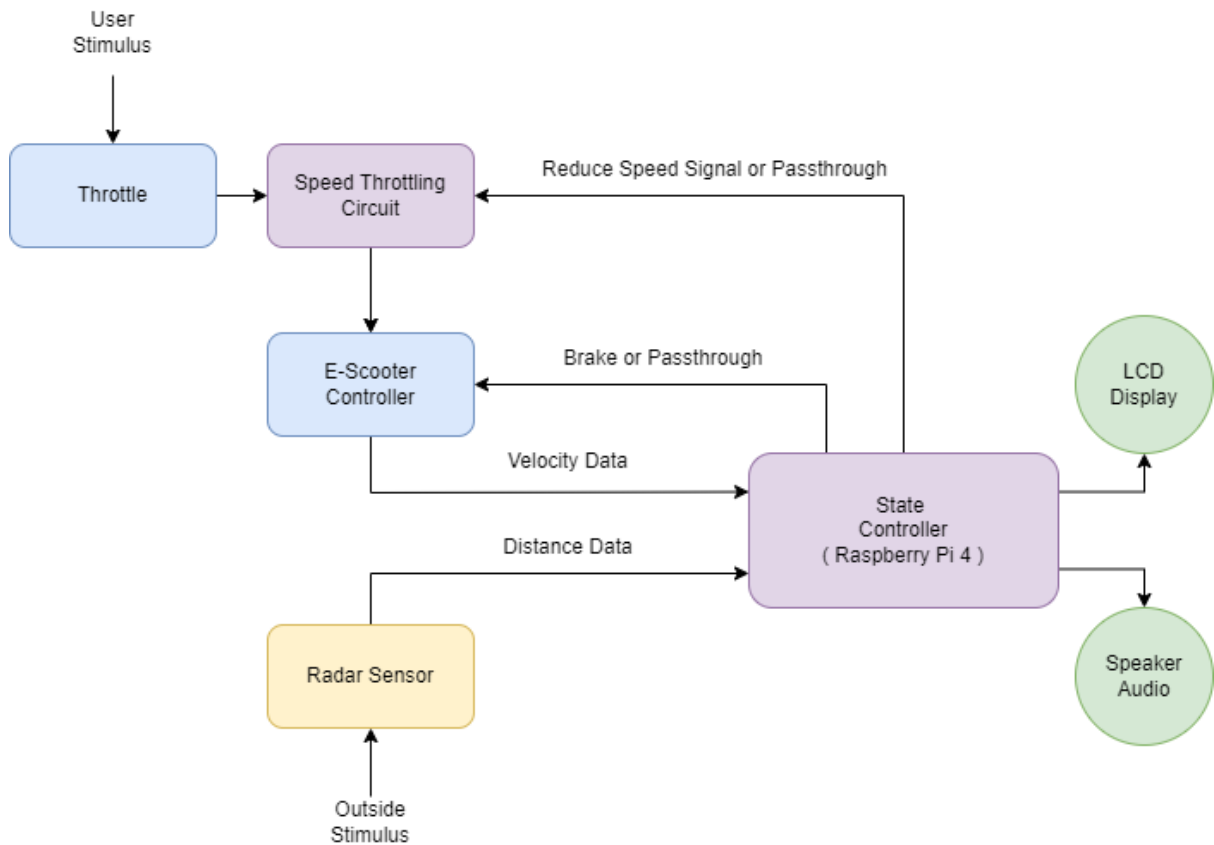


Figure 15: System Control Loop from Software Perspective

Data from the E-Scooter controller and radar sensor equipped with ScootPilot feed velocity and distance data to the state controller onboard the Raspberry Pi. Based on a look up table, the state controller will decide if the E-Scooter must slow down by limiting throttle, emergency brake, or do nothing at all. These control signals are passed back to the E-scooter controller and speed throttling circuit outlined in the Electrical System section. Finally, the speaker will play audio depending on the control signal dictated, and the LCD will receive velocity data from the state controller.

The Look-Up Table (LUT) is a 2D array indexed by the rounded velocity $\in [-24, 24]$ and the rounded distance $\in [0, 20]$. These are dictated by the maximum velocity of the E-Scooter (24 km/h), and the maximum range of the radar sensor (20 m), respectively. For each possible relative velocity and distance, the E-Scooter will brake, throttle speed, or do nothing. These controllers are looked up through the LUT and passed to the various components.

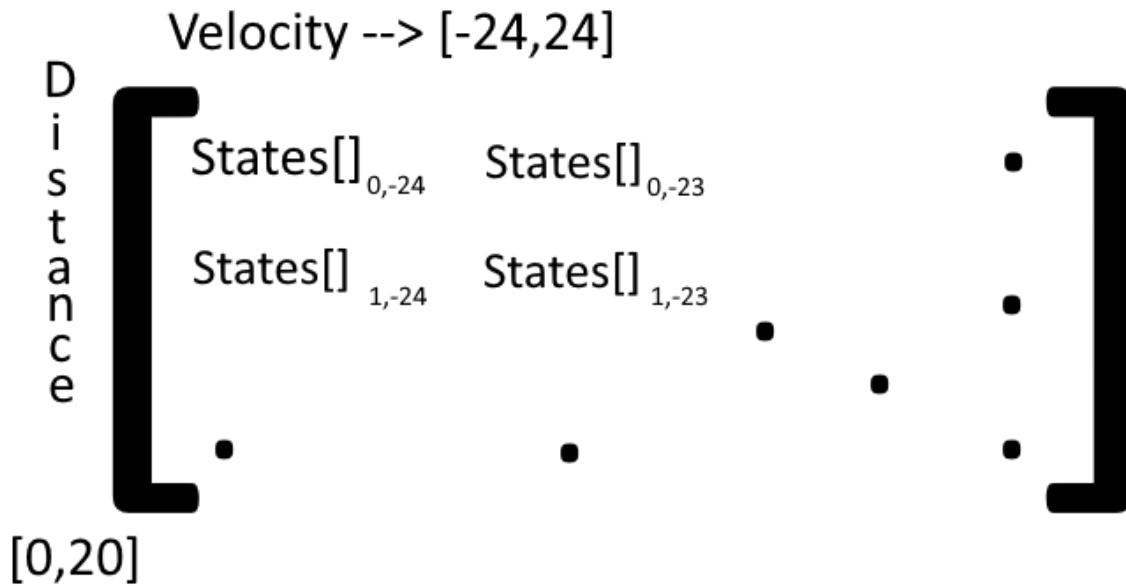


Figure 16: Look up Table for States

The state controller is the main software running on the Raspberry Pi. It dictates what all other components should do based on the input data.

```
main() {

system_init()

while(true) {
    if(power-off) {
        break;
    }

    AvgVelocityDelta = getVelocity()
    AvgDistance = getDistanceFromRadar()

    speakerState, motorState =
getStates(AvgVelocityDelta, AvgDistance)

    updateSpeakerState(speakerState)
    updateMotorState(motorState)
}

system_shutdown()

}

getStates(Velocity v, Distance d) {
    //velocity delta range -24 to 24 m/s
    // distance range 0 to 20 m
    v = round(v)
    d = round(d)
}
```

```

    return LUT[v][d]
}

updateState(State state) {
    gpio(state.pin_number, state.value)
}

```

Figure 17: State controller pseudo code

It begins by initialising the system, then in the main loop, it acquires the velocity and distance data and passes it to the various components through the `getStates` function. This state controller will allow us to modularize the responses to the input data to each system, avoiding unnecessary cross-over in the code base.

3.2.3 Radar Software

The Acconeer XE-121 has an SDK that can be used to develop applications using the sensor. This SDK provides an interface to set configurations in the sensor such as changing the signal strength, which changes the SNR, measurement range, the number of sweeps and algorithms to calculate the distance of the object. In order to properly calibrate our sensor, Acconeer provides the Exploration Tool which provides a way to configure the sensor to the user's needs by visualising the data from the sensor in real-time. Figure 18 below shows the interface that we have access to.

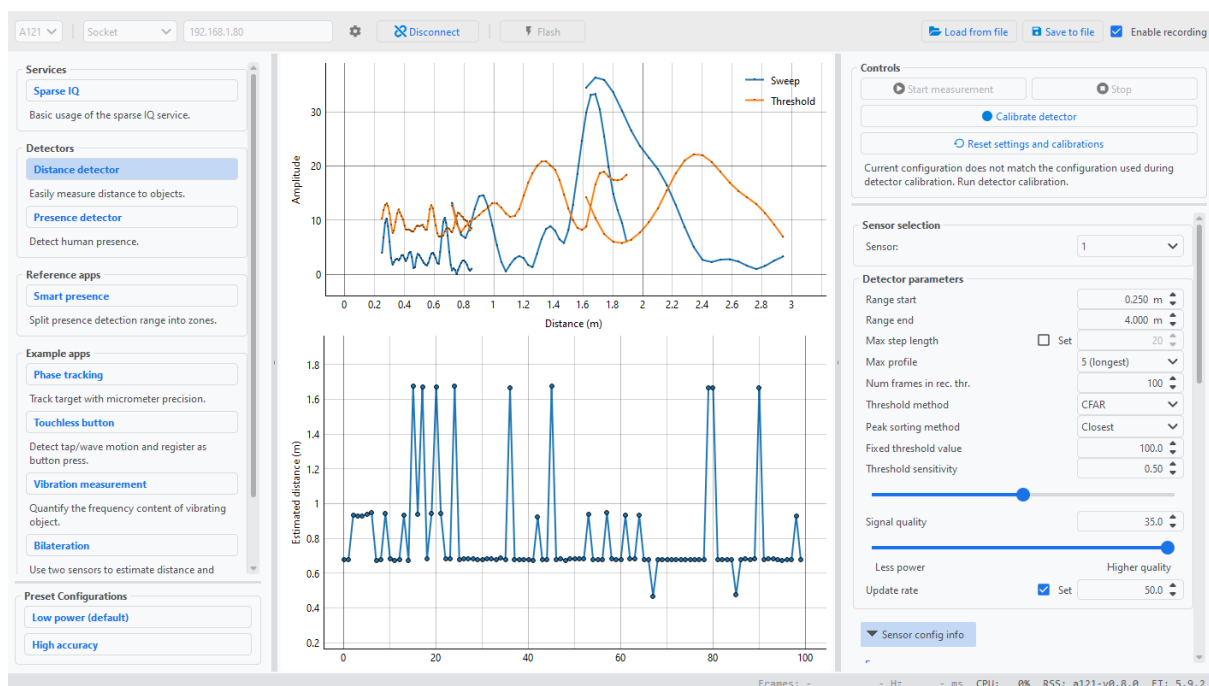


Figure 18: Acconeer Exploration Tool Interface

The sensor allows the user to use a service called the sparse IQ which gives the user raw data in the form of an array of complex numbers as seen in Figure 19. These complex numbers represent the amplitude and phase of the reflected pulse within a certain range. By processing this raw data, we can obtain distance and velocity. Distance can be obtained by taking the absolute value of the complex data and comparing which value gives the largest value. By comparing this absolute value with the threshold, we can determine if an object

has been detected at what distance. With trial and error, it was discovered that using a fixed threshold resulted in fewer false alarms than using the CFAR (Constant false alarm rate) algorithm.

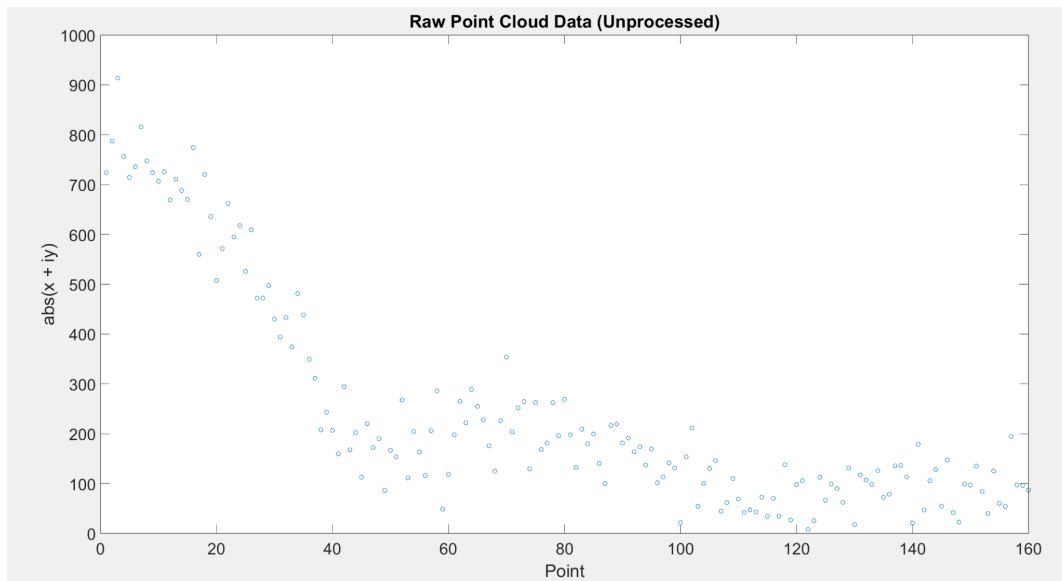


Figure 19: Plotting magnitude of complex Sparse IQ data.

Sometimes, the sensor may give false alarms when detecting an object. Because the sensor is able to record distance data at a fast rate, outliers can be removed by grouping multiple measurements and taking the median. This can be done relatively quickly with a sorting algorithm and taking the center of the array of points. Below, in Figure 20 is a pseudo-code of how this may be implemented.

```
NUMBER_OF_DISTANCE_MEASUREMENTS = 5
```

```
class DistanceData:
```

```
    distance_measurement
    timestamp
```

```
def get_distance_data():
```

```
    distances[NUMBER_OF_DISTANCE_MEASUREMENTS]
```

```
    for i in range(NUMBER_OF_DISTANCE_MEASUREMENTS):
```

```
        dist_data = DistanceData
```

```
        dist_data.distance_measurement = get_distance_measurement()
```

```
        dist_data.timestamp = get_time()
```

```
        distances[i] = dist_data
```

```
    # Insertion sort is fast for small n. Sort by distances. Sorts by distance.
```

```
    insertion_sort(distances)
```

```
    return distances[NUMBER_OF_DISTANCE_MEASUREMENTS//2]
```

```
def calculate_velocity():
```

```
    distance_data_start = get_distance_data()
```

```

distance_data_end = get_distance_data()
velocity = (distance_data.end -
distance_data_start.distance)/(distance_data_end.timestamp -
distance_data_start.timestamp)
return velocity

```

Figure 20: Pseudo code for velocity measurements

3.2.4 GUI Software and Libraries

The GUI will be created using GTK Toolkit. GTK Tool kit is a powerful GUI library that can be used to create expert-looking interfaces. This is a C library that is able to create a GUI and is platform-independent, which means that it can run on the version of Linux running on the Raspberry Pi. Because the Acconeer SDK is written in C, our entire system can be compiled with ease.

3.2.5 Raspberry Pi GPIO ports

Our system will extensively make use of the Raspberry Pi 4B's GPIO and communication ports and pins. There are several ways to do this. One method is to use DRC (Direct Register Control) to control the GPIO pins. This can be done by calling `mmap()` on `/dev/mem`. This will allow the developer to create a new mapping in the virtual address space of the calling process and give the user direct access to the GPIO registers. Another method is to use an API that wraps around the DRC method. This method also allows for controlling off-board and on-board pins which is very advantageous for developers at ADAScooter because the Acconeer XE-121 has accessible pins on its own board. This method allows for developers to access pins on the expansion board as if it were controlling pins on the Raspberry Pi itself.

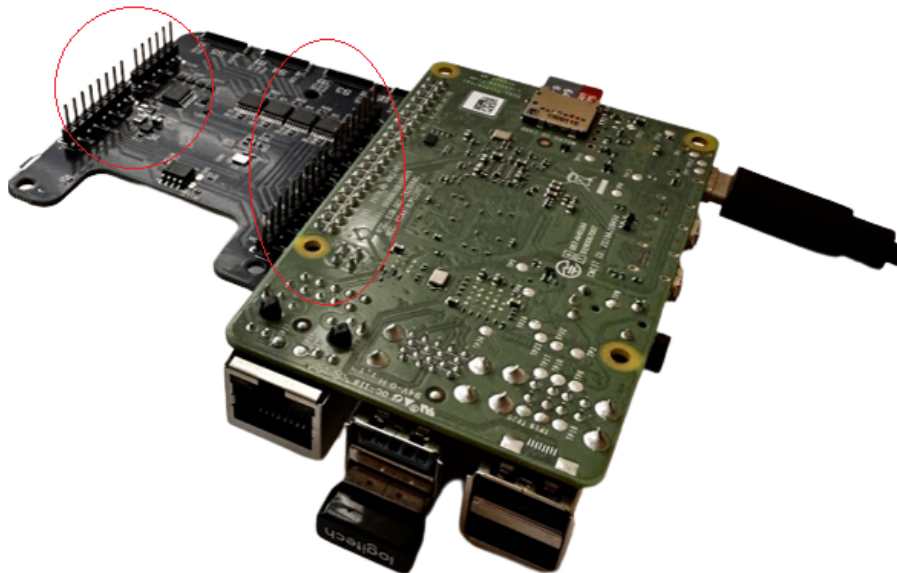


Figure 21: External GPIO ports on the Acconeer XE-121.

3.3 Electrical and Hardware Design

3.3.1 Processor and Sensor

Design Specification ID	Description	Expected Changes for Future Design	Related Requirement Specifications
<i>Des A.3.3.1.1</i>	The microprocessor must process input data fast enough to not create debilitating latency.	None	Req A-4.2.2
<i>Des A.3.3.1.2</i>	The microprocessor should have enough GPIO pins for, radar, speaker, LCD, etc.	None	Req B-3.2.3
<i>Des A.3.3.1.3</i>	The microprocessor power up suing 5V and 1.2A	None	Req A-4.2.1
<i>Des A.3.3.1.4</i>	The radar should be able to detect objects within the safe range.	None	Req B-3.2.7
<i>Des A.3.3.1.5</i>	Radar with sufficient frequency for object detection application	None	Req B-3.2.7

Table 3.3.1.1: Microprocessor and Sensor Design Specification

The Single Board Computer (SBC) used is the Raspberry Pi 4B. This computer has a Quad core Cortex-A72 (ARM v8) 64-bit SoC running at 1.8 GHz with 2 GB of RAM. It will be running the Raspberry Pi OS, a powerful Linux distro which will be able to run any application the team creates. It is an excellent choice for rapid prototyping due to its small footprint, and large number of GPIO pins and ports (HDMI, 4x USB, Ethernet) and rated for maximum 15W power draw but with an average power consumption of about 3W [9]. In addition, Raspberry Pi is an open-source platform that offers users a wide range of options and tools to design and customise their projects according to their specific needs and requirements. Figure 22 illustrates the critical ports available on Raspberry Pi, along with its dimension providing an essential reference for users seeking to leverage the capabilities of this platform.

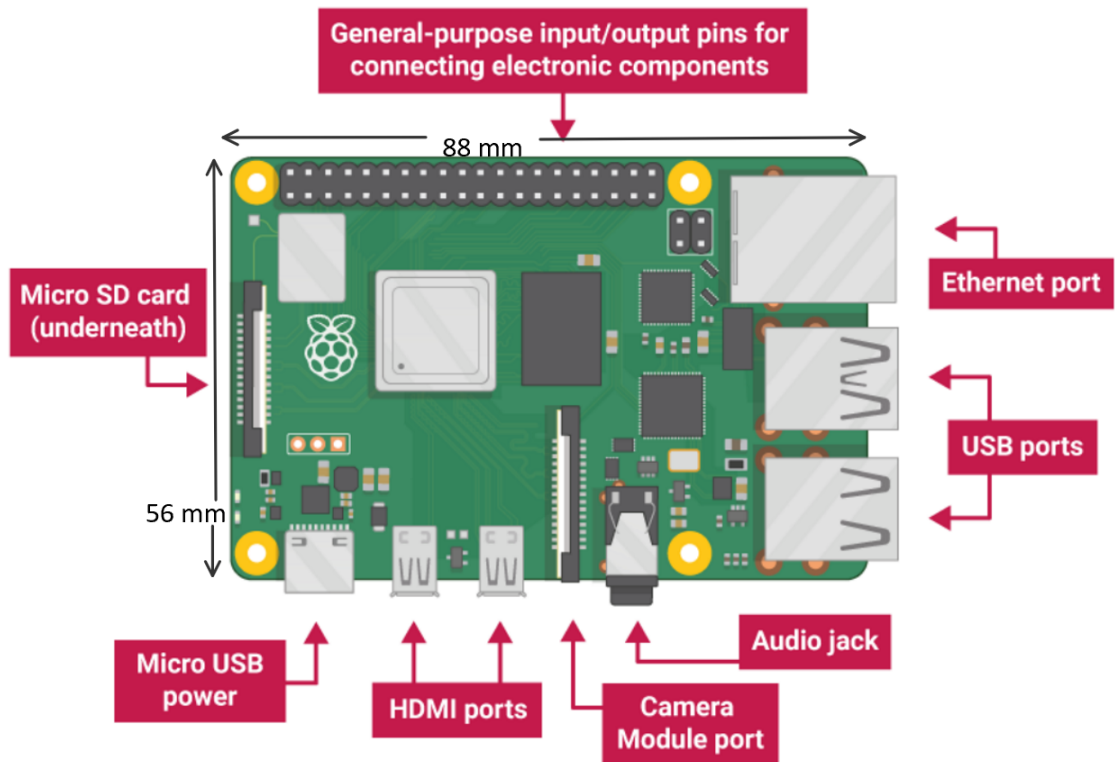


Figure 22: Raspberry Pi 4B Critical Ports and Dimension

Figure 23 shows the GPIO and the 40 pin header of Raspberry Pi 4B which will be used to connect different components to the microprocessor. All 40 pins will connect to the Acconeer XE-121, however the XE-121 has its own header pins that pass through the original 40 header pins on the Raspberry Pi. The pins that can be accessed on the Acconeer XE-121 can be seen in Figure 24.

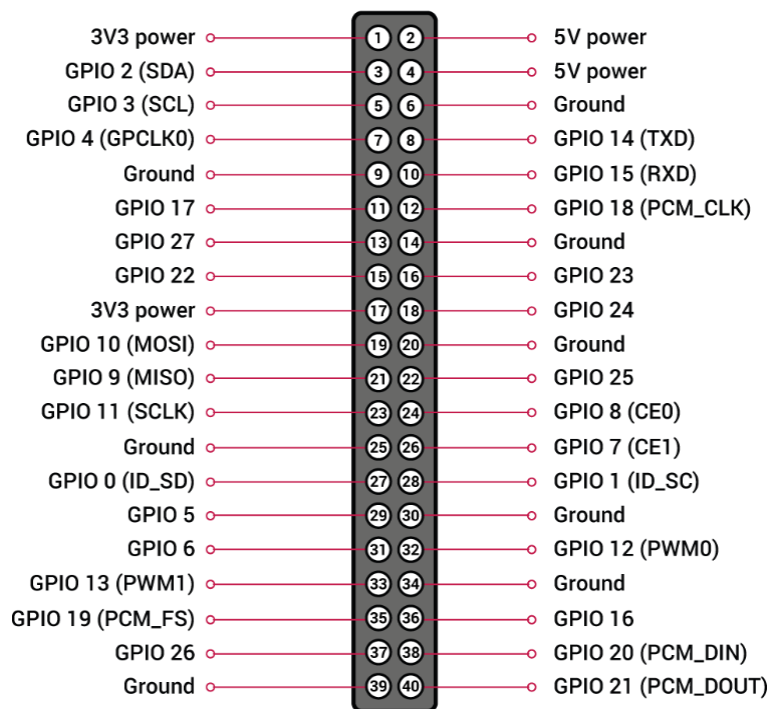


Figure 23: Raspberry Pi 4B GPIO Pin Layout [11]

Another hardware component of the project is the Acconeer XE-121 breakout board. This board has a A121 Radar sensor and header pins which makes it easy to interface with the Raspberry Pi 4 Model B. The radar sensor has a max range of 20m with centimetre precision. The board also has a low power draw at a maximum of 2.5W but will likely consume less during normal operation. Figure 24 demonstrates the Acconeer XE-121 board with its dimensions.

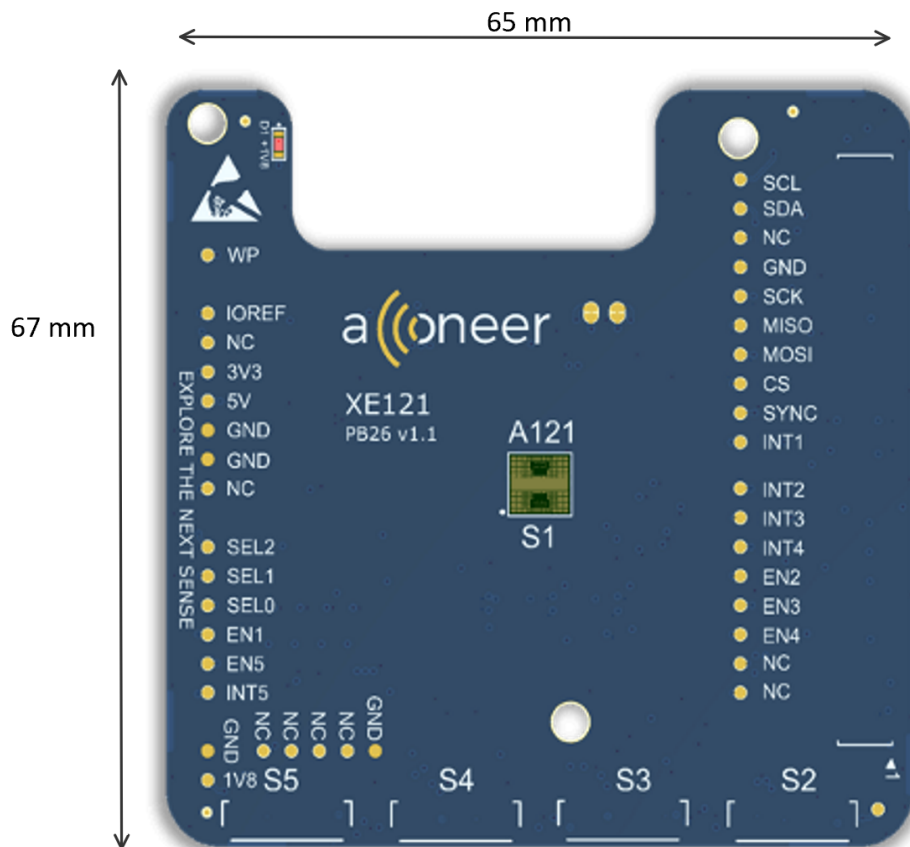


Figure 24: Acconeer XE-121 Front View with the Dimensions

Figure 25 shows the Acconeer XE-121 pins that connect to the Raspberry Pi. Table 3.3.1.2 lists the description of each pin and maps to the ports.

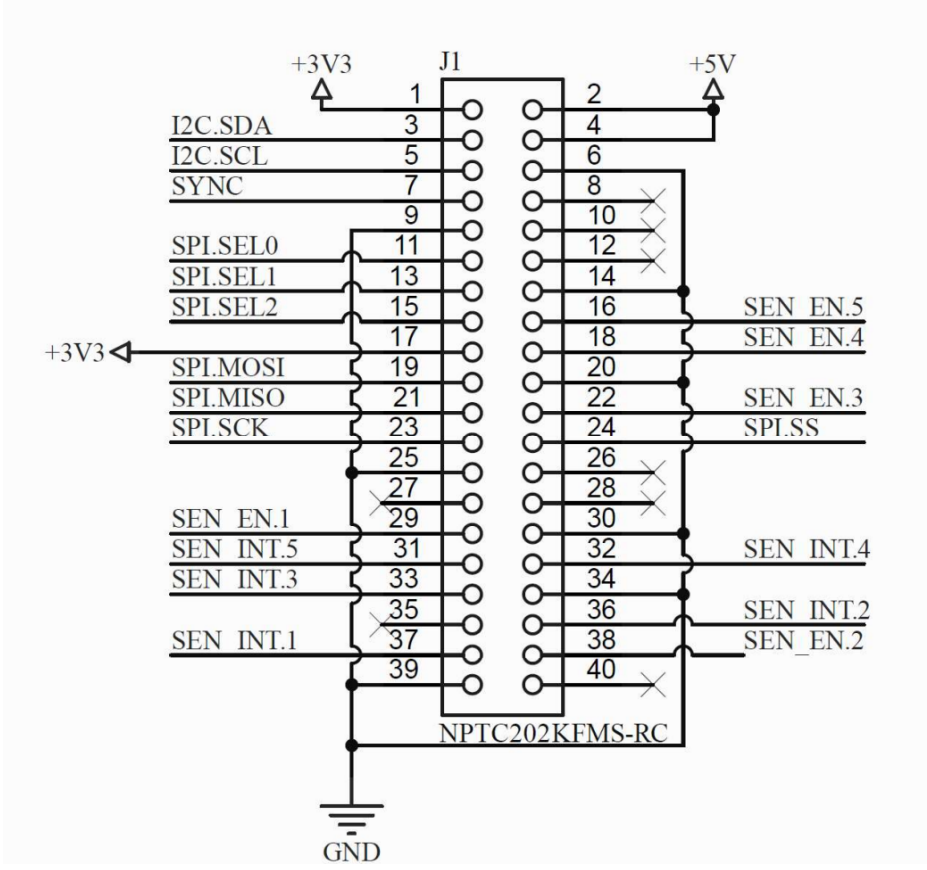


Figure 25: Acconeer XE-121 Connection to the Raspberry Pi Pin Layout

Pin	XE121	Raspberry Pi	Description
1	+3V3	3V3	
2	+5V	5V Power	
3	I2C.SDA	GPIO 2 (I2C1 SDA)	Used for EEPROM on the XE121
4	+5V	5V Power	
5	I2C.SCL	GPIO 3 (I2C1 SCL)	Used for EEPROM on the XE121
6	GND	GND	
7	SYNC	GPIO 4 (GPCLK0)	Signal for sensor synchronization
8	NC	GPIO 14 (UART TX)	Not used on XE121
9	GND	GND	
10	NC	GPIO 15 (UART RX)	Not used on XE121
11	SPI.SEL0	GPIO 17	SPI selection for LVDS enable and SPI SS
12	NC	GPIO 18 (PCM CLK)	Not used on XE121
13	SPI.SEL1	GPIO 27	SPI selection for LVDS enable and SPI SS
14	GND	GND	
15	SPI.SEL2	GPIO 22	SPI selection for LVDS enable and SPI SS
16	SEN_EN.5	GPIO23	XS121 sensor enable signal
17	+3V3	+3V3	
18	SEN_EN.4	GPIO 24	XS121 sensor enable signal
19	SPI.MOSI.CON	GPIO 10 (SPI0 MOSI)	SPI interface for communication with sensor on XE121 and XS121
20	GND	GND	
21	SPI.MISO.CON	GPIO 9 (SPI0 MISO)	SPI interface for communication with sensor on XE121 and XS121
22	SEN_EN.3	GPIO 25	XS121 sensor enable signal
23	SPI.SCK.CON	GPIO 11 (SPI0 SCK)	SPI interface for communication with sensor on XE121 and XS121
24	SPI.SS	GPIO 8 (SPI0 CE0)	SPI interface for communication with sensor on XE121 and XS121
25	GND	GND	
26	NC	GPIO 7 (SPI0 CE1)	Not used on XE121
27	NC	GPIO 0 (EEPROM SDA)	Not used on XE121
28	NC	GPIO1 (EEPROM SCL)	Not used on XE121
29	SEN_EN.1	GPIO 5	XE121 sensor enable signal
30	GND	GND	
31	SEN_INT.5	GPIO 6	XS121 sensor interrupt signal
32	SEN_INT.4	GPIO 12 (PWM0)	XS121 sensor interrupt signal
33	SEN_INT.3	GPIO 13 (PWM1)	XS121 sensor interrupt signal
34	GND	GND	
35	NC	GPIO 19 (PCM FS)	Not used on XE121
36	SEN_INT.2	GPIO 16	XS121 sensor interrupt signal
37	SEN_INT.1	GPIO 26	XE121 sensor Interrupt signal
38	SEN_EN.2	GPIO 20 (PCM DIN)	XS121 sensor enable signal
39	GND	GND	
40	NC	GPIO 21 (PCM DOUT)	Not used on XE121

Table 3.3.1.2: Pin Description of Acconeer XE-121 Connection to the Raspberry Pi [12]

3.3.2 Speaker Wiring and Operation

Design Specification ID	Description	Expected Changes for Future Design	Related Requirement Specifications
<i>Des A.3.3.2.1</i>	Speaker with 60 dB SNR	None	Req A-3.1.3
<i>Des A.3.3.2.2</i>	The speaker is small to easily fit the enclosure	None	Req B-6.1.3

Table 3.3.2.1: Speaker Design Specification

To ensure that the ScootPilot system can alert the user with a sufficiently loud sound, a mini USB-powered speaker designed for Raspberry Pi has been employed. This speaker can be easily powered through the USB port on the Raspberry Pi, which provides a steady 5 volts of power, with a total output of 2x2 W. With a wire length of 1.68 metres and dimensions of 84.0mm x 43.0mm x 32.0mm, this speaker is the ideal choice for the ScootPilot project, meeting both its size and functional requirements. Its compact design and ease of use make it an excellent fit for the system, ensuring clear and effective alerts to the user when necessary. Figure 26 shows the Odseven Mini External USB Stereo Speaker. [13]



Figure 26: Odseven Mini External USB Stereo Speaker

3.3.3 Battery

Design Specification ID	Description	Expected Changes for Future Design	Related Requirement Specifications
<i>Des A.3.3.3.1</i>	Battery capacity with 22.5 Wh or 3000mAh	None	Req A-3.2.2
<i>Des A.3.3.3.2</i>	Battery max rated power of at least 15W	None	Req A-4.2.1 Req A-5.1.4

Table 3.3.3.1: Battery Design Specification

The components which consume the most power and are on at all times are the LCD screen, the processor, and the radar sensor. Adding the power consumption of these components should give us a typical power consumption of less than 10W for the whole system. Considering the braking system and the speakers, the max power draw of the whole system should be no more than 15W at full tilt. The range of our scooter is listed at 28 km, or about 1.2 hours at max speed. Factoring in full tilt consumption and operation at lower speeds, our battery should support a maximum power draw of 15W and an average power draw of 10W for 1.5 hours. At 5V DC, the battery capacity should be a minimum 4500mAh. The battery capacity calculations are shown below in Eq 13 and 14.

$$\text{Battery capacity (Wh)} = 15 \text{ W} \times 1.5 \text{ h} = 22.5 \text{ Wh} \quad \text{Eq. 13}$$

$$\text{Battery capacity (mAh)} = \frac{22.5 \times 1000}{5 \text{ V}} = 4500 \text{ mAh} \quad \text{Eq. 14}$$

We plan to use a power bank for our design. Common 5V-3A DC output power banks are available with capacities starting from 5000mAh which meets our design requirements [5]. Alternatives include using 4 parallel AA batteries and building our own power supply meeting our design specifications.

3.3.4 E-scooter Controller

To control the throttle of the scooter we control the connection of the throttle using a relay to reroute the signal when the throttle needs to be reduced to slow down. In safe conditions the throttle is connected directly to the controller to allow for maximum speed as the user wishes. When conditions unsafe for maximum speed are determined by the device, the throttle signal to the controller is attenuated using a variable resistor controlled by a PWM from another GPIO signal. This allows for a safe, gradual deceleration while maintaining full user control in safe scenarios without having to resort to using the emergency brake. Figure 27 highlights the relay circuitry to enable the throttle.

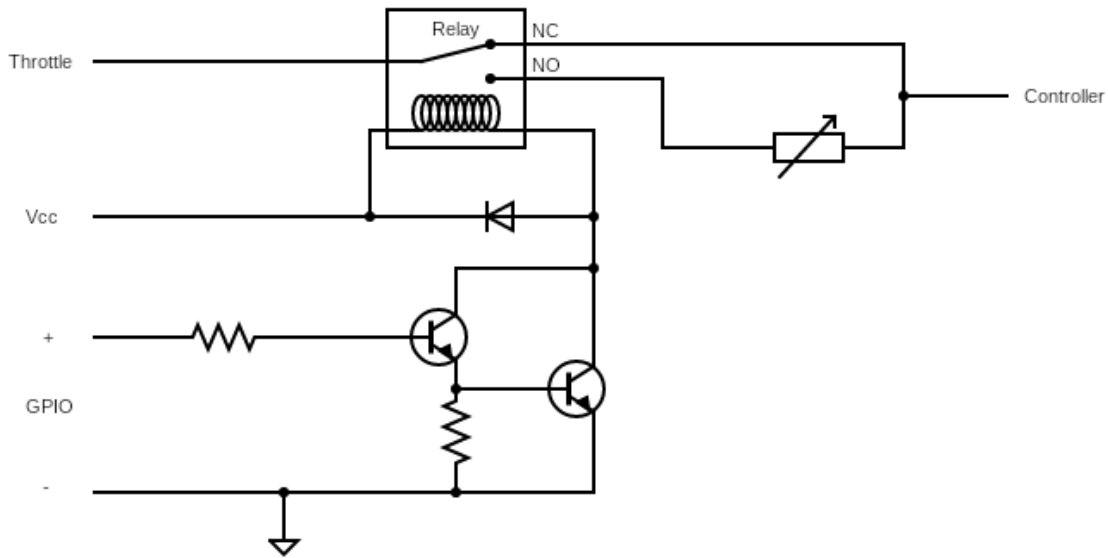


Figure 27: Circuitry used for controlling the throttle.

This circuit amplifies the GPIO signal through a NPN darlington pair to close the relay when the GPIO signal is present. The use of the darlington pair is to ensure the amplified signal is strong enough to close the relay, while guaranteeing that it turns fully off when no signal is present. A flywheel diode is used to protect the darlington pair from back voltage generated when the relay is de-energized. When the relay is open (NC), the throttle signal is allowed to pass-through to the controller. When the relay is closed (NO), the throttle signal is modulated to reduce the throttle by half. Currently, this is represented by the variable resistor in the NO path.

Design Specification ID	Description	Expected Changes for Future Design	Related Requirement Specifications
<i>Des A.3.3.4.1</i>	Throttle circuit can attach to Raspberry Pi GPIO pins	None	None
<i>Des A.3.3.4.2</i>	GPIO signal can fully open and close relay	None	None
<i>Des A.3.3.4.3</i>	The impedance / circuit in the throttle NO path modulates the throttle to 50%	None	None

Table 3.3.4.1: Design Specification for E-Scooter Controller and Throttle Control Circuit

Note that the throttle control circuit design specifications do not have a requirement specifications ID as this feature was added during progress meeting #2.

3.3.5 Display

Design Specification ID	Description	Expected Changes for Future Design	Related Requirement Specifications
<i>Des B.3.3.5.1</i>	5 inch display for clear visual feedback	A mobile app may replace the screen	Req A-3.1.4 Req A-5.1.5
<i>Des B.3.3.5.2</i>	Compatibility with Raspberry Pi 4B	An app may connect to the Raspberry Pi via bluetooth	Req A-4.2.3 Req A-5.1.3
<i>Des B.3.3.5.3</i>	The display is touchscreen	A mobile app may replace the screen	Req B-3.2.3

Table 3.3.5.1: Design Specification for Display

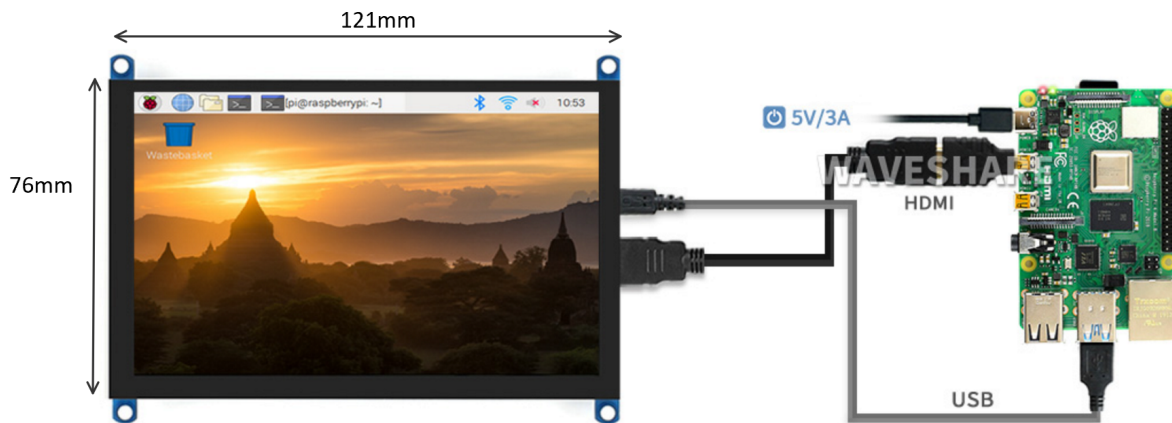


Figure 28: 5-inch Touchscreen Displays with Dimensions and Connection to Raspberry Pi .

We will make use of a capacitive touch screen display for the uses of our product. This allows for the user to interact with the interface easily and intuitively without adding extra bulk in the form of controls and dials. The screen is bright and its large size makes it easier for the user to see the interface in bright sunny conditions. It can be connected directly to the Raspberry Pi via HDMI and the touch interface and DC power are connected via USB. During normal operation at 5V the display should consume about 2.5W of power [14]. Therefore, we will use Waveshare 5" Capacitive LCD Display for ScootPilot which is shown in Figure 28.

3.3.6 E-Brake Motor Controller

Design Specification ID	Description	Expected Changes for Future Design	Related Requirement Specifications
Des A.3.3.6.1	Controller supports Pulse Width Modulation	None	Req A-5.1.3
Des A.3.3.6.2	Controller supports the 12V 320mA as required by the motor	None	Req A-4.2.3
Des A.3.3.6.3	Controller is compatible with the Raspberry Pi	None	Req A-4.2.3

Table 3.3.6.1: Design Specification for Motor Controller

The DFRobot Dual motor controller is based on the L298N dual full-bridge driver. The e-brake control motor will be connected to one of the motor terminals, as well as our battery bank to the power supply pins. The Raspberry Pi GPIO pins will be used to control the throttle motor through the control pins on the diagram below. Note: this motor controller was chosen versus a single motor controller because of a favourable price. The second motor terminal will not be used in the ScootPilot product. [6]

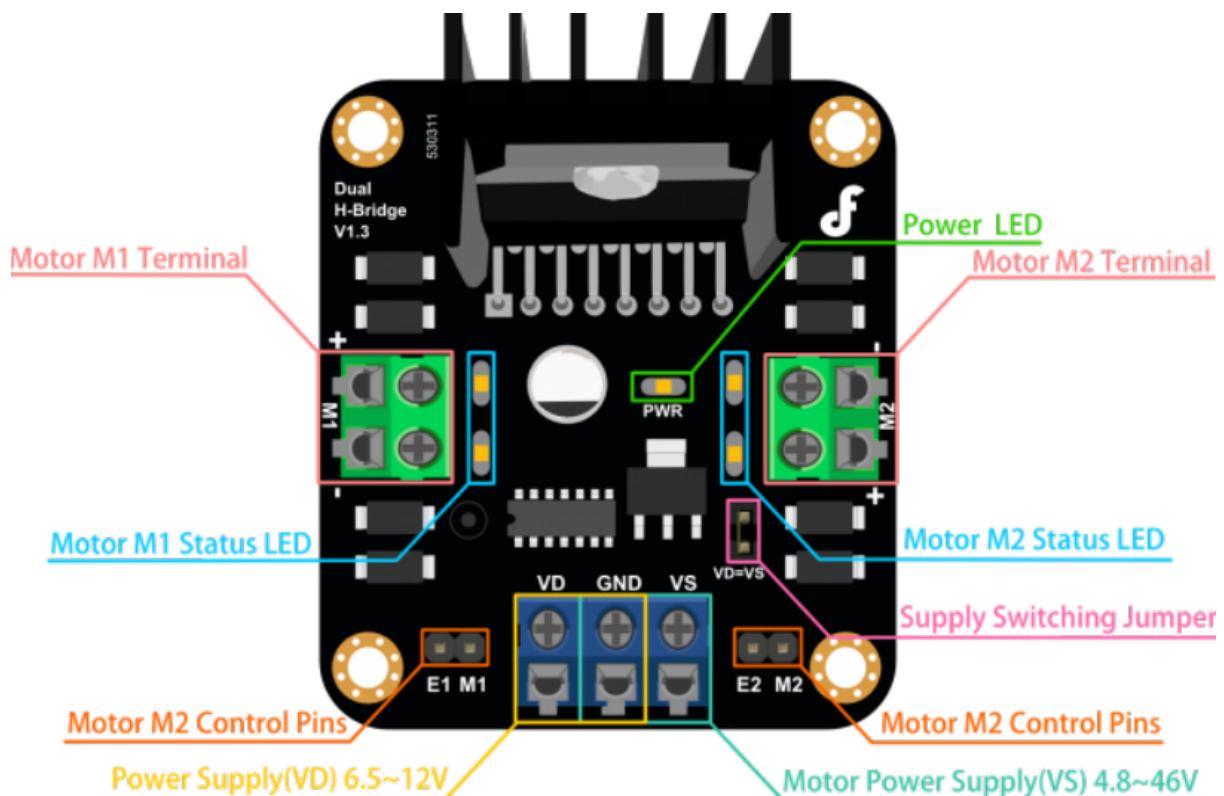


Figure 29: Description of throttle motor controller [6]

4. Conclusion

This design specification document serves as a reference for ADAScooter to best meet the obligations laid out in the ScootPilot requirements specification. It describes the implementation requirements via design specifications tables, and design choices via the figures, tables, and supporting text. It will be used heavily to guide the team members of ADAScooter to fully implement a product to best serve our customers and market.

ScootPilot's design specifications are broken into 3 major sections: Mechanical Design, Software Design, and Electrical / Hardware Design. Each of these sections lay out the design implementation and theory for their corresponding focuses. The Mechanical Design section's main focus is on automated braking and housing, while the Software Design section's focus is on data collection from the controller and radar sensor and implementing a state controller to control the various sub-systems. Finally, the Electrical / Hardware section describes the various circuit designs and power needs for the system.

ADAScooter is passionate about safety. Our goal is to not only protect our customers with ScootPilot, but to continue to help increase the safety of novel technologies that are coming to market. We hope that providing affordable safety solutions will increase the adoption and accelerate the legislation of these technologies while continuing to have the general public's best interests in mind.

5. References

- [1] Raspberry pi 4 model B 8G: Digi-key electronics," *Digi-key*. [Online]. Available: <https://www.digikey.com/en/products/detail/raspberry-pi/RASPBERRY-PI-4-MODEL-B-8G/12159401>. [Accessed: 17-Mar-2023].
- [2] "XE121: Digi-key electronics," *Digi-Key*. [Online]. Available: <https://www.digikey.com/en/products/detail/acconeer-ab/XE121/16582348>. [Accessed: 17-Mar-2023].
- [3] "5inch capacitive touch screen LCD (H) slimmed-down version, 800×480, HDMI, toughened glass panel, Low Power," *PiShop.ca*. [Online]. Available: <https://www.pishop.ca/product/5inch-capacitive-touch-screen-lcd-h-slimmed-down-version-800-480-hdmi-toughened-glass-panel-low-power/>. [Accessed: 17-Mar-2023].
- [4] "Mini USB powered speakers for Raspberry Pi," *X2 Robotics in Canada*. [Online]. Available: <https://x2robotics.ca/mini-usb-powered-speakers-for-raspberry-pi>. [Accessed: 17-Mar-2023].
- [5] "Charmast 10400 smallest PD Portable Charger,Mini Power Bank USB C 20W PD & QC 3.0 Quick Charge External Battery Pack, small compact battery pack compatible with iPhones,Samsung,smartphone," *Amazon.ca: Electronics*. [Online]. Available: <https://www.amazon.ca/Charmast-Smallest-Portable-10400mAh-Compatible/dp/B07L933SN5/>. [Accessed: 17-Mar-2023].
- [6] "DFRobot 4.8-46V, 2A Dual Motor Controller," *RobotShop USA*. [Online]. Available: <https://www.robotshop.com/products/dfrobot-4-8-46v-2a-dual-motor-controller>. [Accessed: 17-Mar-2023].
- [7] "Cytron 12V 75rpm Spur Gearmotor," *RobotShop Canada*. [Online]. Available: https://ca.robotshop.com/products/cytron-12v-75rpm-spur-gearmotor?fbclid=IwAR33Ypt_R7S18TAuXRVsyZVUv3BPixWxexDpREHKJF3Xk2Jvf1DizDYGdrE. [Accessed: 17-Mar-2023].
- [8] K. Hashmi, "Bicycle Upgrading Guide," Khurram Hashmi's Web Site, 2013. [Online]. Available: http://www.khurramhashmi.org/khurramweb_com/bicycle_upgrading_guide-13.html. [Accessed: Mar. 15, 2023].
- [9] "Cytron 12V 75rpm Spur Gearmotor," *RobotShop Canada*. [Online]. Available: <https://ca.robotshop.com/products/cytron-12v-75rpm-spur-gearmotor>. [Accessed: 17-Mar-2023].
- [10] "How much power does the PI4B use? power measurements," *RasPi.TV*, 25-Jun-2019. [Online]. Available: <https://raspi.tv/2019/how-much-power-does-the-pi4b-use-power-measurements>. [Accessed: 17-Mar-2023].
- [11] "DATASHEET - raspberry pi." [Online]. Available: <https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf>. [Accessed: 18-Mar-2023].
- [12] Acconeer AB, "XE121 - Acconeer AB | Sensors, Transducers | DigiKey," Digi-Key Electronics, 2021. [Online]. Available: <https://www.digikey.ca/en/products/detail/acconeer-ab/XE121/16582348>. [Accessed: Mar. 12, 2023].
- [13] ODSven, "ODSeven Mini External USB Stereo Speaker for Raspberry Pi," ODSven, 2021. [Online]. Available: <https://odseven.com/products/odseven-mini-external-usb-stereo-speaker-for-raspberry-pi>. [Accessed: Mar. 14, 2023]

[14] [Waveshare Electronics, "5inch HDMI LCD (B) (with case)," Waveshare Electronics, 2021. [Online]. Available: <https://www.waveshare.com/5inch-dsi-lcd.htm>. [Accessed: Mar. 18, 2023].]

Appendix A: Test Plan

This appendix will outline the testing coverage and methods to ensure the safety of the users of the ADAS system.

A.1 Proof of Concept Deliverables

On April 12th, 2023, the employees of ADAScooter will present the following features and functions of the ScootPilot system:

1. The functionality of the radar sensor interfacing with the Raspberry Pi microprocessor.
2. Variable speed control from the motor controller depending on road conditions and objects in front of the scooter.

A.2 Testing Procedures

Test: Radar Sensor Validation	Time:	Date:
Testing Procedure: <ul style="list-style-type: none">• The tester will power on the system.• The tester will position the sensor so an object can be placed in front.• The tester will run the radar sensor program via command line.• The tester will move an object towards the stationary sensor.		
Expected Outcome: Distance measurements should be accurately displayed, decrementing slowly.		
Observed Outcome:		
Improvements/Comments:		
Score: <input type="checkbox"/> Pass <input type="checkbox"/> Fail	Tester:	

Test: Sound Verification	Time:	Date:
Testing Procedure:		
<ul style="list-style-type: none"> • Tester will connect the speaker to the Raspberry Pi 4B • Tester will run program to play sound from Raspberry Pi 4B • Tester will move towards the radar sensor to collision range. 		
Expected Outcome: Tester should hear a sound coming from the speaker.		
Observed Outcome:		
Improvements/Comments:		
Score: <input type="checkbox"/> Pass <input type="checkbox"/> Fail	Tester:	

Test: Verify LCD Display	Time:	Date:
Testing Procedure:		
<ul style="list-style-type: none"> • Tester will connect the LCD Touch screen to the Raspberry Pi. • Tester will power on the LCD Touch screen and the Raspberry Pi. • Tester will navigate the GUI within the LCD Touch screen. • Tester will touch the settings icon to open the settings menu. 		
Expected Outcome: Screen should be responsive and be able to access all menus.		
Observed Outcome:		
Improvements/Comments:		
Score: <input type="checkbox"/> Pass <input type="checkbox"/> Fail	Tester:	

Test: Verify Emergency Stop	Time:	Date:
Testing Procedure: <ul style="list-style-type: none"> • Tester will connect the ScootPilot system to the E-Scooter. • Tester will calibrate the sensor. • Tester will ride the scooter in a straight line. • Another tester will move an unexpected object in front of the scooter safely. 		
Expected Outcome: The scooter will stop right before colliding with the obstacle. The speakers will play an urgent sounding beeping sound to alert the tester.		
Observed Outcome:		
Improvements/Comments:		
Score: <input type="checkbox"/> Pass <input type="checkbox"/> Fail	Tester:	

Test: Verify Speed Throttle	Time:	Date:
Testing Procedure: <ul style="list-style-type: none"> • Tester will connect the ScootPilot system to the E-Scooter. • Tester will calibrate the sensor. • Tester will move towards a stationary object slowly. 		
Expected Outcome: Scooter should throttle speed as the tester gets closer to the object.		
Observed Outcome:		
Improvements/Comments:		
Score: <input type="checkbox"/> Pass <input type="checkbox"/> Fail	Tester:	

Test: Battery operating time validation	Time:	Date:
Testing Procedure: <ul style="list-style-type: none"> • Tester will connect the system to the scooter. • Tester will calibrate the radar sensor. • Tester will ride the scooter for its specified operating time and verify the ScootPilot battery does not die early. 		
Expected Outcome: The ScootPilot battery will last as long as the main E-Scooter battery.		
Observed Outcome:		
Improvements/Comments:		
Score: <input type="checkbox"/> Pass <input type="checkbox"/> Fail	Tester:	

Test: UI validation	Time:	Date:
Testing Procedure: <ul style="list-style-type: none"> • Tester will connect the ScootPilot system to the E-Scooter. • Tester will verify functionality of all UI buttons and toggles. 		
Expected Outcome: UI works as described in UI document		
Observed Outcome:		
Improvements/Comments:		
Score: <input type="checkbox"/> Pass <input type="checkbox"/> Fail	Tester:	

Test: Test relay circuit for pass through	Time:	Date:
Testing Procedure: <ul style="list-style-type: none"> • Tester will power the Raspberry Pi 4B. • Tester will make sure that the motor controller is connected to motors. • Tester will trigger GPIO controlling relay through the command line with a test script. • Toggle the GPIO pin and watch the effects. 		
Expected Outcome: The motors will not throttle if GPIO is false. Otherwise, the motor should throttle.		
Observed Outcome:		
Improvements/Comments:		
Score: <input type="checkbox"/> Pass <input type="checkbox"/> Fail	Tester:	

Test: Scooter controller velocity data	Time:	Date:
Testing Procedure: <ul style="list-style-type: none"> • Tester will connect the ScootPilot system to the E-Scooter. • Tester will check the velocity read-out from the controller on the LCD display while driving and compare it to a GPS velocity calculated from cell-phone 		
Expected Outcome: Velocity readings from the cellphone and E-Scooter controller will match within $\pm 1\text{km/h}$		
Observed Outcome:		
Improvements/Comments:		
Score: <input type="checkbox"/> Pass <input type="checkbox"/> Fail	Tester:	

Appendix B: Alternative Design

B.1 Auto-Braking Motor Mount

Several ideas were considered as possible means for which to accomplish the physical aspect of autonomous braking. The main goal is to be able to apply the brakes on the e-scooter in a safe and timely manner, without impeding regular use of the scooter, whenever the system deems it appropriate to do so according to the following requirement specifications: Req A-3.2.1, Req B-3.2.8. Specifically, we need to be able to mount a motor which can activate the brake lever without impeding a user from using the brakes themselves.

It came down to two choices of possible motor mount designs. One included a pulley to be able to change the line of direction from the motor, and the design involved a hole going through the handlebars through which the wire to pull the brake lever could be passed through. We came to the conclusion that the pulley design would slightly impede regular use of the brake lever and we're also concerned about the longevity of such a small part which would need to handle relatively high forces on its axis of rotation. The figure below shows the chosen design on the left and the alternate pulley design on the right.

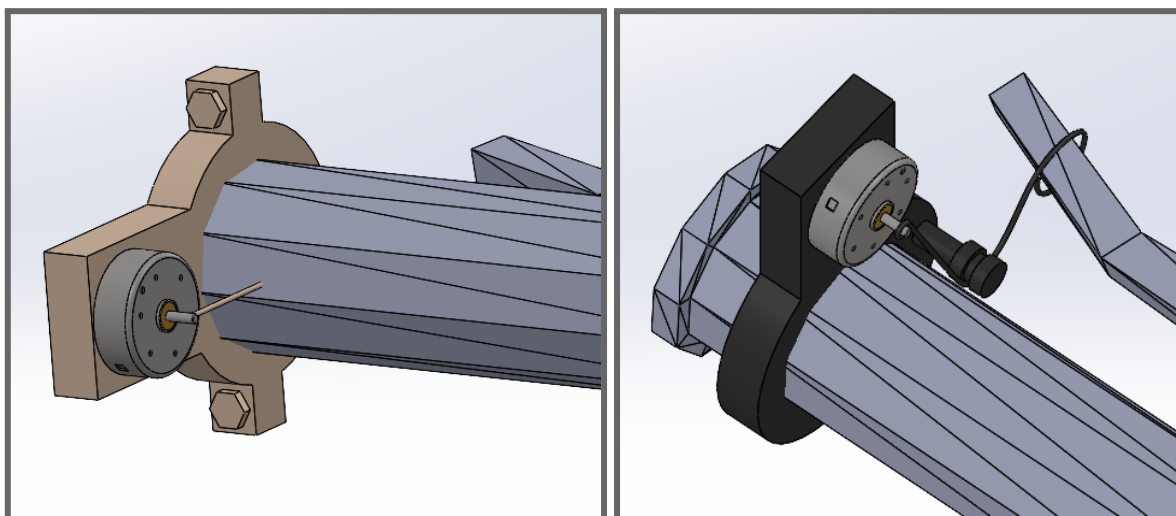


Figure I: Chosen and Alternative Design for Motor Mount

B. 2 Velocity measurements

Because the radar sensor may have distance measurements that are inaccurate, we need to reject outlier distance data in order to accurately measure relative velocity. However, this is computationally expensive as we need to sample distance multiple times to reject outliers. The scooter has a hall effect sensor which can be used to calculate speed. This allows the radar sensor to focus on taking distance measurements faster and take fewer resources and may provide a more accurate way of measuring speed. Using this method will make it easier to meet *Req A.3.2.1* and *Req B.3.2.7*. Figure II highlights where the hall effect signal can be read on the scooter motor controller.

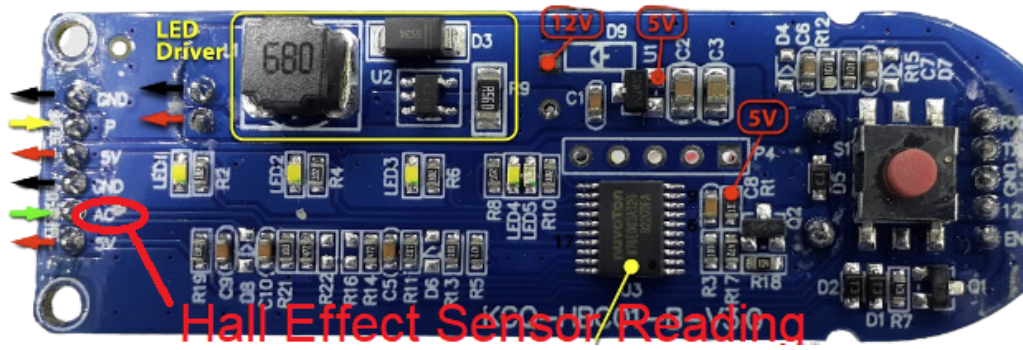


Figure II: Scooter motor speed controller. The circled AC signal is where the Hall effect sensor can be read.

The hall effect sensors can be accessed via the scooter controller. The voltage at the hall effect sensor port varies from 0.8V to 4.2V. This signal can be converted into a pulse through a zero crossover circuit. The frequency of these pulses can be interpreted to calculate the speed of the scooter.