BikeBuddy

Authors: Jack Wang, Jason Lu, Johnny Tian
Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—BikeBuddy is intended to improve safety for bicyclists by allowing them to have better situational awareness and giving other road users a better understanding of cyclists' intentions. This is accomplished through a waterproof system incorporating microwave radars that sense vehicles located behind, on the side, and in front of the bicycle along with LED turn signals with auto cancellation. An embedded computer integrates information from the radars and provides visual alerts to the bicyclist on a centrally mounted display. The system achieves a 3% false negative and 26% false positive rate while having 95% distance detection accuracy.**

***Index Terms*—Bicycle, Embedded Systems, Radar, Raspberry Pi, Safety, User Interface, Vehicle Detection, Magnetometer**

1 INTRODUCTION

Bicycling is a healthy and environmentally friendly way to travel. Cycling has become more popular over the past years. However, the safety of cyclists on the road remains a significant problem, especially in the congested metropolitan areas. Bike commuters need a better safety system to protect them when sharing the road with larger vehicles. A waterproof bicycle safety system with blind spot detection, rear and forward obstacle distance sensing, and turn signaling with automatic cancellation will improve bike commuter safety through increased cyclists' awareness of their surroundings and vehicle awareness of cyclists' intentions. The alerts can be displayed on a mounted screen, enhancing the situational awareness of the cyclists.

There are some existing technologies for bike safety, including mirrors, Garmin Varia RTL510, and wearable devices. A group in 2019 [6] did a similar project with LiDAR and a safety vest to improve bike safety, and another team in 2021 used microwave radars [2] for blind-spot detection and had turn signals. Compared to the team that used LiDAR, Our project used radar for blind spot and range detection, enabling all-weather operation. Also unique to our project compared to the other capstone projects, the screen display centralizes the information in one place for cyclists. The overall goal of the system is to provide bike commuters with an easy-use system with better detection accuracy to enhance their safety.

2 USE-CASE REQUIREMENTS

To fulfill the functionality of the system, the following design requirements are proposed:

- Cost: The final system should cost $\leq$ \$200
- Battery Life: The system should have at least 2 hours of endurance. According to data from Strava [1], the average commute distance in the U.S. is 8.3 mi (ca. 13 km). Assuming a biking speed of 15 mi/h (ca. 24 km/h), a 2-hour endurance allows bikers to use the system for a day of commute with buffer time.
- Detection Lead-Time: The system should give users enough time to react. The human response time is between 100 ms and 300 ms. Some studies have found that people need approximately 1.5 s response lead time to react to road hazards [19]. So, the system should give a warning at least 1.5 seconds before collision.
- Uptime: $\geq$ 99.999%.
- Confusion Matrix:
  - $\leq$ 40% False Negative
  - $\leq$ 30% False Positives

  The result from the 2019 project [6] had a false negative rate of 60% and a false positive rate of 41%. The confusion matrix is set this way so that our system performs at least 20% better than the previous version.
- Ruggedness Rating: IPX4. The device should work for commuters who commute in rainy conditions. IPX4 means no rating for protection against solids and "protection from sprays and splashing of water in all directions" [11].
- Turn Signal Visibility: Minimum of 100 feet of visibility in daytime, 500 feet visibility after sunset. Pennsylvania law requires a red rear reflector or light to be visible from 500 feet away between sunset and sunrise [8]. Although our turn signals do not fall under that law since they are not red, it sets a reasonable baseline requirement for nighttime visibility. The 100 feet (ca. 30 m) is a little more arbitrary, but it should be sufficient for drivers behind to see and react in time.

Note: We dropped the Ease of Installation requirement from the original design report since it would've been excessively difficult to target this requirement for our prototype. However, this requirement should still be considered for the product that enters the market.

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION

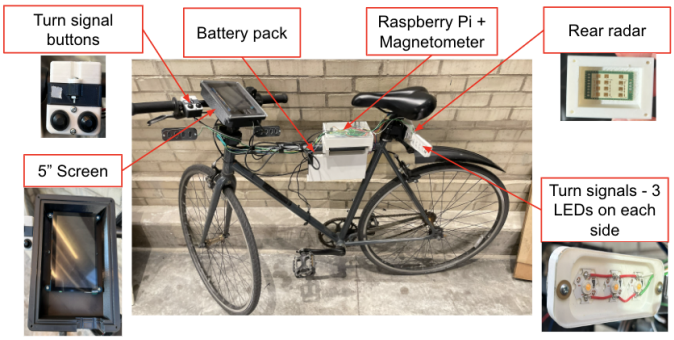

Figure 1: Overall System

For readability, we have placed the block diagram as Fig. 10 at the end of this document.

At a high level, our system consists of two functional areas integrated into one system: (1) Obstacle detection and (2) turn signaling. The high-level system setup remained similar as mentioned in the Design Report, except we have changed "Vehicle Detection" to "Obstacle Detection" to generalize it more from just vehicles to other things on the road like cones.

3.1 Obstacle Detection

The obstacle detection subsystem is responsible for detecting obstacles around the bicycle and displaying their distances if in front or behind the bicycle or indicating if a vehicle is in an adjacent lanes (for blind spot monitoring).

The sensing portion consists of two radar modules, one mounted on the front of the bicycle and one mounted on the rear of the bicycle. The front module is responsible for tracking objects ahead, and the rear module monitors objects both directly behind and in the adjacent lanes for blind spot monitoring.

Both radar modules are Doppler radars, which allow them to sense the speed and direction of objects in addition to distance and angle. Doppler radars work by measuring the time it takes for radar pulses to return, along with sensing the phase shift in the return signal due to the movement of the object [21] [17].

One change from the Design Report is the segmentation strategy of the rear radar. Since we are using only a single radar to cover the entire rear area, it requires that we identify whether vehicles are in the same lane as us or in adjacent lanes to provide the appropriate visuals.

Our previous approach consisted of slicing the radar's field of view (which is a semicircle shape) into sectors based on the angle of the object, as we proposed in the Design Report and is shown in Fig. 2.

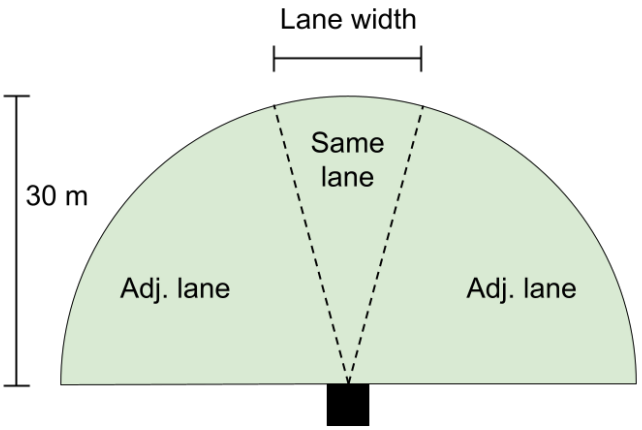

Figure 2: Early rear radar segmentation scheme: the black box is the radar and the green semicircle is the radar detection zone (not drawn to scale), and each zone is labelled with what lane a car within it would be considered

For our final strategy, we chose a more straightforward approach where we simply decide which region a car is in by its horizontal distance from the bicycle (e.g., how far it is to the left or the right of the bicycle), with $x = 0$ centered on the bicycle. Objects to the left behind the bicycle will have negative x values, whereas objects to the right will have positive values. If the objects are within ± 0.5 m (for the rear in demo mode, in non-demo mode it is within ± 1 m) horizontally of the bicycle, we consider it to be behind the bicycle and therefore it will trigger the rear range indicator. If instead the objects exceed those thresholds, they will be treated as if they were in adjacent lanes and will trigger blind spot indicators instead. This is visualized in Fig. 3, although note that the direction of travel is downwards so left of the bicycle is actually on the right side of the diagram.

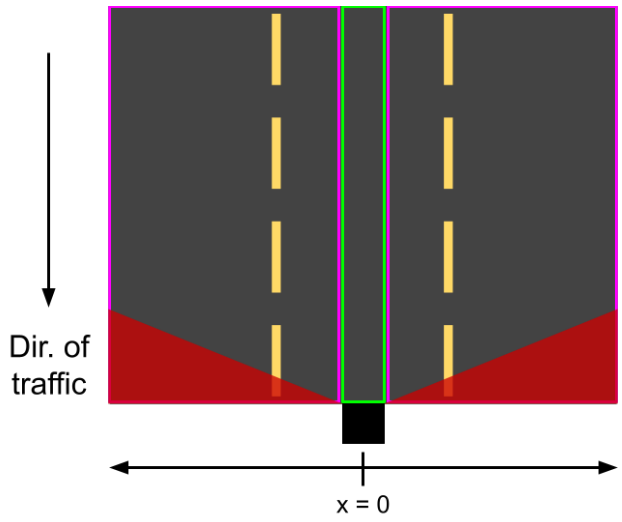

Figure 3: Final rear radar segmentation scheme: The black box is the radar, mounted at the rear of the bicycle. The yellow markings delineate lane markings on the ground, the green box indicates the area we consider to be "rear" of the bicycle, the purple boxes indicate the area we consider to be in the blind spot, and the red triangles indicate occlusion of the radar detection due to the enclosure physically blocking radio waves. Note that the radar detection zone is a semicircle shape, so the actual distance detectable by the radar varies.

The radars provide a list of detected targets and their information to the Raspberry Pi, which filters the data and then displays them in the appropriate manner (e.g., lighting up a blind spot indicator if a car is approaching on the side).

To display warnings, we also have a small screen mounted on the handlebar of the bicycle that is connected to the Raspberry Pi. The display will flash colors in various parts of the screen to indicate danger and warnings.

3.1.1 Forward and Rear Collision Warnings

In our original design report, we described forward and rear collision warnings (sections 3.1.1 and sections 3.1.2). For forward collision warnings (FCW), like in a car, the system would alert you to brake if you are in danger of colliding with the car in front of you. For rear collision warnings (RCW), the system will warn you to move out of the way if it determines the car behind you is moving too fast to brake in time and might hit you.

We implemented these features in our prototype, but we eventually disabled them because they were too noisy and triggered too often. In addition, they would often block other parts of the interface that we wanted to see. We believe that one cause is that the equations used to estimate the stopping distances required for both the bicycle for FCW and the car for RCW were too conservative, so they would trigger collision warnings too often. We also should've implemented a feature that filtered out objects that were traveling less than a certain velocity, such as 2 m/s, to prevent the case where just walking around the bicycle close would trigger alerts falsely.

3.1.2 Front and Rear Range Indicators

Despite the removal of FCW and RCW, we are still able to give the user information on the distance of objects in front and behind them as before. We utilize the distance data provided by the radars and visually indicate their distances on the screen. This only applies to objects within the region considered "behind the bicycle" and "in front of the bicycle", so objects within adjacent lanes will not trigger these indicators.

3.1.3 Blind Spot Monitoring

For blind spot monitoring, any cars in adjacent lanes (as described above) will trigger a blind spot indication regardless of their distance from the vehicle. This is intended to warn the user if a car is coming up on your side so you don't switch lanes and collide with them, just like how a blind spot indicator on your car mirror would warn you.

Unlike in a car, however, we warn the user through visual indications on the display rather than indicators on a mirror.

3.2 Turn Signaling

The turn signaling mechanism consists of the turn signal buttons that are located in the front of the bicycle, the auto-cancellation system located in the middle of the bicycle, along the actual turn lights located in the back of the bicycle.

For the turn signal controls, we have two momentary switches mounted on the handlebar of the bicycle. Pressing on the momentary switch will close a circuit to cause a voltage change that is detectable by the Raspberry Pi, causing it to turn on the turn signals. The turn signals can then be turned off again by pressing either of the momentary switches.

Additionally, for user convenience, the system will also include an auto-cancellation system that is intended to automatically deactivate the turn signals once a turn is completed, much like a car would. This is implemented using a magnetometer mounted on the center of the bicycle.

The magnetometer is used as a compass to give us the heading of the bicycle, and the idea is that we'll record the starting heading of the bicycle when the turn signal is first activated. Then, once the heading of the bicycle relative to the starting heading exceeds a certain threshold, we'll turn off the turn signals.

The turn signals themselves are simply LEDs mounted in the rear of the bicycle, with transistors controlled by the Raspberry Pi.

3.3 Principles of Engineering

We wanted to build a system that was reliable and simple, and we needed to iterate the design multiple times to

adjust any issues.

3.3.1 Source Code Management

We used Git as our source control system, hosted in a Github repository. This allowed us to keep track of changes and easily be able to revert some changes if they break something.

3.3.2 Radars

For radars, we broke down the design into small phases. We started by making sure of the basic functionality of the radar using simple wire connections and breadboard. We tested the functionality by using plotting and printing before moving on to implementing the detection algorithm. During the implementation, we ran tests to identify the detection outputs. We had to change the segregation of the radar to make the system simpler to ensure a more reliable output.

3.3.3 Hardware

For the electrical system, we needed to read the datasheet of each component to ensure the designed circuits wouldn't cause any piece of components to overload themselves when creating the schematics. In addition to checking numbers and calculating, we also did empirical tests to ensure the numbers given were correct, such as the LED turn-on voltage, LED on-chip resistor resistance, transistor threshold voltage, etc.

During hardware testing, we used 3D printing and laser cutting to rapidly prototype our parts to quickly iterate and update hardware design to ensure the enclosures can meet designed use-case requirements.

3.4 Principles of Science

When analyzing the data, we did multiple trials for different tests to ensure the accuracy of the data. In addition, we went beyond the design requirements to explore possible correlations between two parameters. We interpreted the data in a scientific way, providing a reasonable explanation for possible deviation from the expected value.

3.5 Principles of Mathematics

We have used numbers and algebraic calculations to calculate the expected results of the system, as well as using them to validate our approach. For example, given the power bank that we have, we used formulas to calculate the requirements of the power consumption of the system (see 4 for more details). In dealing with MOSFET, we used numerical analysis to determine if we were over the max current allowance of the MOSFET (see 6 for more details).

4 DESIGN REQUIREMENTS

4.1 Power Limitations

The users would like a solution that will last long enough for their daily commute, so this requires that the design meet certain power usage limits.

Given a power bank of 26 800 mA h and a target run time of at least 2 h, we have a maximum current draw of:

$$I = \frac{26\,800\,\text{mA h}}{2\,\text{h}} = 13.4\,\text{A} \tag{1}$$

However, the specific power bank we're using, the Anker 337, only supports a maximum of 3 A per USB port and a total of 6 A. Therefore, our entire system must draw less than 6 A, with each system attached to a USB port drawing less than 3 A. That way, we can meet both the battery electrical limitations and the total runtime requirement.

4.2 Simultaneous Targets

The rear radar needs to be capable of tracking at least three separate vehicles in three different lanes, one for a car directly behind the bicycle and one car each in the adjacent lanes.

We previously had a minimum distance resolution sub-requirement of 9 feet since a typical car lane width in the U.S. is 9-12 feet for urban roads [22]. Minimum distance resolution refers to how close two objects can get to each other before the radar cannot distinguish between them. However, this sub-requirement no longer makes sense due to the way we're segmenting the rear radar zone which doesn't depend on road lane width nor being able to identify vehicles from each other.

4.3 Data Transmission

The radar must be capable of communicating with a baud rate of 115200 for UART communication with the Raspberry Pi.

4.4 Radar Accuracy

The radar shall provide a minimum accuracy of $\pm 10\,\%$ for all quantitative measurements (speed, distance, angle).

For velocity, it shall identify the correct direction of the target $\geq 95\,\%$ of the time.

4.5 Radar Data Update Frequency

The radar shall provide location updates for all detected vehicles at a minimum frequency of 10 Hz.

4.6 Detection Distance

The system should have a minimum detection range of 14 m. Section 2 describes that the system should provide 1.5 s response lead time for blind spot collision.

Assuming a speed differential of 15 mph between the car and the bicycle:

$$15\,\text{mi/h} = 24\,\text{km/h} * \frac{1000\,\text{m}}{1\,\text{km}} * \frac{1\,\text{h}}{3600\,\text{s}} = 6.67\,\text{m/s} \quad (2)$$

$$6.67\,\text{m/s} * 1.5\,\text{s} = 10\,\text{m} \quad (3)$$

Therefore, the system needs to be able to detect objects that are 10 m away to fulfill the requirement. Adding 30% buffer, we need to ensure the detection range of the radar will be at least 14 m.

5 DESIGN TRADE STUDIES

5.1 UI Framework

Here are some options that we considered:

1. Electron
2. Qt
3. Gtk
4. Tauri - this was added during testing because of the high memory usage of Electron, and we wanted something conceptually similar to it but that had hopefully lower resource usage.

Here are the criteria that we used to evaluate the frameworks. Note that not every factor was weighted equally:

1. **Cross-platform** – Can it run on both MacOS and Linux?
2. **Language/Framework/Tooling Familiarity** - How familiar are already with the language, frameworks, and tools?
3. **Adoption** – Who are the major users of it?
4. **Baseline resource usage** – How much CPU usage and RAM do we use to render roughly the same things?
5. **Cross-compileable** – Can we easily build packages for the RPi 4 from a host computer?

Sample "Hello world" apps were created in all 4 frameworks and their memory and CPU usage were measured. Testing was conducted on a M1 MacBook Air running MacOS Sonoma (14.2.1) with 16 GB RAM.

The memory usage was obtained by using Activity Monitor (like Task Manager, except for MacOS) and summing up the relevant threads' memory usage - e.g., for Electron we have multiple threads running around so we need to account for all of them. For calculating CPU usage, we used an admittedly lot less scientific method where we just watched the CPU usage of the threads and a value as the max only if the CPU usage repeatedly hit it – e.g., hitting 1% CPU usage just once wouldn't count, only if the usage fluctuated up to 1% a few times.

Table 1 contains a summary of our testing, and a full version with citations can be found here.

For Tauri, we were unable to find any major applications that we knew of that used it, which made us a little more hesitant about whether it would be as production-ready and usable for real-world apps as the other three. Therefore, we ruled out considering it further.

In the end, the decision came down to performance overhead vs. ease of development. Given that we were most familiar with HTML/CSS/JavaScript for building UIs and were comfortable using HTML/CSS/JavaScript (equally so with C, more than C++), we'd rather use Electron despite the performance overhead simply because the lower learning curve is more important to us.

5.2 Embedded System

Another decision we had to make was the choice of the device to use as the compute module. Our final two contenders were the Raspberry Pi 4 and 5. Some reasons why we were focusing on Raspberry Pis is because they are available in the ECE inventory (hence zero cost for us) and have good documentation and resources.

The Raspberry Pi 5 features significantly better performance compared to the Raspberry Pi 4 (2 - 3x on many benchmarks according to [27]). However, in exchange, the peak power draw for the Pi 5 is 12 W vs. only 8 W for Pi 4, and in one set of benchmarks the Pi 5 drew nearly double power at idle and under load [4]. This can be an issue for us as battery life endurance is one of the requirements, and having excessive power draw can make hitting that difficult.

There were a few other minor considerations such as the Pi 5 running hotter [4] and the USB max current being limited on the Pi 5 to 600 mA compared to 1.2 A limit on the Pi 4 using the same 3A charger [4] [15]. However, in the end, the main tradeoff was between performance and power draw.

We felt that the Pi 4 would have enough performance to run our workload, so the additional performance was not necessary. Therefore, we settled on the Pi 4.

5.3 Radar

Many sensors can provide distance information between two objects, aiding blind spot detection and collision detection. Based on the use-case requirements described in 2, we are interested in the following performance of a sensor:

- All-weather operation
- Line of Sight Requirements
- Range
- Accuracy

The following sensors are considered:

- Radar
- LiDAR

Table 1: Comparison of various UI frameworks according to our metrics

	Cross-Platform	**Language + Framework Familiarity**	**Adoption**	**Baseline Resource Usage (Memory/CPU)**	**Cross Compilable**
Electron	Yes	We are familiar with Javascript, HTML, CSS, and some JavaScript UI frameworks, but otherwise no prior experience with Electron	Used by many major apps, like 1Password, VSCode, Slack, and Discord	106 MB/0.1%	Yes
Qt	Yes	We are less familiar with C++ and Python, no prior experience with Qt. It seems as if you don't need to use Qt tools, but it's recommended and we aren't familiar with it	VLC , FreeCAD , many KDE apps	19.8 MB/0.0%	Seems to be possible but very complicated
Gtk	Yes	We are familiar with C, but otherwise no prior experience with Gtk	Firefox , GIMP, Transmission, GNOME desktop	22.6 MB/0.0%	Seems to be possible but complicated
Tauri	Yes	We are familiar with Javascript, HTML, CSS, and some JavaScript UI frameworks, but otherwise no prior experience with Tauri	No known major apps using it	74.6 MB/0.0%	Currently not possible

- Ultrasonic Sensor

In order to support the usage of the system in rainy conditions, the detection sensor will be covered by a waterproofed cage. This means that we would require a sensor that cannot be affected by ambient conditions. Due to the enclosed cage, the sensor should also not be limited by direct line of sight. Radar has advantages over LiDAR as radar signals will be able to better penetrate weather, allowing detection in bad weather conditions [18] [9]. Furthermore, radar can penetrate solid materials, whereas it is unlikely that LiDAR can pass through opaque materials, which limits our enclosure material choices.

The range of the detection sensors needs to be greater than 10 m per the use-case requirements. Both LiDAR and radar can achieve this range. Ultrasonic sensors, although they might be more cost-effective, typically have a shorter range than what we require.

While accuracy is comparable, LiDAR typically has very high precision in measuring distance, especially in the short to medium range, whereas radar has a slightly lower precision [9].

Combining all the factors above, radar stands out as the most ideal detection sensor for this system. We have chosen K-LD7 radar as the module because it would provide direct serial output, reducing the workload of signal processing. However, one thing to note is that the K-LD7 will not register a target if the radar and the target are relatively stationary [23]. For the specific case of bike blindspot detection, it is very unlikely that the vehicle behind the bike will travel at the same speed as the bicycle. So, it is considered reasonable to sacrifice such drawback for the ability to detect in all weather conditions.

5.4 Turn signal and Auto-Cancellation

Since we're planning to implement turn signals that users can control with auto-cancellation, purchasing off-the-shelf flashing lights from Amazon or other sources is not feasible. The uncertainty of the wiring diagrams and the complexity of figuring it out is comparable to designing entirely new turn signals. Consequently, we explored motorcycle signals, often sold as LED chunks. However, our research revealed that common lightweight motorcycle turn signal lights operate on a 12V power supply, typical for motorcycles. Introducing motorcycle LEDs and a DC-to-DC converter adds another layer of complexity. Therefore, we've opted to purchase bright yellow LED chips to craft our custom turn signals.

Regarding the turn switch, our initial plan was to buy motorcycle turn switches, primarily for easy momentary activation and convenient handlebar mounting. However, an aftermarket motorcycle switch we acquired from Amazon turned out to not be momentary, meaning it doesn't return to neutral after activation, causing it incompatible with the auto-cancellation feature. As a solution, we're purchasing

waterproof momentary switches and customizing our turn switches.

For the auto-cancellation feature, our original idea was to use a potentiometer as a handlebar position sensor, considering the option of utilizing gearing to amplify turning for more drastic readings. However, through observations, we noticed that during bike rides, the handlebar rarely turns more than 5 degrees on each side when making a turn. Additionally, the maximum amplification achievable through gearing would be at most three times, increasing the required torque. This compounds the existing issue of high-resistance potentiometers. Therefore, our current plan involves using a magnetometer mounted on the Raspberry Pi to measure the bike's heading and implement auto-cancellation based on the entire bike's dynamics.

6 SYSTEM IMPLEMENTATION

6.1 Turn Signal

The schematic is shown in Fig. 4

6.1.1 Turn Signal Lighting

Our solution integrates 3 5V 1W LEDs (Hyunduo 5V 1W 200lm Yellow), each emitting 200 lumens of vibrant yellow light on each side of the bicycle (front left, front right, rear left, rear right) for a total of 12 LEDs onboard the bicycle. This configuration ensures signaling capabilities both at the front and back of the bike, improving visibility for bikers and making them more noticeable by other participants on the road.

To control the power supply, we will send control signals to the gates of N-MOSFETs (BS170 from the 18-220 lab) to regulate the power on and off from a separate 5V 3A USB port on the power bank rather than powering them from the RPi's 5V pins.

Originally, we were expecting a total current draw of 2.4 A combined from all the LEDs, assuming no resistor in place:

$$I = 12 \times \frac{1\,\mathrm{W}}{5\,\mathrm{V}} = 12 \times 0.2\,\mathrm{A} = 2.4\,\mathrm{A} \tag{4}$$

This would have far exceeded the maximum current output from Raspberry Pi, so that's why we used the combination of MOSFETs and a separate power source to illuminate the LEDs. However, it turns out that with resistors inline, the current draw drops down to a peak of 200 mA which means we could theoretically power the LEDs from the Pi. However, we decided to stick with powering the LEDs from a separate USB port.

6.1.2 Auto-Cancelling

To improve the user experience, we're implementing turn signal auto-cancellation, similar to how cars will automatically deactivate your turn signal once you complete a turn.

We used a magnetometer (Adafruit MMC5603) as a compass in order to sense the heading of the bicycle. The magnetometer itself is mounted on the center of the bicycle, so as the entire bicycle turns the heading will change. This is preferable to mounting it on the handlebars because when stopping at a traffic light, some bicyclists turn their handlebars to the side. We do not want to deactivate the turn signals when they rotate it back to straight to continue moving as they haven't finished (or started for that matter) turning.

Using a magnetometer as a compass was not as trivial as we thought. In the end, we also needed to perform compass calibration (as described in Jason's status reports) to get the system to work reliably. The code used for calibration is primarily derived from a Jupyter notebook provided by Adafruit [13].

The magnetometer is controlled by a Python script running onboard the Pi which reads the magnetic fields, applies calibration values, and then computes the heading. The heading is then sent to the user interface program for further use.

When the turn signal is first activated, the current heading of the bicycle is recorded as the start heading. Then, the heading of the bicycle is monitored at a rate of 10 Hz, and if it exceeds 30° in either direction of the starting angle, the turn signals will be deactivated. Note that this 30° is as measured by the magnetometer, but due to inaccuracies in the magnetometer, the actual turn angle required does deviate from this.

6.1.3 User Control

The user will control the turn signals through two waterproof buttons (Twidec Momentary Push Button) that are enclosed in a custom enclosure and communicate with the Raspberry Pi. The buttons are directly wired between GND and GPIO pins in the Pi which have the internal pull-up resistors active. Therefore, when the switches are not pressed the Raspberry Pi will read a logic high, but when pressed the pin will read a logic low. The software on the Pi will detect this and trigger a turn signal action.

Upon the initial button press, the appropriate LEDs will begin blinking and an indicator will also blink on the screen, similar to how a car has the turn signal indicator blink on the dashboard. As mentioned above, if the bike's orientation deviates by around 30° from the starting direction during the turn, the turn signal will automatically deactivate. In cases where the user intends to change lanes using the turn signal, they can press either of the two buttons to cancel the signal.

6.2 Radar

6.2.1 Hardware

The radar used for blind spot detection was mounted at the back of the bike, facing the rear. The radar used for front range indication was mounted in the front of the

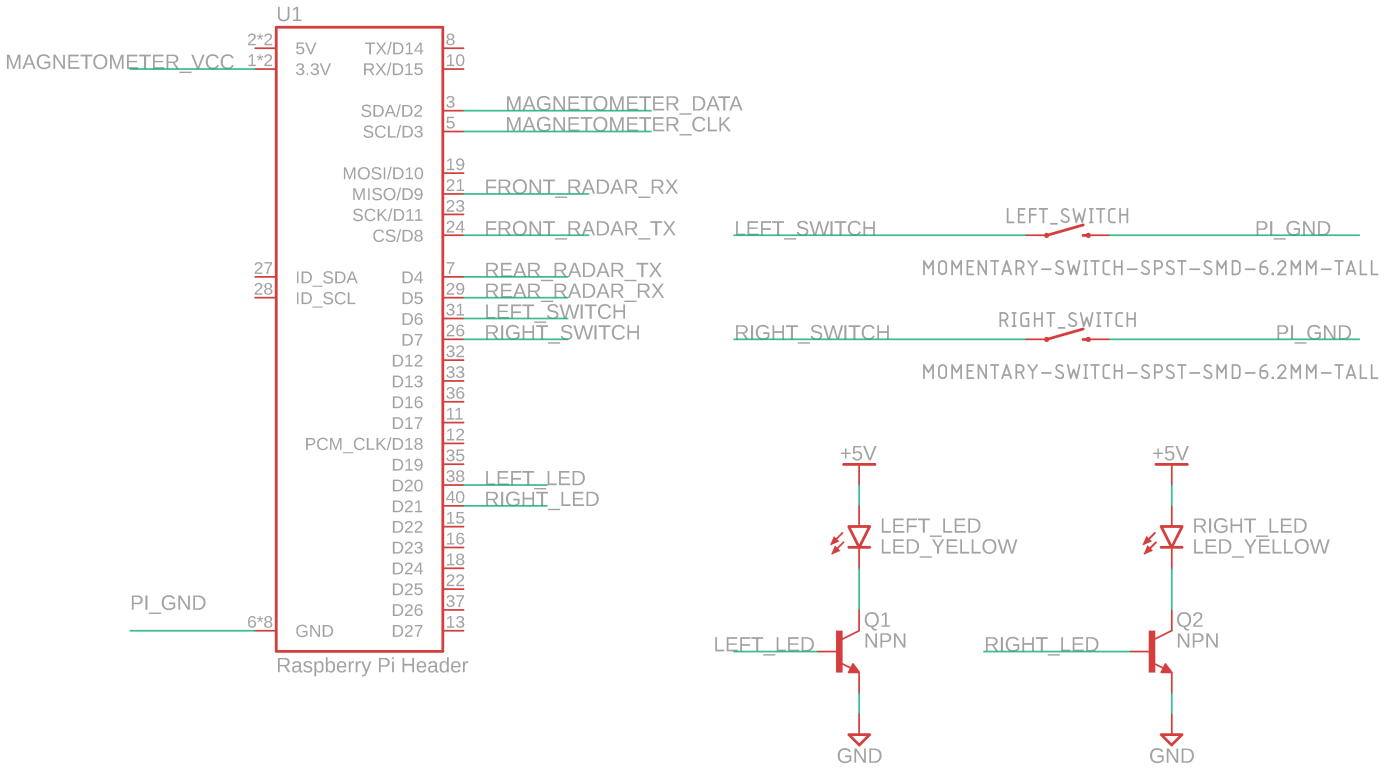

Figure 4: Turn Signal Schematic[3][10][5][26]

bike, facing front. Both radars were enclosed by a waterproof box, which was to ensure the all-weather operation requirements. The radars were powered by the 5V output from the Raspberry Pi. The Tx and the Rx pins of the radar were connected to the UART Tx and Rx pins of the Raspberry Pi. All the wiring of the radar went between it and the Raspberry Pi.

6.2.2 Software

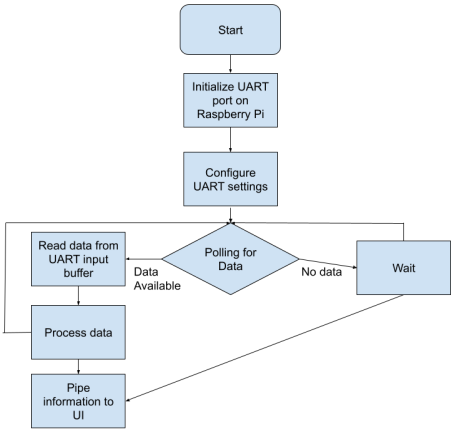

Figure 5: Radar Flow Chart

The basic hardware setup remained unchanged for the most part. After the initialization of the radar, however, it did not work. We went to the datasheet to realize that the radars required even parity pits for UART, but the default UART on the Raspberry Pi 4 did not support even parity. We had to consult the datasheet to find the UART ports that support even parity bits. We used UART2 and UART3 on the Pi at the end.

The radar sensor communicated with the Raspberry Pi over UART. We opened the UART port and used the protocol 2,000,000-8E1. A Python driver was available to interface with the K-LD7 radar module. However, since we only used some functionalities of the radar, we did not need to rely on the driver. The code was adapted from some sample codes from RFbeam on how to send the correct bits to initialize the radar [25]. 5 shows the flow chart of communication. We were particularly interested in distance range,

max velocity, and baud rate parameters for our detection purpose.

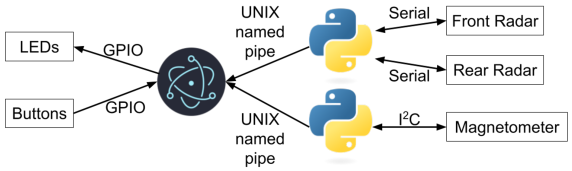

Figure 6: Software Block Diagram

In addition to the UART issue described in the previous section, other challenges with the radar software setup included de-noising the targets and sending them properly to the UI interface. This is necessary because a single vehicle can show up as multiple radar targets [12]. During the first attempt, we plotted all the targets that were detected to observe the detection results. However, it turned out that there was a lot of noise in the data, where some data points would get dropped across different frames. To solve this issue, we discussed and wrote different algorithms on the whiteboard to de-noise the results. The decision was to sample all the data points across three consecutive frames. Only the targets that appeared in all three frames would be treated as an obstacle and sent to the interface. We discovered that we could use named pipes to send the information between the radar processing script in Python and the UI implementation in JavaScript. We packed the data into JSON format and did some basic testing to verify the communication.

6.3 User Interface Software

Our Raspberry Pi 4 runs off a standard Raspberry Pi OS (which is based on Debian Linux [14]) installation, created using the Raspberry Pi Imager.

Using Linux instead of an RTOS or other custom operating system gives us access to pre-written drivers for the various peripherals (especially the display) and access to more traditional desktop graphical frameworks. This simplifies bring up significantly.

The user interface itself is built using Electron which “is a framework for building desktop applications using JavaScript, HTML, and CSS ... that work on Windows, macOS, and Linux — no native development experience required” [7]. This allowed us to initially start developing on a Mac and switch over to developing on Linux later without having to rewrite anything.

Since working with just pure JavaScript, HTML, and CSS can be a little difficult to accomplish what we want to build, we used a framework called React. React actually uses JSX which is JavaScript with syntax extensions [24], but it allows us to accomplish everything we need to do including things like styling, displaying live sensor data, and handling user inputs.

We initially designed a mockup of the UI in a software called Figma. The final UI looks like what is depicted in Fig. 7.

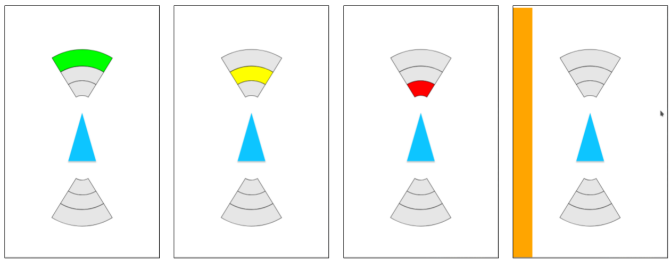

Figure 7: From left to right, the images show the visuals indicating a car in front at (1) far distance (2) medium distance (3) close distance (4) a car on the left side behind the bicycle

To display the range of cars, we have what are called ”range indicators”, one for the front of the bicycle and one for the rear. The range indicators consist of three arcs, each colored a different color. The furthest arc is green corresponding to if a car is far from you. The middle one is yellow, and the closest arc is red indicating that a car is close to you. This design is inspired by Toyota’s parking assist proximity sensing system which can be seen in the ”Distance Display” section of the 2021 Toyota Venza manual.

The bicycle itself is represented by a blue triangle, but it’s really just there to separate the front and rear range indicators.

The blind spot indicators are orange vertical bars that appear on the side of the screen when a car is in an adjacent lane. The bar will appear on the side that the vehicle is approaching.

Lastly, the turn signals themselves are indicated by blinking turn signal icons that are copied from Tesla’s turn signal design which can be seen in the Tesla Model Y manual. They only appear when the corresponding side’s turn signal is activated.

Overall, the user interface software is responsible for quite a bit of functionality of the system. Perhaps most importantly, it displays the relevant information to the user. Internally, it receives data from Python scripts that communicate with the radars and magnetometers and displays them in an appropriate manner. And as mentioned above, it also handles turn signal button presses and actually controls the turn signal LEDs too.

To ease in testing the user interface’s logic and behavior before all the hardware was ready, we built a debug system that allows us to simulate radar data and turn signal button presses. In addition, it also displays live data such as the radar data and the turn signal status. A screenshot of it is shown in Fig. 8

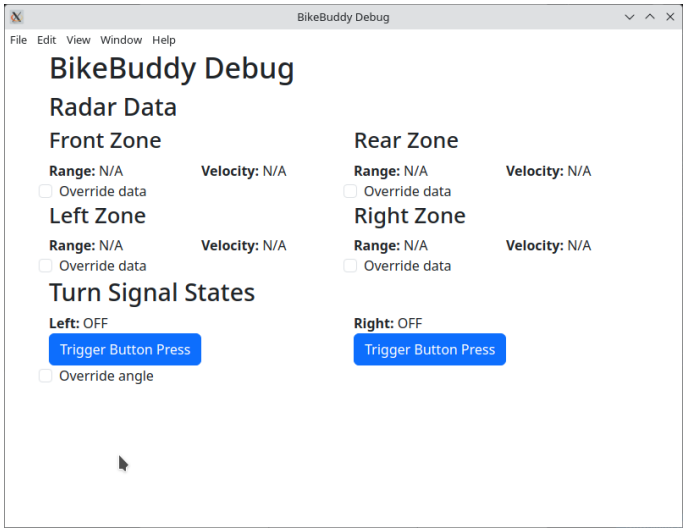

Figure 8: A screenshot of the debug window for the user interface.

7 TEST & VALIDATION

For most of the design specifications, we completed unit testing and integrated real-world testing. Unit testing involved evaluating the device indoors without installation on a bike and conducting tests outdoors with the device installed on a stationary bike. Real-world testing entailed assessing the device in actual traffic conditions, with video recording for subsequent analysis.

7.1 Power Consumption

Power consumption is a key component of our project. Our goal is to ensure that our system can continuously operate for more than 2 hours, so the total power consumption needs to be less than 6 A given the requirements stated previously.

After the system was installed, we conducted the test by placing ammeters between the outputs of the battery pack and RPi 4 and the LEDs. During the test, we made sure all the LEDs were turned on and radars were actively detecting to simulate a worst case workload. We then summed the current draw of the RPi (and its peripherals) and the LEDs to get the total power consumption. We took the peak current of the system and the average current of the system over 100 samples. Table 2 shows the collected information.

The result indicated that the entire system satisfied the overall power consumption requirements. Furthermore, as the RPi and LEDs each occupy a separate USB port, we easily met the 3 A per-USB port current limit.

7.2 Battery Life

The system was able to run well beyond 2 hrs. During demo day, we powered the system from the battery pack the entire time without recharging.

7.3 System Uptime

The system was up the entire 3 hour period of demo day, which is 100%. This satisfied the up-time requirements.

7.4 Data Transmission

The radar was capable of communicating with a baud rate of 115200 (2,000,000 in reality) for UART communication with the Raspberry Pi.

7.5 Radar Accuracy Baseline

These tests aimed to benchmark the performance of the radar, noting the possible offsets between the real performance and the datasheet. The following tests were performed:

- Distance Accuracy: The radar was placed at a stationary location, and a car was placed more than 30 meters always from the radar. The car stopped at 5 different locations from the radar. The distance output of the radar was recorded and the distance between the car and the radar was measured using a tape measure. The results are shown in Table 3.

  The average error in the distance is 5.44%, which is within the requirement of $\leq$ +/-10% deviation, so the radar met the accuracy requirements with distance measurements.

- Distance-Error Relationship: This test was not required by our design requirements. However, we were curious about whether there was a relationship between the distance accuracy and the distance of the object. The following graph was generated from the data in 3

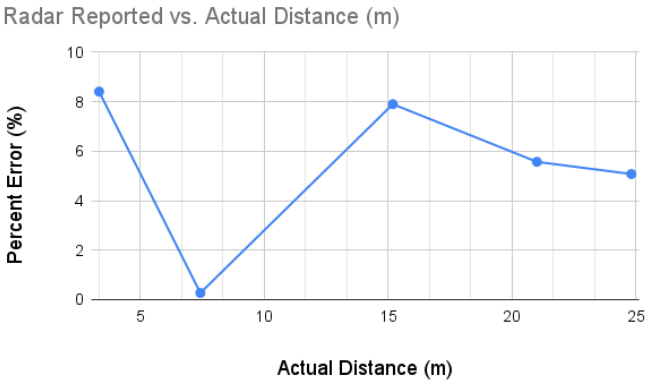

Figure 9: Radar Error vs. Distance

As we can see from the graph, there was no particular relationship between the error of the radar and the distance.

Table 2: Current Draw Data

Device	Peak Current (A)	Avg. Current (A)
RPi	1.440703	0.947011
LEDs	0.199674	0.19727
Total	1.640377	1.144281

Table 3: Distance Accuracy

	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**	
Reported (m)	19.84	23.56	13.99	7.40	3.61	
Actual (m)	21.01	24.82	15.19	7.42	3.33	
Difference (m)	1.17	1.26	1.20	0.02	0.28	
Percent Error (%)	5.57	5.08	7.90	0.27	8.41	
Average (%)						5.44

- Speed: The radar was placed at a stationary location, and the car was parked 30 m away from the radar to start. The car traveled towards the bicycle at 5 mph based on the speedometer on the car. The output velocity of the radar was recorded. The test was repeated 3 times and an average was taken. The results are shown in Table 4.

  The average speed error of 7.00 % is within the requirement of $\leq$ +/-10% deviation, so the radar met the accuracy requirements with velocity measurements. Note that we did not measure the actual speed of the car using something like a radar gun, so it's possible the vehicle's speed deviated from 5 mi/h.

  Although we did not specifically test the requirement for correctly identifying the direction of the velocity, the results from the speed test matched the direction of the travel of the car.

- Maximum Range: The radar was placed at a stationary location, and the car was parked 30 m away from the radar to start. The range of rear detection and the range of side detection were both tested.

  1. Rear Detection: The car was aligned to the center of the radar and driven at a constant velocity towards the bicycle. The data point at which the car was first reliably detected was recorded. The test was conducted 3 times and the results were averaged.

  2. Side Detection: The car was aligned to the right side of the radar and driven at a constant velocity towards the bicycle. The data point at which the car was first reliably detected was recorded. The test was conducted 3 times and the results were averaged.

  We required the object to be detected from at least 14m from the radar. The max range in this case was 25.14 m, which satisfied the requirement. The side detection showed up at around 18 m, but the target was not very reliable, so these data points were chosen as they were constantly showing the target. We also observed that, for this specific unit test, the farther the car, the more likely it would be picked up by the rear region.

- Data Update Frequency: No explicit data was recorded for this test. However, from observation, the users were able to get updates for obstacle information in a timely fashion.

7.6 Detection Lead Time

The detection lead time was tested by putting the bike that contained the system on the side of the road. We started a timer when the blind spot detection warning was triggered and stopped the timer when the car passed the bike. The results varied between 1 second and 2 seconds.

Since the radar detection is based on distance, the detection lead time would be affected by the speed differences between the moving object and the bike. In the requirement, we assumed a 15 mph speed difference, which yields a 1.5 s lead time requirement. The road environment that we tested on would have a speed difference of close to 30 mph. So this was supportive evidence for sufficient detection lead time.

7.7 Simultaneous Targets

This requirement was tested by having 3 people running towards the radar. All three targets were able to be detected.

7.8 Radar Accuracy Confusion Matrix

The confusion matrix was computed based on the data of the real-world testing in this video. In the test, we rode the bike on the street while recording the output from our system and the ground truth of the actual surroundings. In

Table 4: Velocity Data

	Trial 1	**Trial 2**	**Trial 3**	
Velocity (m/s)	-1.844444407	-2.330555651	-2.061111132	
Average (m/s)				-2.07870373
Average (mph)				4.649927815
Error (%)				7.00

Table 5: Max Range Data

	Trial 1	**Trial 2**	**Trial 3**	
Max Range (Rear)				
Range (m)	25.30	24.87	25.24	
Average (m)				25.14
Max Range (Right)				
Range (m)	12.13	11.79	11.60	
Average (m)				11.84

the 2.5-minute test, we rode the bike into different traffic conditions that a rider might be in, such as in the bike lane and making a turn in the turn lane. We compared the video and the output from our system and recorded the total time of the occurrence of false positives and false negatives.

It yielded a false positive rate of 25.83 % and a false negative rate of 3.26 %, which were within our requirements of $\leq 40\%$ False Negatives and $\leq 30\%$ False Positives.

7.9 Waterproof

We have initially planned on performing IPX4 waterproof testing on our system. However, due to the complex testing procedures of IPX4, we decided to modify the test to something simpler yet still partially mimics a real-life raining situation.

To test our waterproof system, we decided to make use of the on-campus sprinkler system. Once part of our bike is sealed, we push them next to the sprinkler and let the components get sprayed directly to mimic a rainy situation. We were able to fully wet the rear radar and LED modules, and they were still functional after they were soaked. This suggests that we should be able to meet our designed use-case requirement: allowing commuters to ride their bike with our system in the rain.

7.10 Turn Signal Brightness

The turn signal brightness was tested both in the day and at night. We put the bike at a stationary location and picked a spot that was 100 ft away from the bike during the day and 500 ft away from the bike at night. Both distances were measured by using Google Maps satellite images.

In both cases, we were able to see the turn signal of the bike, meaning that it achieved the turn signal brightness requirements. In a subsequent experiment using a transistor that has a much higher current tolerance, we discovered that the brightness of the LEDs was significantly decreased due to higher resistance, so this change was reverted.

8 PROJECT MANAGEMENT

8.1 Schedule

The schedule and responsibilities are shown in Fig. 11.

8.2 Team Member Responsibilities

All team members worked on system integration and testing.

- Jack: Radar initialization and implementation, radar tuning
- Jason: User interface software, turn signal software (to handle button inputs, LED outputs, and use the magnetometer as compass to auto-cancel the signals)
- Johnny: CAD design for exterior, 3D printing, and building enclosures, parts installation

8.3 Bill of Materials and Budget

The BOM as of writing this report is in Table 6. All of the purchases were used.

8.4 Risk Management

Several risk factors were involved in this project:

Table 6: Bill of materials

Part Name	Quantity	Cost	Total
RPi 4	1	0	0
Hosyond 5 Inch IPS LCD	1	42.99	42.99
hdmi cable	0	0	0
RFbeam Microwave GmbH K-LD7	2	89.275	178.55
Anker Power Bank, 26,800 mAh	1	56.2	56.2
Feiteplus Handlebar Mount Switch	1	13	13
Hyuduo 1W 5V LED Chip Bulb	1	11.75	11.75
electrical wire	10	0	0
BS170	2	0	0
Twidec Momentary Push Button Switch	1	9.99	9.99
Adafruit MMC5603	1	11.19	11.19
Freenove Breakout Board	1	11.95	11.95
Techspark acrylics	2	8	16
McMaster-Carr printing filment	2	24.95	49.9
McMaster-Carr sealant	1	14	14
McMaster-Carr 4 in 1 cable	1	14.61	14.61
McMaster-Carr heat shrink	1	4.62	4.62
McMaster-Carr seal rubber	1	12	12
			$446.75

8.4.1 Design

- Waterproof Case: One concern was making sure the material and the setup of the case would not compromise the performance of our system. We managed this risk by reviewing the documentation of the radar which guided us on selecting radome materials and the available materials. We also discussed this topic with the instructors to determine the optimal solutions.

- Auto-cancellation turn signals: We were not sure how well the magnetometer would perform to detect heading changes of the bike, which is the information supplier for the auto-cancellation turn signals. We managed the risk by allowing us to have backup plans like using a potentiometer or flex sensors.

- Integration: The biggest risk during integration was something would not work as intended, as we had little time to correct it. We were facing issues with our turn signals, which we realized were due to the misuse of the transistors. We managed this risk by working in parallel and identifying the key components that needed to work first and prioritizing their functionality.

- Radar: We wanted to make sure that we could get the radar to work, but it was a big task from setting it up to processing data. We managed this by breaking down the tasks into smaller items. For each item, we implemented them and did unit tests before moving on to the next. This gave a good "phase check" of the system.

8.4.2 Logistics

The logistics risks were that we would run over time for some tasks on the schedule. We managed this risk by having around a week of slack time. It turned out that the slack time was critical to the project's success.

8.4.3 Resources

We wanted to make sure that our project would fall within the allocated budget. Any greedy purchases would risk us running over the budget. We managed this by being conservative during the first round of the purchases. We researched available resources that we could use instead of buying new ones. For each individual component, we researched the pros and cons of different models and selected the best that suited our needs. We also prioritized the key components of the project first like the Pi and the radar.

9 ETHICAL ISSUES

Bike commuters would be the ideal users for our system because they will directly benefit from the system as the targeted group why this system is built. The bike commuter and the public who share the road with them are most vulnerable. If the system fails, it will directly impact their safety. If the system provides false information, then they also run a higher risk of getting into an accident.

Our BikeBuddy will have a great impact on the global scale. The system will be able to be installed on different kinds of bicycles, so each individual around the world could benefit from such a system. Bike safety is definitely not only a local concern in Pittsburgh but an issue faced by all bike commuters around the globe. We try to make the

system generic so that it will be able to perform things like blind-spot detection and illuminate turn signals in diverse road and weather conditions. It will be designed to be able to use and interpret the information easily. This system will not only be beneficial to the cyclist community around the world but to all the general public who shares the road. It will likely reduce the number of crashes between the car and the bike, creating a safer road environment. It will not just be a system that will contribute to those in academics, but a system that people can actually use, benefiting their day-to-day lives on a bike, no matter where they are.

Integrating blind spot monitoring and turn signaling features represents not only a technological advancement but also an innovation intertwined with cultural factors. In cultures where cycling is popular, these innovations can significantly enhance safety, reflecting a commitment to promoting responsible behavior on the roads and emphasizing cyclists' well-being.

In addition, clear and consistent signaling enhances adherence to traffic regulations, contributing to a smoother flow of traffic and improved road safety. When riders consistently utilize turn signals, it becomes easier for law enforcement to monitor the traffic.

If we are able to encourage more people to bicycle instead of taking their car by giving users more confidence in their safety on the road, we can improve the environment by taking more vehicles off the road. This will hopefully improve air quality, reduce the use of fossil fuels, and reduce noise pollution.

However, our product itself may have a slight negative environmental impact due to the materials involved in building the system. For example, our battery (Anker 337) uses lithium, and the extraction of lithium can have negative impacts on the environment. While our product does not use large amounts of lithium compared to things like electric vehicles or other resources such as rare earth metals that are perhaps needed for things like the Raspberry Pi, we nonetheless do contribute to these issues.

Public health is very relevant to our project. We are searching for sustainable ways to travel and biking is one of them. The goal of our project would be to let more people feel safe about cycling and bring more people into this domain. We would like to use our technology to mitigate the risk of bike accidents and reduce the number of hospital visits due to accidents. One issue would be distraction-related accidents. Although the system is aimed at safety, it might distract the cyclists while they are reading the information. There are trade-offs between providing enough information to the rider and not too much information. To mitigate this, we are making the user interface simple and clear, so that people do not need to read small text from the display while riding.

This has an interconnection with public health considerations. The safety of cycling has always been a major topic, as traffic accidents due to bicycles are not reducing every year. The project will enhance the situational awareness of cyclists, providing them with more tools that they can use to protect themselves while sharing the road with larger vehicles.

Enhancing public welfare is every community's goal. People should be able to access products equally, especially those safety-critical equipment. Our project will try to enhance such factors by allowing more people to have access to a bike safety tool. This will address the issues that people feel biking is too dangerous so they will stay away from it. When more people start biking, the community becomes greener, with less noise generated by cars and other motor vehicles. This will enhance the happiness of the community. One trade-off is between the price of our system and its functionality. We do not want the system to have lower quality just to make it cheap. Instead, we are picking reasonable sensors and solutions that will fulfill our requirements, while being relatively cheap.

10 RELATED WORK

Within ECE at CMU, there were several previous teams that did something similar for their capstone.

A group in 2019 [6] has developed a similar bike safety suit that included a safety vest, blind spot detection using LiDAR, and a user interface app to control settings.

Another group in 2019 [28] seems to have developed a system that seems to have used ultra-wideband technology to localize bicycles and vehicles relative to each other [16]. Unlike other systems listed here, it seems that both the vehicle and bicycle will receive alerts.

Finally, another group in 2021 [2] developed a microwave radar-based bicycle safety and awareness tool.

There also exist several product families out in the real world that do something similar to our project.

For example, Garmin's Varia series includes products such as the Varia RTL515 which combines a tail light and a radar to warn of vehicles approaching from behind, or the Varia RCT715 which combines a taillight, radar, and camera which saves footage in the case of an incident.

Other bicycle radars according to [20] include the Bryton Gardia R300L which is a taillight (with brakelight functionality) and radar, along with the Magicshine SEEMEE 508 which also combines a taillight (with brakelight functionality) and radar. All these radars can see up to 140 m away (190 m for the Gardia radar).

For the display and centralized hub, Garmin's EDGE series of bicycle computers seems to have some of the same functionality. For example, they can have navigation data, and when integrated with a radar, can display an approaching vehicle on the screen.

Lastly, Amazon seems to carry bicycle turn signals, similar to what we developed here. For example, this WSD-CAM tail light includes features such as brake lights, turn signals, anti-theft alarms, and horns.

11 SUMMARY

BikeBuddy provides a new solution to cycle safety by allowing cyclists to gain more situational awareness with tools like blind spot and range detection. The added turn signals will also allow the cyclist to communicate with other road users better, enhancing safety. The use of radar and waterproof enclosures allows the system to function even during adverse weather conditions, which allows the commuter to bike with such a system in the rain.

There were several lessons that we learned over the course of this project. On the logistical side, we learned that it's important to run as many tasks in parallel as possible to save time. We also learned that even one week of slack time wasn't enough. More slack time can be helpful.

On the technical side, one thing we learned was that magnetometers are not trivial to use as compasses. They need to be calibrated, and nearby magnetic objects such as the tool drawers in TechSpark can affect the accuracy of it. If we were to do this again, we would have used an IMU instead.

It is also a lot of complexity to use radars for object detection, even if the radar already does data processing for you. It was a persistent challenge to get the radar to not be so noisy.

Lastly, make sure to read the datasheets, especially with respect to the limits, and make sure you know what you're subjecting the device to. We blew up a bunch of MOSFETs because we didn't look at the current limits and pushed too much current through them.

More positively, JavaScript proved to be quite versatile. We were able to use it for desktop programming and some embedded system tasks instead of just web development which is really nice. The developer ecosystem is quite nice, especially the hot-reloading which allows code changes to automatically propagate to the running application.

Building a debug system for the user interface which allowed us to simulate sensor inputs also proved to be quite useful. We would strongly recommend building something like this.

Glossary of Acronyms

- BSM - Blind Spot Monitoring
- RPi – Raspberry Pi

References

[1] Outside Online. *Strava's End-of-Year Insights*. https://velo.outsideonline.com/road/road-racing/strava-end-year-insights-live-fastest-state/.

[2] Albany Bloor, Emily Clayton, and Jason Xu. *ECE 18-500 Project: Bikewardsview*. https://course.ece.cmu.edu/ ece500/projects/f21-teamb3/. 2021.

[3] SparkFun Electronics. *SparkFun-Boards*. `https://library.io/libraries/509-SparkFun-Boards`. 2023.

[4] Bret. *Raspberry Pi 5 Review. Harder, Better, Faster, Stronger*. Feb. 2024. URL: `https://bret.dk/raspberry-pi-5-review/#Raspberry-Pi-5-Benchmarks`.

[5] SparkFun Electronics. *SparkFun-Connector*. `https://library.io/libraries/513-SparkFun-Connectors`. 2023.

[6] Michael You, Sid Lathar, and Benjamin Huang. *ECE 18-500 Project: CycleSafe*. https://course.ece.cmu.edu/ ece500/projects/s19-teama1/. 2019.

[7] OpenJS Foundation and Electron Contributors. *What is Electron?* URL: `https://www.electronjs.org/docs/latest`.

[8] Roy Gothie. *Visibility and conspicuity while riding your bike*. Mar. 2019. URL: `https://www.penndot.pa.gov/PennDOTWay/pages/Article.aspx?post=206`.

[9] Muhammad Hasanujjaman, Mostafa Zaman Chowdhury, and Yeong Min Jang. "Sensor Fusion in Autonomous Vehicle with Traffic Surveillance Camera System: Detection, Localization, and AI Networking". In: (Mar. 2023). URL: `https://www.researchgate.net/publication/369468206_Sensor_Fusion_in_Autonomous_Vehicle_with_Traffic_Surveillance_Camera_System_Detection_Localization_and_AI_Networking`.

[10] SparkFun Electronics. *SparkFun-IC-Special-Function*. `https://library.io/libraries/527-SparkFun-IC-Special-Function`. 2023.

[11] Petra Industries. *Be the Expert: Waterproof, Water-Resistant and Understanding the Standards Rating Charts*. Mar. 2022. URL: `https://blog.petra.com/blog/understanding-water-standards-rating-charts/`.

[12] *K-LD7. data sheet*. Version Rev. B. RFBeam Microwave GmbH. Mar. 2023. URL: `https://rfbeam.ch/wp-content/uploads/dlm_uploads/2022/10/K-LD7_Datasheet.pdf`.

[13] ladyada. `https://github.com/adafruit/Adafruit_SensorLab/blob/master/notebooks/Mag_Gyro_Calibration.ipynb`. Jan. 2020.

[14] Simon Long. *Bullseye - the new version of Raspberry Pi OS*. Nov. 2021. URL: `https://www.raspberrypi.com/news/bookworm-the-new-version-of-raspberry-pi-os/`.

[15] Raspberry Pi Ltd. and Documentation Contributors. *Raspberry Pi hardware*. URL: `https://www.raspberrypi.com/documentation/computers/raspberry-pi.html`.

[16] *MDEK1001. Ultra-Wideband (UWB) Transceiver Development Kit*. URL: https://www.qorvo.com/products/p/MDEK1001.

[17] WI - Weather Forecast Office Milwaukee/Sullivan. *Using and Understanding Doppler Radar*. URL: https://www.weather.gov/mkx/using-radar.

[18] Lea Moussy. *LiDAR vs. RADAR*. May 2020. URL: https://www.yellowscan.com/knowledge/lidar-vs-radar/.

[19] MIT News. *How fast can humans react to car hazards?* 2019. URL: https://news.mit.edu/2019/how-fast-humans-react-car-hazards-0807.

[20] Paul Norman. *What are rearview radar bike lights and should you use one?* Nov. 2023. URL: https://www.bikeradar.com/advice/buyers-guides/bike-rearview-radars.

[21] National Oceanic and Atmospheric Administration. *How radar works*. Sept. 2023. URL: https://www.noaa.gov/jetstream/doppler/how-radar-works.

[22] Outside Online. *Lane Width*. URL: https://safety.fhwa.dot.gov/geometric/pubs/mitigationstrategies/chapter3/3_lanewidth.cfm.

[23] *Products*. URL: https://rfbeam.ch/products/.

[24] *React*. React team and external contributors. May 2024. URL: https://react.dev/.

[25] RFbeam Microwave GmbH. *RFbeam Microwave GmbH*. Python Version: 3.7.4. Enter the corresponding COM Port and make sure all modules are installed before executing. Rev 1.0 — 15.10.2019 — - Initial version — FN. Sept. 2019. URL: https://forums.raspberrypi.com/viewtopic.php?t=317286.

[26] SparkFun Electronics. *SparkFun-Switches*. https://library.io/libraries/535-SparkFun-Switches. 2023.

[27] Elliot Williams. *A Raspberry Pi 5 Is Better Than Two Pi 4s*. Sept. 2023. URL: https://hackaday.com/2023/09/28/a-raspberry-pi-5-is-better-than-two-pi-4s/.

[28] Zexi Yao and Kyle Kozlowski. *Team B0: Vehicle Cyclist Collision avoidance System. Carnegie Mellon ECE Capstone, Fall 2019: Zexi Yao, Kyle Kozlowski and NOBODY*. 2019. URL: http://course.ece.cmu.edu/~ece500/projects/f19-teamb0/.

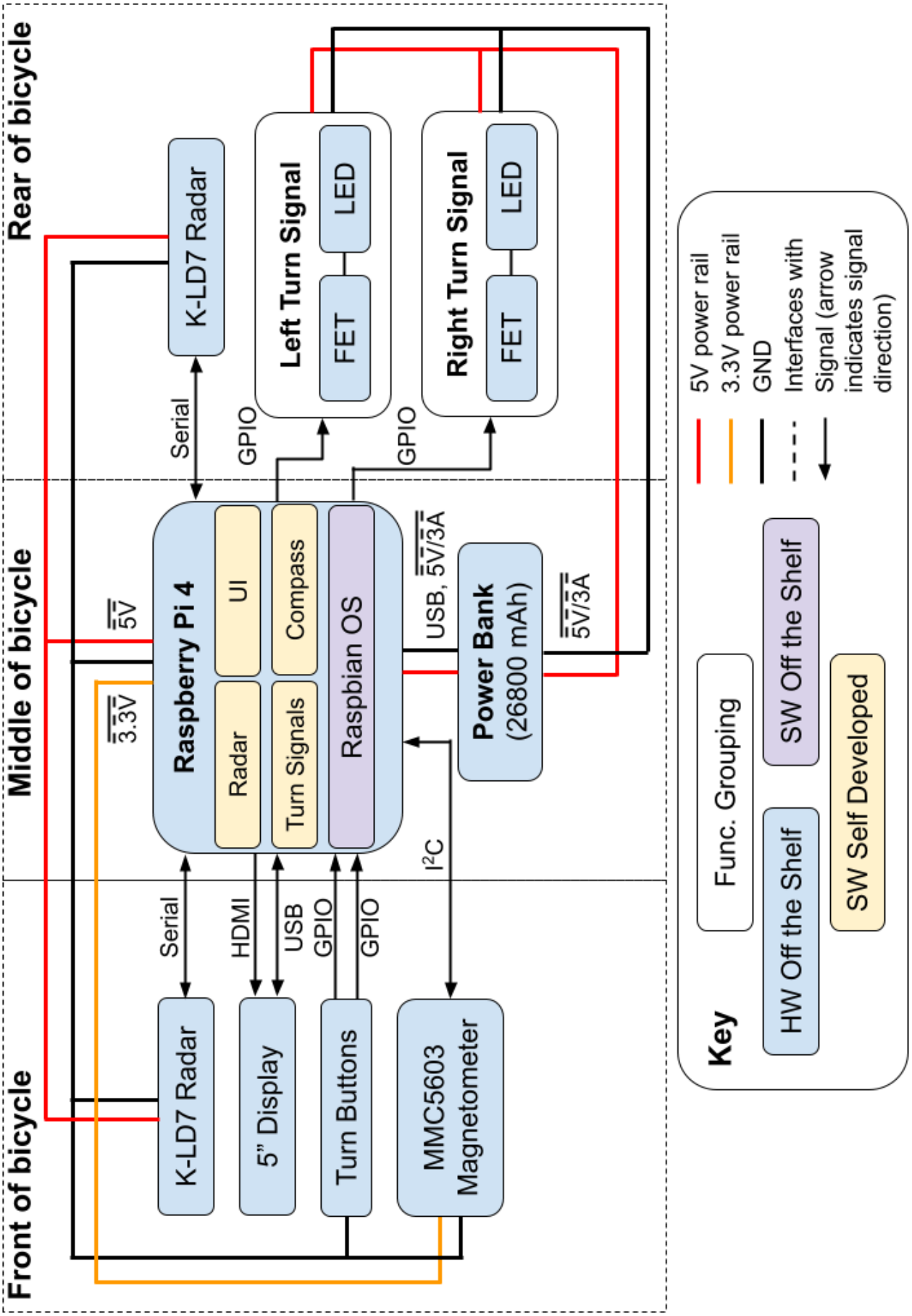

Figure 10: A full-page version of the same system block diagram as depicted earlier.

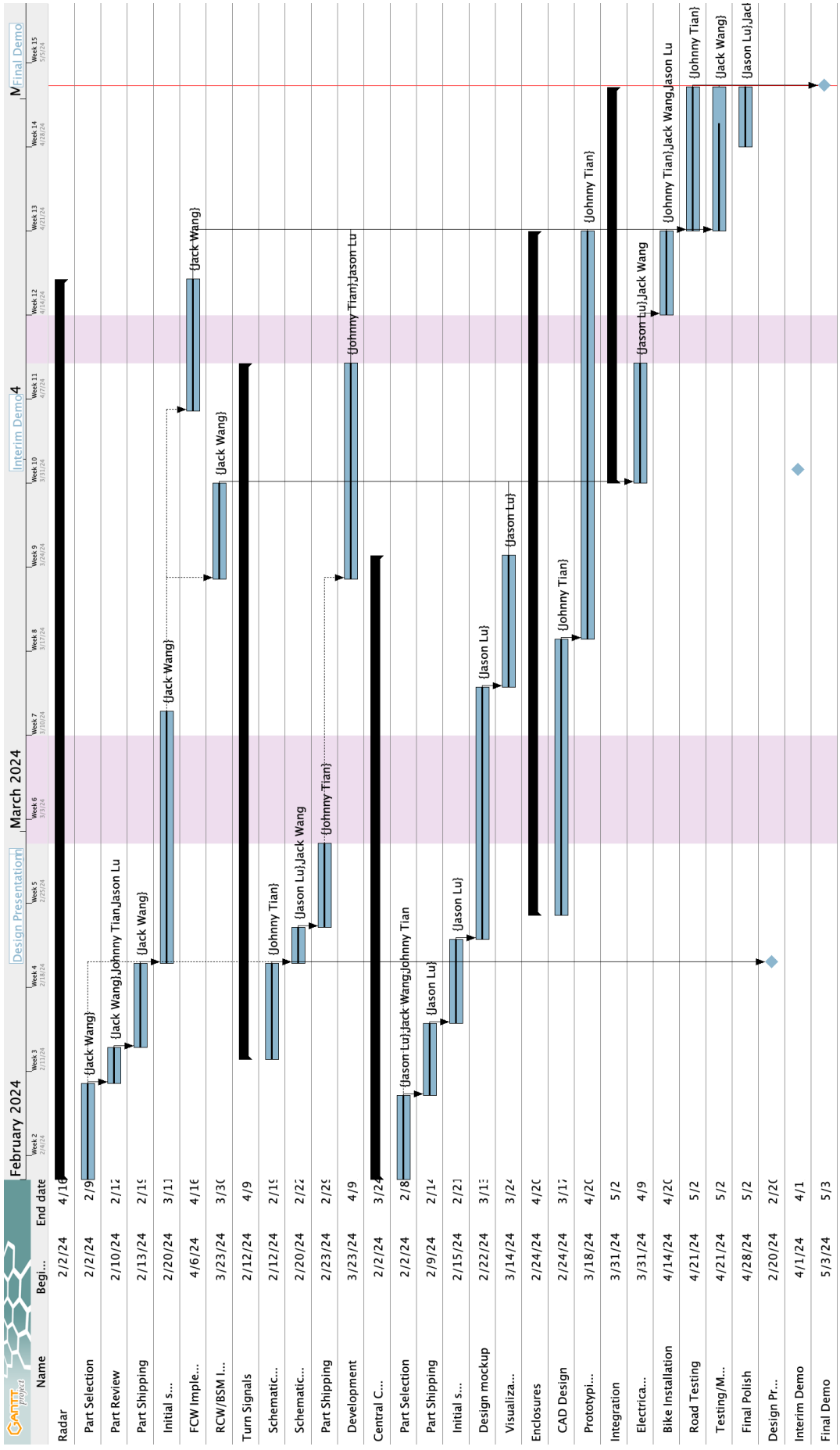

Figure 11: Gantt Chart