CycleGuard

EECS 473
Fall 2024

Brian Good, Noah Hicks, Grant Hincher, Nate Sochocki, Tyler Trent

Table of Contents

CycleGuard

EECS 473
Fall 2024

Introduction

Cyclists face many dangers on the road, with increasing incidents of accidents involving vehicles occuring every year. Alarmingly, the number of preventable deaths from bicycles have increased by 47% in the last ten years[1]. These collisions occur in various contexts, but notably 40% of collisions with bicycles and vehicles are rear-end collisions[2]. This type of collision can occur when a cyclist is operating at a much slower speed and a vehicle attempts to overtake them or plainly does not see them. Bicycles often lack the necessary safety equipment to alert the rider of potential collisions and alert drivers in trailing motor vehicles, such as brake lights and turn signals. Such basic safety equipment that is standard, and required, in motor vehicles is often not found in bikes.

Our project addresses this issue by providing a comprehensive bike safety system that enhances both cyclist awareness and visibility to drivers. The system includes a robust radar module capable of reporting traffic speed, directions, and position to visibly alert the cyclist of rear-approaching traffic and potential collisions, integrated lights for stopping, turn signal indicators (also serving as hazard lights when both signals are engaged), and proximity warning lights to alert drivers. This combination of safety features not only assists cyclists in reducing the likelihood of collisions but also gives them greater confidence on the road.

We were able to display our achievements at the CoE Design Expo by bringing in a bike and attaching the system to it to display our product, while also producing a trailer video which displays the product working in real scenarios on the road. Our product functionality was assessed a few days later in action by course staff. We achieved all the main goals we established in our proposal, namely detecting trailing vehicles and assigning danger levels, integrating turn signals, a proximity warning light, a piezo buzzer for an audible alert, and a brake light with automatic braking detection through an IMU. The only goal listed in our proposal which we did not achieve was our stretch goal to automatically turn off the turn signals after completing a turn.

Our Design

Project Goals & Concept

The overall goal of our project was to improve rider safety and visibility to address the upward trend in fatal accidents caused by cyclist-vehicle collisions. We achieved this in several ways, notably by improving rider awareness of approaching vehicles, and improving visibility of the bike by implementing a proximity warning light. Additionally, we added an automatic brake light and dedicated turn signals so the cyclist

[1] https://injuryfacts.nsc.org/home-and-community/safety-topics/bicycle-deaths/
[2] https://bikeleague.org/wp-content/uploads/2023/03/EBC_report_final.pdf

wouldn't have to hold out their arm, compromising handlebar control in the event of an emergency. Figure 1 below illustrates the physical layout of these components as separate rear and front modules.

Figure 1: Assembled Modules Mounted On The Bike

We believe that this product is very feasible in today's market, as it's unique, and fits into a lucrative market. Products like Garmin's Varia[3] have some of the same functionality, but lack features such as a brake light, turn signals, and a mechanism to alert the cyclist for $200. Another upside of our product is that it's an all-in-one solution, while with Garmin's product you need to purchase their Edge bike computer[4] to be alerted of vehicles for an additional $200. Our goal was to provide a middle ground product that provides some of the advanced features offered by products like the Edge, while at a lower price-point than purchasing both Garmin products.

Design Constraints

The biggest design constraint for us was that the radar needs to accurately detect the speed and distance of approaching vehicles, as well as the direction. The radar module we used (OPS243-C) allows range detection up to 196 ft and speed reporting of up to 328 mph. In our case, we consistently detected up to 130+ ft and had no issues detecting vehicle speed. This allows us to assign a danger level to detected vehicles, ranging from 1-4 (with 4 being the most dangerous), which is used to alert the cyclist via a series of LEDs on the front module and on the rear module the proximity LED blinks at a frequency corresponding to the danger level—alerting trailing vehicles of a dangerously close following distance.

[3] https://www.garmin.com/en-US/p/698001
[4] https://www.garmin.com/en-US/p/698436/pn/010-02385-00?gQT=2

A constraint of specifically the rear module is the brightness of the LEDs, which must be able to be seen clearly in the daytime and also need to be small form factor to fit on the PCB. We utilized a system of small surface area, high current LEDs to accomplish this. The LEDs on the rear are 0.136" x 0.136" and are rated at 350mA, with a lumen rating of 87 lm. Due to the small form factor and thus surface area, these LEDs are very bright and serve our use case well.

The two modules in our system communicate over Bluetooth Low Energy (BLE) in a server-client setup (back module is the server, front module is the client). Given that the two modules must go through a pairing sequence each time they power on, it is important that the pairing takes a reasonable amount of time for the user. We defined a constraint to have a maximum of 10 seconds between power on and pairing.

Our system was also constrained by the physical challenges inherent in a cycling environment. We needed a mounting solution that was sleek enough to be appealing to a user while also robust enough to withstand all of the violent motion that can occur during the course of a bike ride. To satisfy these, we designed our enclosures to be as compact as possible, providing a tight fit for the respective PCB's sitting atop the batteries as well as the radar in the rear. Additionally, we used rubber decoupling screws to mount the radar in such a manner that it was free to sway with the motion, mitigating the effect of mechanical vibrations. This cut down on the rigidity and damped some of the harmful motion. The final dimensions were approximately 4.25x4x1.6" for the rear and 3x2.75x0.75" in the front. These were mostly limited by the PCB and radar size. If we were to design our own radar integrated PCB for the rear and compact the front, we could likely reduce these dimensions further.

Another constraint of the system was the battery life, as the product needs to be able to last for several rides without having to be recharged. Since our system is divided into two modules (front handlebar and rear seat post), we used LiPo batteries to supply power to each. The front component uses a 3.7V, 2500mAh LiPo, while the rear uses a 7.6V, 3500mAh LiHV (high voltage LiPo). To estimate the battery life of each component, we consider an idle and average use case. Our findings are highlighted in table 1 below.

Component	**Idle (Hr)**	**Average (Hr)**
Front	30.23	21.13
Rear	10.6	6.74

Table 1: Battery Life Estimates

For both modules, the idle case is when no object is being detected, thus no front lights are illuminated on the front, which draws 82.7 mA, and on the rear the proximity light turns on for 100 ms every 2 seconds, drawing 330mA. The average case for the front consists of a turn signal on for 5 minutes every hour, drawing 100mA, and an average danger level of 2 (two LEDs illuminated) drawing 120mA. For the rear we consider the case where a car is detected 50 minutes out of every hour, drawing 555mA, the brake light is on for 5 minutes, also drawing 555mA, a turn signal is on for 4 minutes, drawing 545mA, and the hazards are on for 1 minute with draw of 747mA.

Lastly, we needed to finish the project before the deadline, as well as below our allotted budget of $1000. We were able to achieve both of these, coming in just $1.50 within our budget, and finishing a bit early, a few days before the design expo.

Mechanical Implementation

The casing was primarily designed with respect to packaging our components together in as little space as possible and keeping our radar safe from sudden, violent movements. This was previously discussed in the design constraints section. Beyond that, we wanted the front to have a slick and intuitive UI, and an easy mounting system. It sits atop the board and provides access to all of our buttons and has a clean display with our LED's. As well as this, an external interface with our USB jack was cut into the case so it can be charged and/or flashed as an assembled module. The PCB was designed before the front casing, so it mirrors those design choices and tries to make it more intuitive with clear labels. We chose to use zip ties and fit the bottom of the module to handle bars as a common and simple installation method.

The rear was more so concerned with durability than the front. This part of the bike often has no suspension, and is subject to a lot of force. We acquired an aftermarket tail light seat post mount meant for another product that served as the basis for our mounting design. From this, we matched up the mounting pattern and sank some heat set nuts into the print so we could have metal threads all the way through, ensuring our system was securely attached to the bike. We used these same nuts to anchor the radar, which was attached via the rubber decoupling screws to damp the system. The CAD models for the rear and front are illustrated below in figure 2 and 3, respectively.

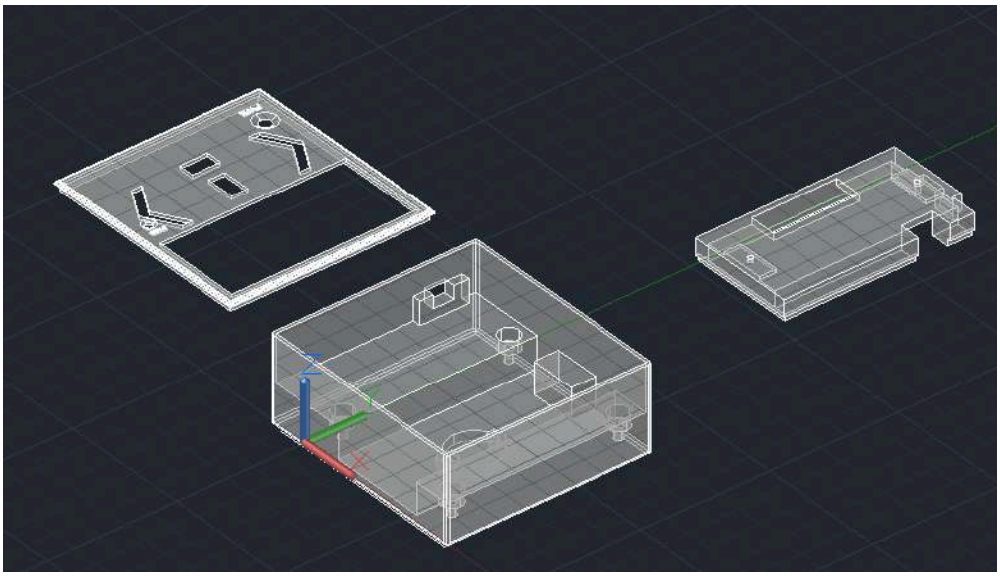

Figure 2: Rear Module Casing

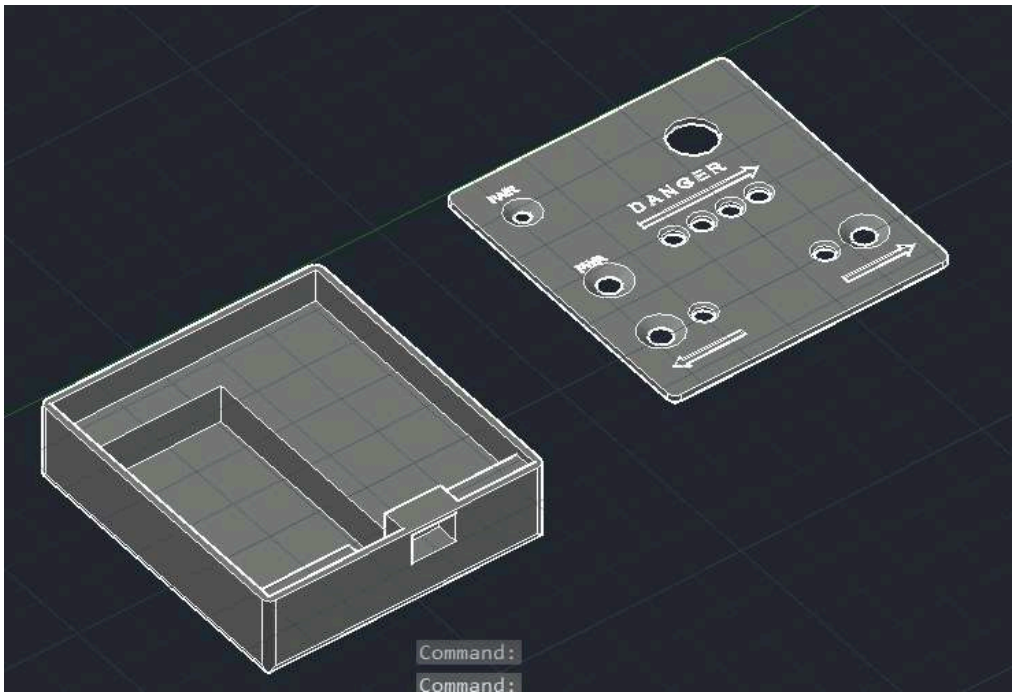

Figure 3: Front Module Casing

The PCB sits atop the radar in another print layer, attached via the same decouplers to keep the radar totally separated from the rest of the system. The PCB was shifted below the patch antenna on the radar to ensure as little interference as possible. Throughout the successive print iterations, we found the print

material itself to cause severe issues with our radar range, so much of this was cut away to allow a clear line of sight in the final design. The appropriate cutouts were also made so we had access to all of our buttons and lights, and a charging port for our balance charger was added on the bottom. When printing the final iteration, we also used PETG filament instead of the standard PLA to further increase durability and flex. This all culminated in an enclosure that we logged extensive testing on without any mechanical issues or worries, and we are quite confident it could hold up to normal cycling conditions over extended periods of time.

System Architecture & Block Diagram

Our system is divided into two modules: one to detect vehicles and braking, while communicating to the vehicles on the road, and another to display that information to the rider.

Most of the functionality in our product comes from the rear, shown in figure 4 below. The sensors on the module consist of a doppler & FMCW radar, and an IMU. We communicate to the radar using UART, and the IMU uses I2C. For the LED outputs, we simply use GPIO as an interface. Since the current pull of these LEDs are over 200mA, we use the 7.6V battery to power them, and toggle a FET with the GPIO to allow current to flow through them.

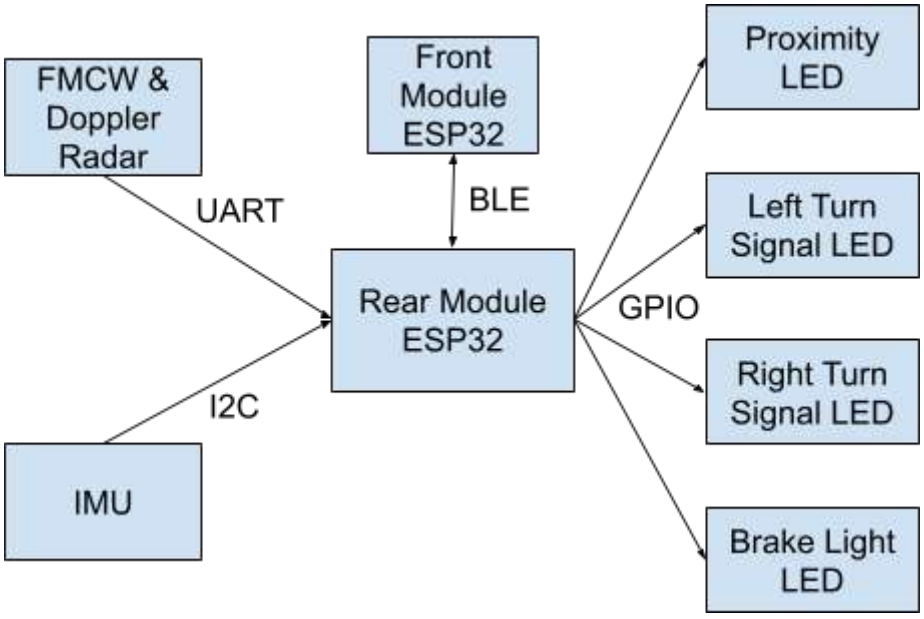

Figure 4: Rear Module Block Diagram

The front module gets attached to the handlebars, and it's used to communicate information to the rider, and toggle the turn signals. Everything on the front is interfaced simply using GPIO pins. Since the LEDs on this board only pull 20mA, we were able to bypass the FET that we used on the back, and just

power each LED from its own GPIO. The block diagram for the front module can be observed in figure 5 below.

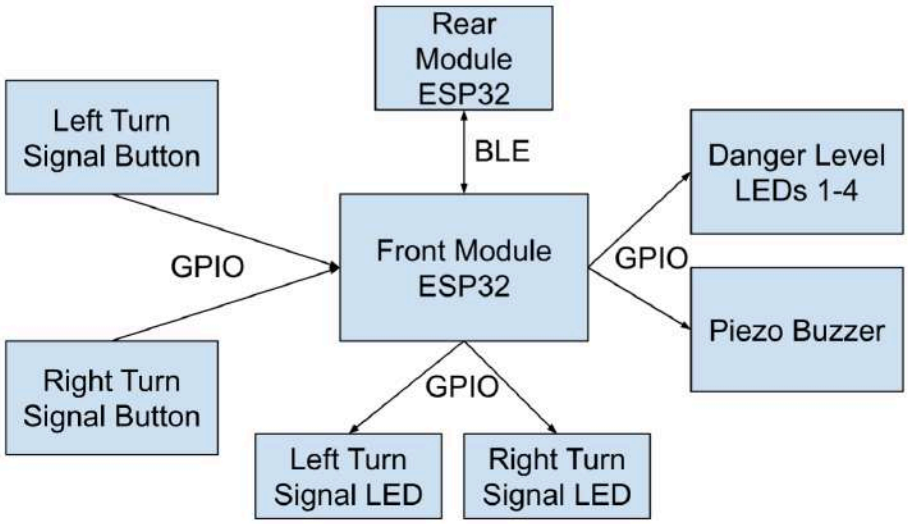

Figure 5: Front Module Block Diagram

Technical Challenges

Radar Challenges

The radiating pattern of the antenna used in our radar module works well enough for our project to be demonstrably functional, but it was not perfect for our application. In particular, noise from the bike wheel causes a lot of issues when the seat is lowered. We receive a fair amount of false positive danger readings when the seat is lowered, and none when it is raised. This is a consequence of our radar having a relatively broad beam in the plane of the wheel, specifically 24 degrees from center either way as stated in the product data sheet. Realistically, we would want a lot more directivity since vehicles will be in the same plane as the center of the radar in most cases. Widening this beam just allows for unwanted noise like we had.

Were we continuing this project, it would be worthwhile from a cost and functionality perspective to design and implement our own antenna and radar module, adjusting the array parameters and narrowing down this beam. The physical placement of the radar could also be altered, but this runs the risk of throwing off our line of sight and negatively affecting our detection range.

PCB Challenges

With our first iteration of the rear PCB we experienced difficulties with the 7.4V to 3.3V voltage regulator. The symbol we used, which was sourced from Digikey, incorrectly labeled the Vout pin as GND,

so the initial rear PCB design shorted 3.3V to our entire GND plane. We attempted to solve this by desoldering this pin, since there is a separate Vout pin on the LDO, but this did not fix the issue. We believe that shorting the GND plane must have shorted one or more parts on our board, rendering it useless. We reordered the PCB after resolving the incorrect footprint and the voltage regulator functioned as expected after that.

On the second iteration of the PCB, we discovered that we improperly wired the SCL and SDA traces from the IMU to the ESP32. The datasheet for the ESP32 specified to connect SCL/SDA lines to pins 21 and 22 in reference to GPIO pins 21 and 22, however we misinterpreted this and wired them to the actual pins numbered 21 and 22—which are supposed to be left disconnected on the ESP32. We had previously added some through holes to connect wires to some of the GPIO pins on the ESP32 externally in case we ran into any issues, so we were able to use these and solder physical wires from the pull-up resistors for the SCL/SDA lines directly to GPIO pins 21 and 22. After this was done the board functioned correctly with no further issues.

BLE Challenges

We took advantage of existing Arduino libraries for our BLE functionality including the pairing sequence. Initially, the example we based our code off of used 30 seconds for its search time on the client side (server initializes and starts advertising immediately). An issue we found was that if the two modules failed to pair within the first 3-5 seconds of the search time, then it was very unlikely that they would pair by the end of the 30 seconds.

Our solution was to reduce the search time to only 1 second, and restart the device (and thus the pairing process) on a failure to pair. Now, we see that it almost always takes only 2-3 seconds to pair, and never takes more than 10. Given the close proximity of the devices, the 1 second search time seems to be adequate, and even if we have to search multiple times, it only comes at a 1 second penalty rather than the original 30 seconds.

Furthermore, any disconnect between the two modules when previously connected results in the same device restart. This means that a disconnect results in a 2-3 second window of unresponsiveness of the front module before re-pairing is complete. It is also worthwhile to note that we never encountered any disconnects during any of our field testing, but wanted to make sure our system could handle this failure if it were to occur. We are quite satisfied with the results of our BLE pairing process and failure recovery.

Milestones, schedule, and budget

Milestone 1

For the first milestone, we were intending to demonstrate that we can obtain sufficient readings from our radar in order to use them for calculating danger level, as well as a breadboarded synchronous light system similar to what we implemented on the front module to communicate danger to the cyclist. Both of these objectives were completed and demonstrated for our demo. At the time of the first milestone we also intended to have the final PCB design ready to be reviewed and ordered. We did not have this completed at the time of the first milestone due to delay in some design decisions regarding LED and button placements as well as how we wanted to flash code to the boards. We promptly got to work on this and were able to get the PCB reviewed the following week with some changes needing to be made.

Milestone 2

For the second milestone we intended to have our warning LED system fully functional on a breadboard as well as obtaining adequate readings from our IMU to illuminate our brake light. We were able to demonstrate our LED system fully working, however our IMU readings were not quite where we wanted them to be at the time of demo. This is due to a gravity component which we had not fully determined the proper way to mitigate for the deceleration readings we were looking for. We also intended to have our PCB assembled in order to test it at this time, however due to the delays from the first milestone, the boards were in fabrication at this time soon to be shipped to us.

Budget

We ran over our original budget by approximately $100. This is mainly due to the fact that we needed to reorder our PCBs due to the faulty symbol on the initial rear PCB design. We spent a total of $998.50 ($887.25 not including shipping costs) and our original cost prediction was $886. The second order of the PCBs cost us $123.18 which is more than the difference between our original cost and our final running total so other than this unexpected reorder we stayed within our budget quite well.

Lessons learned

One part of our project that went extremely well was the development and design of the front module. The front PCBs were perfect on the first order, and not much debugging was needed on the software side. While the code was relatively simple, we were able to be mostly done with the front module by the middle of November.

Another aspect that went well was the 3D printed casing. Around the beginning of November we had a high quality prototype to begin testing in. We made some small changes to the design, but both the front and back largely stayed the same throughout the end of the project. With that being said, the material used for the casing posed a problem with the radar antenna, producing inaccurate results when going through both the PLA and PETG filament. In retrospect, we should have tested with our printed materials earlier to address issues of permeability, giving us more time to find an adequate solution. Even after consulting with OmniPreSense and receiving their recommendations, we still struggled to get accurate results with any type of material over the antennas. Despite following their guidelines, the radar still performed much better without anything over it.

Our IMU of choice also was problematic, as it was very inaccurate at times. We would sometimes get outliers more often than useful results, and it was very difficult to debug. We wanted to account for gravity when going up or down a hill, but even on flat ground we received readings such as 12m/s^2 or 6m/s^2, which would cause an offset to be applied when there should not be. At first, we attempted to trivially change our thresholds to account for this, but in retrospect we should have explored alternative, more sophisticated methods for data processing earlier.

For example, our solution ended up being using a median filter and requiring multiple braking ‘events’ in a row. We checked the median of the buffer every 70ms, and required 5 values in a row to be above the threshold. If we hadn’t spent so much time trying to account for a flawed methodology, then we may have come to this conclusion earlier.

Overall, we feel we have gained significant technical knowledge in the field of radar, real time operating systems (RTOS), BLE, IMUs, PCB design, and testing strategies.

Team Contributions

Team member	Contribution	Effort
Noah Hicks:	Contributed to the rear PCB routing, the IMU interface, proximity light interface, and radar debugging. Wrote the tasks readIMU, flashBrake, and the proximity tasks. Also assisted with testing of both modules and brainstorming solutions to various issues.	20%
Grant Hincher:	Designed all physical components for printing, came up with mechanical solutions for secure and functional mounting. Assisted in testing and software debugging for various subsystems involved (brakes, radar detection).	20%

Brian Good:	Contributed BLE connectivity and communication between front and rear modules. Assisted with initial radar testing. Also helped with field testing and debugging efforts for all parts of the system at the end of the project. Edited video for demo day.	20%
Nate Sochocki:	Came up with the project idea, configured and designed software for radar and relating tasks, designed initial project code, assisted in PCB routing and debugging. Helped orchestrate testing of the system.	20%
Tyler Trent:	Designed the majority of both front and rear PCB schematics and routed the front PCB. Also involved in much of PCB testing and troubleshooting for redesign. Helped with field testing after PCB design.	20%

Cost of Manufacture

In 2023, there were more than 42 million road biking participants in the United States[5]. Assuming that only the top 5% of cyclists ride often enough to justify a bike safety system, then there are 2.1 million possible customers in our market. We think it is reasonable to start with 5000 units for our first round of production targeting this audience.

For 5000 units, JLCPCB quoted 40 cents per rear PCB, and 52.5 cents per front PCB for a total of 92.5 cents per system. The ESP32's come in at $2.50 per unit. Batteries cost $10 and $15 for the front and rear modules, respectively. Our casing would be mass produced via injection molding at an estimated cost per unit of 25 cents. We attempted to get a quote from OmniPreSense for 5000 radar units but never received a response. We estimate that we could receive a 20% discount off of the sticker price due to our order size, for a total of $191.20 per module. In total, we estimate that our material cost per unit would be $219.88. This feels reasonable for a device that we hope to sell for $300, leaving us with about 30% profit margin.

[5] https://www.statista.com/statistics/763746/road-paved-surface-bicycling-participants-us/

Parts and Budget Items

PART	COST
OPS243-C Radar	$239.00
ESP32 and IMU Dev Kits	$41.11
Batteries and respective chargers	$50.43
Bike post mount	$14.51
First PCB order (front and rear)	$140.07
Second PCB order (rear reorder)	$97.31
Module mounting hardware	$55.05
Bicycle	$127.49
Miscellaneous PCB passives	$122.28
TOTAL	**$887.25**

Table 2: Final Budget

References

Software Interfaces

BLE Interface:

```
enum class BLE_Message : uint8_t {
    DANGER_LEVEL_0,
    DANGER_LEVEL_1,
    DANGER_LEVEL_2,
    DANGER_LEVEL_3,
    DANGER_LEVEL_4,
    LEFT_TURN_SIGNAL_ON,
    LEFT_TURN_SIGNAL_OFF,
    RIGHT_TURN_SIGNAL_ON,
    RIGHT_TURN_SIGNAL_OFF,
```

```
  BRAKE_SIGNAL
};
class MyServerCallbacks: public BLEServerCallbacks {
  // callbacks triggered when BLE connection is established/ended
  void onConnect(BLEServer* pServer);
  void onDisconnect(BLEServer* pServer);
};
class MyCallbacks: public BLECharacteristicCallbacks {
  // callback triggered when BLE message is received
  void onWrite(BLECharacteristic *pCharacteristic);
};
void init_BLE_server(void);
void init_BLE_client(void);
int serverSendMessage(BLE_Message message);
int clientSendMessage(BLE_Message message);
```

Radar Interface:

```
#define RX_PIN 16
#define TX_PIN 17

//Initializes OPS243 radar to report speed (in fps), distance (ft) and only detect inbound objects
void initRadar();

//does a serial read to get data currently on the serial port
string readRawRadar();

//gets the speed (fps) reading of the current object from the radar
float getRadarSpeed();

//gets the distance (ft) reading of the current object detected from the radar
float getRadarDistance();
```

IMU Interface:

```
// sets up the I2C bus for transactions with the IMU
boolean Adafruit_LSM6DS::begin_I2C(uint8_t i2c_address, TwoWire *wire, int32_t sensor_id);

// returns an IMU sensor object for reading the axes acceleration values
Adafruit_Sensor *Adafruit_LSM6DS::getAccelerometerSensor(void);

// function which fills an imu accel object with readings from all 3 axes
```

*bool Adafruit_LSM6DS_Accelerometer::getEvent(sensors_event_t *event);*

LED Interface:

```
// initializes all LEDs on the rear module as outputs
void initRearLights();

// functions which writes a 1 to the rear proximity light
void setRearLight();

// function which writes a 0 to the rear proximity light
void resetRearLight();


// interrupt handlers which perform a deferred interrupt to send a BLE signal to the rear module
void IRAM_ATTR leftTurn();
void IRAM_ATTR rightTurn();


// maps the front danger level LEDs to the according GPIO pin
#define LED0_PIN 14
#define LED0_PIN 15
#define LED0_PIN 16
#define LED0_PIN 17

uint8_t lightPins[4] = {LED0_PIN, LED1_PIN, LED2_PIN, LED3_PIN};

// initializes all front LEDs
void initFrontLights();
void setLED(uint8_t LED);
void resetLED(uint8_t LED);
// takes in a danger level value and sets the corresponding LEDs
void setFrontStatus(uint8_t danger);
```

Piezo Buzzer Interface:

```
#define piezoPin 18
// initializes the piezo buzzer
void initPiezo();

// plays the danger level 4 alert sound on the piezo buzzer
```

void soundAlert();

// plays the pairing indicator tone on the piezo buzzer
void soundPair();

Final PCB Images

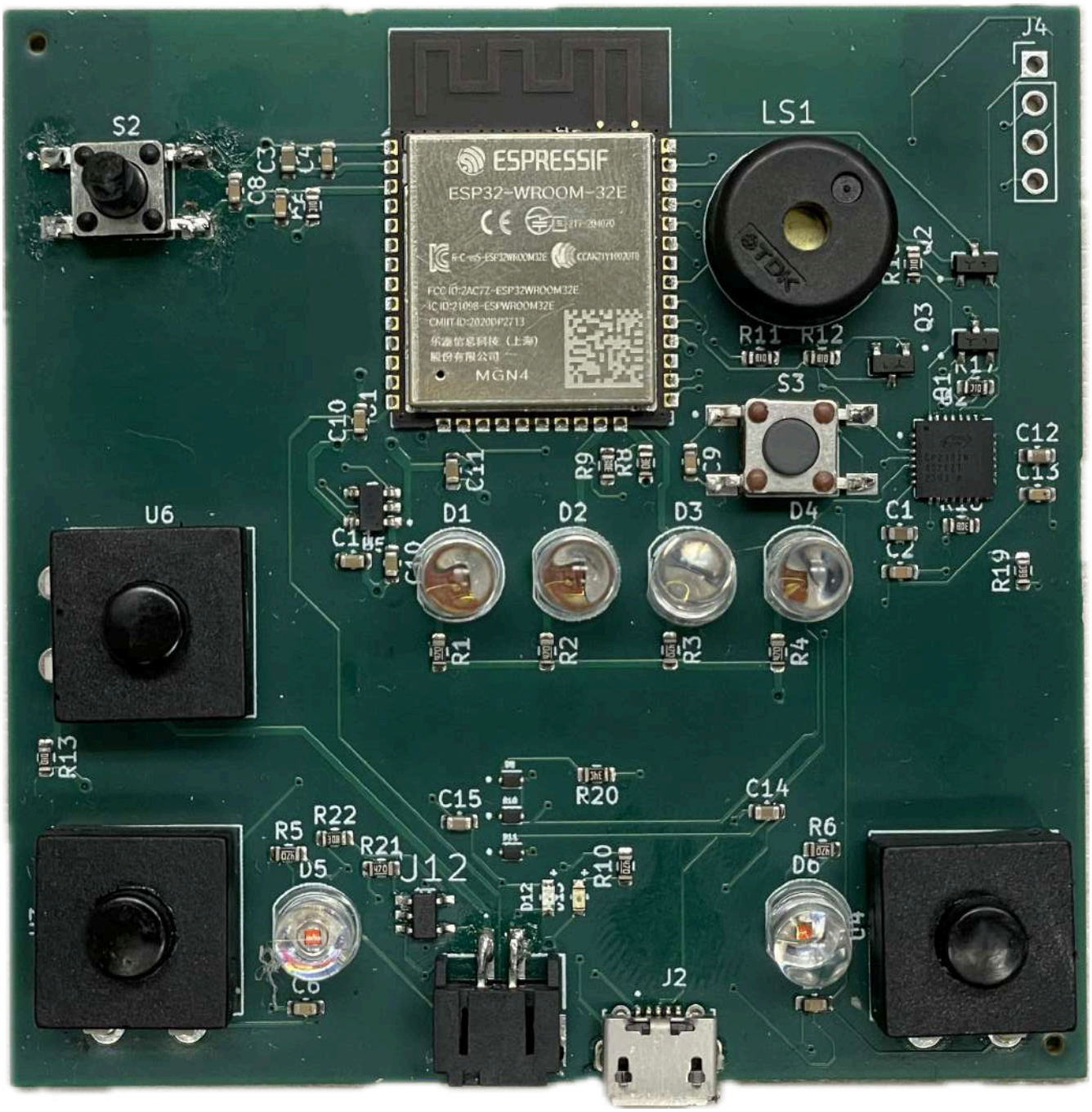

Figure 6: Front PCB

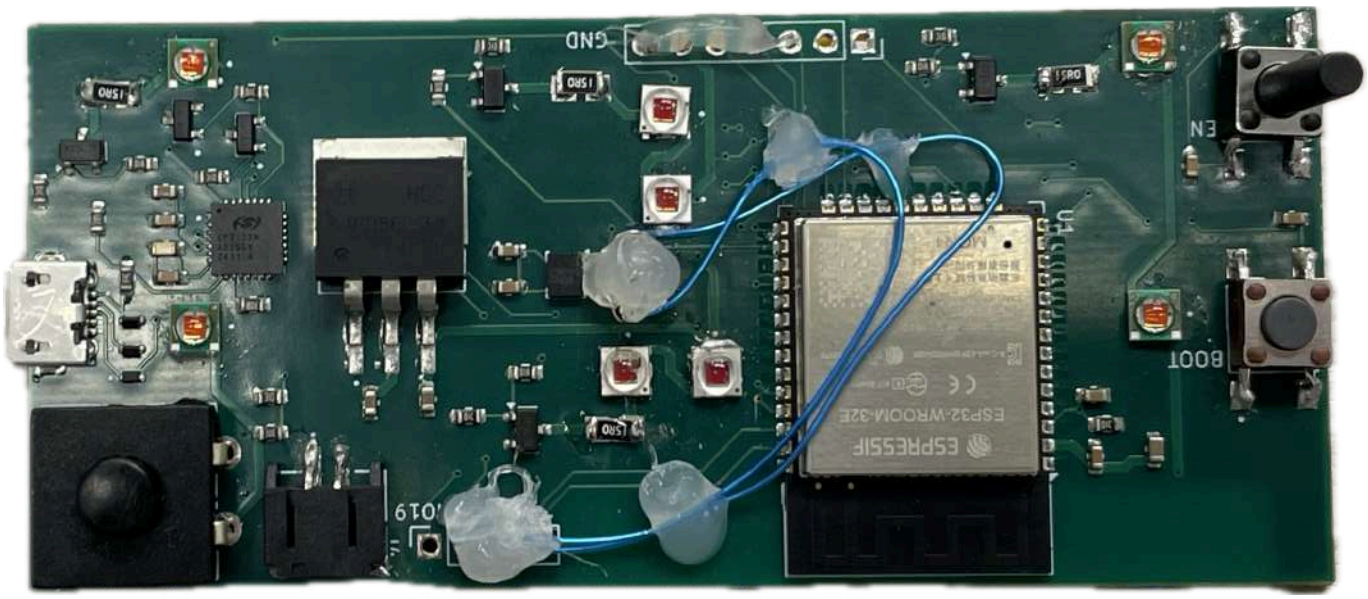

Figure 7: Rear PCB

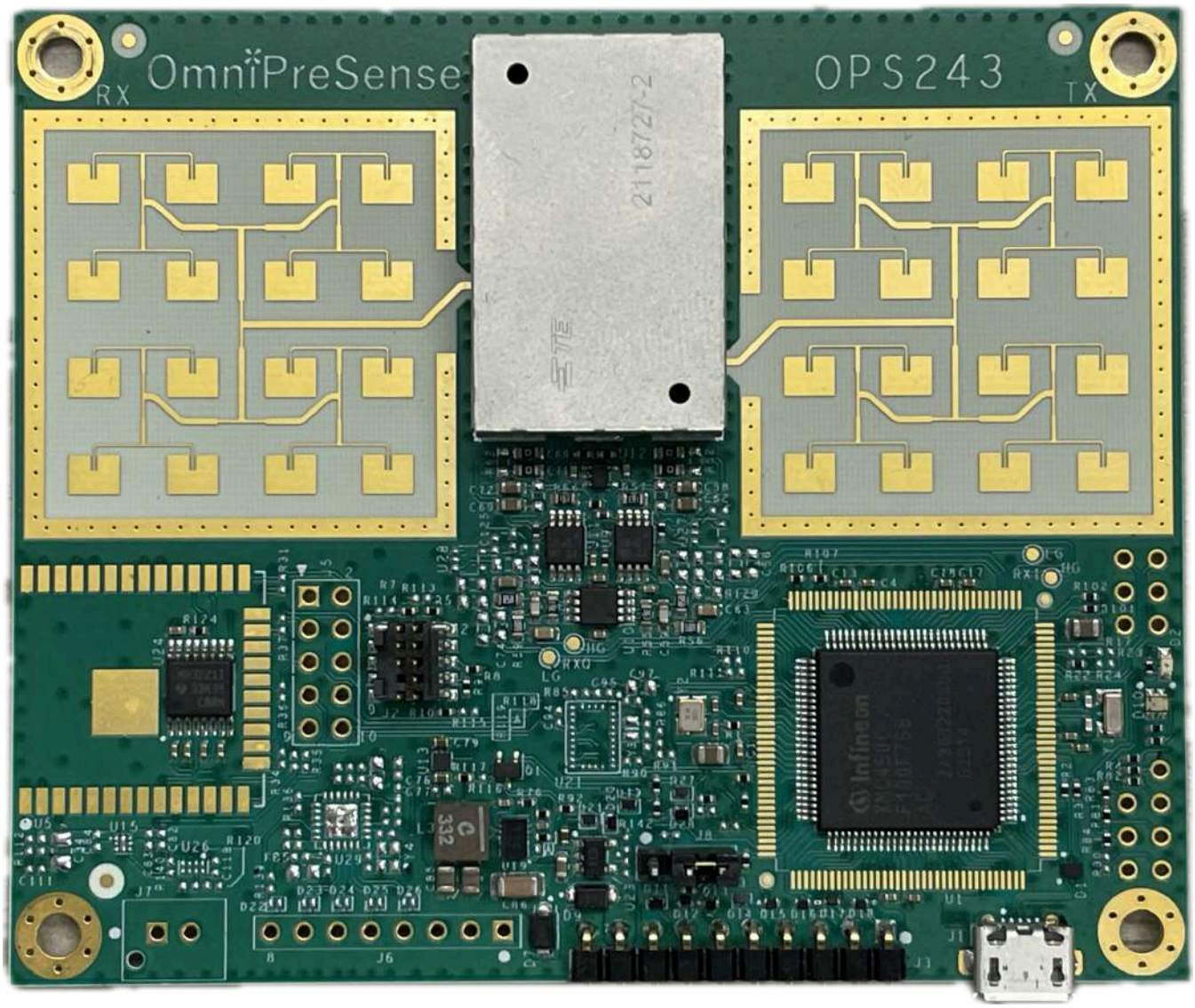

Figure 8: OPS243-C Doppler & FMCW Radar

Acknowledgements

We want to give a special thanks to the following people/groups for their contributions to our project:

John McCloskey for allowing us to use his 3D printer to print all of our case designs.

Neil Kolban for the example we based our BLE code off of (https://github.com/nkolban/esp32-snippets/blob/master/cpp_utils/tests/BLETests/SampleClient_Notify.cpp)

Rob Tillaary for the library we used for our median filter (https://github.com/RobTillaart/RunningMedian)

Adafruit for the library we used to communicate with our IMU (https://github.com/adafruit/Adafruit_LSM6DS)