Pulse

Final report Engineering Design (4WBB0)

October 30, 2023

Group no: 192	
Name	**Student ID**
Axel Reitz	1853945
Daniel Tyukov	1819283
Matyas Szabolcs	1835521
Mukil Krishna Balamurugan	1803298
Remo van Rijswijk	1830627

1 Group effectiveness

1.1 Group composition and strengths

The team consists of students from the mechanical engineering, electrical engineering, industrial design, mathematics and computer science. Some team members had great expertise in the field of programming and back-end development, knowledge that was of great use for solving problems during this project. The mechanical engineers worked to ensure that the physical components would be sturdy, ergonomic, and aesthetically pleasing. The industrial design member brought in an essential perspective, ensuring that the product was not only functional but also user-focused. Drawings of the desired design for the device were delivered as sketches to accomplish this. The mechanical engineering student then converted these drawings into digital components that were 3D printed. For a visual overview of the group composition, see Figure 1.

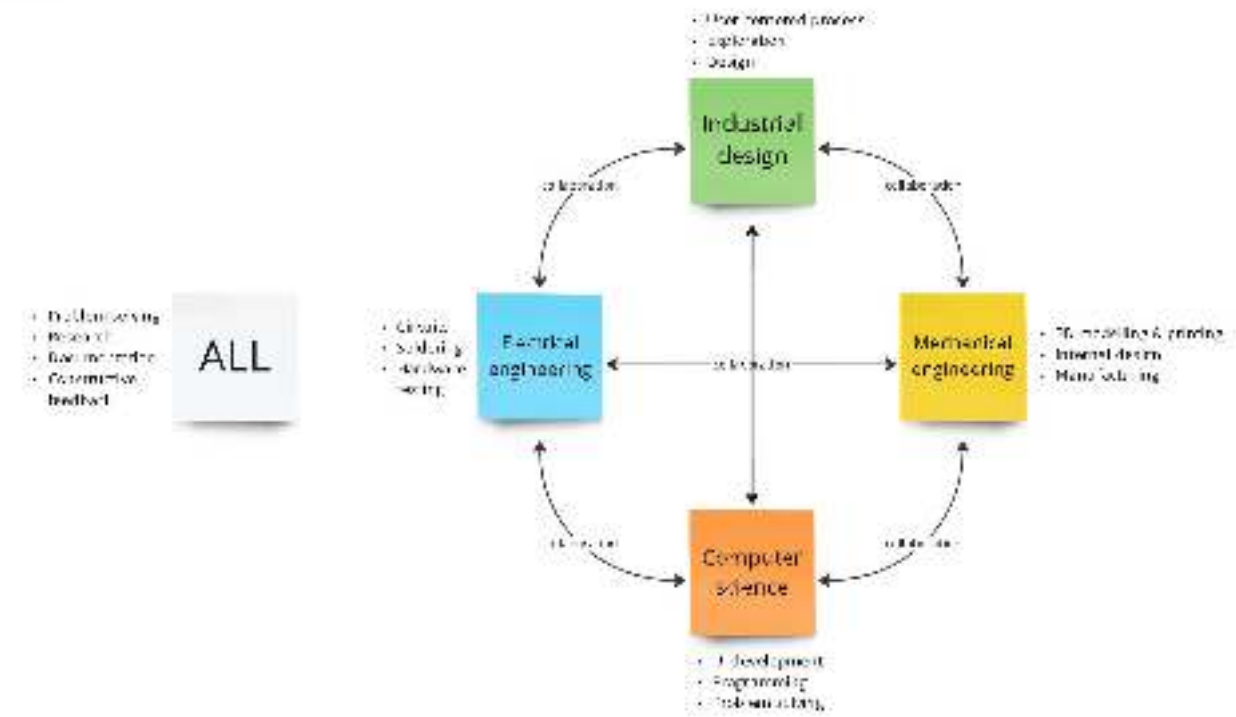


Figure 1: Overview of contributions

Throughout our project, we emphasized iterative design. Every prototype and design decision was subjected to group scrutiny, leveraging our diverse backgrounds. This iterative feedback from students with different expertise areas, combined with real-world testing, allowed us to refine "Pulse" into a product that we believe meets our design goal.

1.2 Weaknesses

In the beginning of the project, the team was eager to immediately jump on a single idea. Ideation was a bit limited, and not much effort was put into working together to come up with an innovative design goal; it was more of a competition. As the team was primarily composed of highly technical engineers (which also has its advantages), out of of the box thinking was difficult in the beginning. Individuals thought in limitations, rather than in opportunities. Moreover, as caused by the competitive atmosphere within the team, it was harder for those with the quieter voices to equally communicate their thoughts during this phase.

1.3 SSA's

The SSA tasks that were determined at the end of each meeting helped to properly organize and divide tasks. They encouraged discipline, as every team member was required to present their deliverable during the meetings. This set the bar high, and the project progressed at a high pace.

1.4 Team meetings

Generally speaking, the team was very active and motivated during meetings, something that the tutor can confirm. In the beginning, meetings were however less organized and way more chaotic. Between individuals, there was a great difference in who spoke up too much or too little during meetings. Sometimes, this might have caused some people not to be able to voice their thoughts and opinions sufficiently. This could have come from the fact that everybody wanted to proceed with and push their own concept.

Luckily, this issue was quickly identified and addressed during the feedback sessions. More effort was put into sticking to a set meeting structure and giving everyone equal opportunities to speak up. Meetings became more fruitful, professional, and inclusive. The atmosphere significantly improved as well. Nonetheless, as a team, we had to be careful not to relapse into chaotic meetings. This was one of the primary tasks of the (revolving) chairman.

1.5 Communication and Collaboration

Decision-making was done by considering all arguments and opinions and then collectively deciding on how to continue. The group also employed a WhatsApp group, a Drive folder, a GitHub repository, and a Discord server for both documentation and communication. Communication and collaboration gradually improved throughout the project and the team unanimously agreed that the group's effectiveness and productivity were on a high level.

The design and visualization of the final "Pulse" casing were created in collaboration with the whole team, where aspects such as expected size measurements were considered beforehand as the actual electronics were developed. This showed proactive and exploratory thinking not confined to what already existed, and demonstrated proactive communication between team members.

1.6 Summary

In conclusion, our group's effectiveness was a blend of our diverse skills, a structured approach to meetings, open communication channels, and an iterative design philosophy. While we had our challenges, our proactive approach to addressing them and our commitment to the project ensured that we remained on track and produced a product we are all proud of.

2 Design goal

2.1 Design brief

The team was briefed on designing a product or system that promotes sustainable behavior. Sustainability is defined as *"the use of environment and resources to meet the needs of the present without compromising the ability of future generations to meet their own needs"* [11]. Within the team, promoting bicycle usage was a recurring theme during the first meetings. Cycling is one of the most sustainable ways of transportation [8] and in the Netherlands, almost 60% of citizens between 18 and 64 years old have reported using their bicycle more than twice a week [4], making the Netherlands the undisputed cycling capital of the world.

2.2 Problem statement

Thus, it was decided to design for the promotion of bicycle usage, which includes improving safety and desirability for cycling. After brainstorming, a key area that was found to have potential room for improvement was bicycle navigation. Traditional bicycle navigation systems rely on auditory and/or visual cues, whether someone is using Google Maps on their mobile device or a dedicated navigation device. Visual instruments shift attention away from traffic, while voice instructions can be experienced as distracting. [3] When combined with over- or in-ear headphones, ambient sounds and warnings are blocked as well [6], potentially causing diminished situational awareness. Safety issues concerning the substantial dangers of phone use while cycling [1], and hands-free cycling should also be considered.

For the deaf community and those with hearing impairments, bicycle navigation using auditory cues is logically even more challenging or even impossible. [9] In the Netherlands, approximately 1.5 million people are hearing-impaired or deaf [10]. Extrapolating this number to hearing-impaired cyclists, it could be estimated that hundreds of thousands of cyclists in the Netherlands are unable to utilize auditory bicycle navigation. A visual overview of our problem statement can be seen in Figure 2.

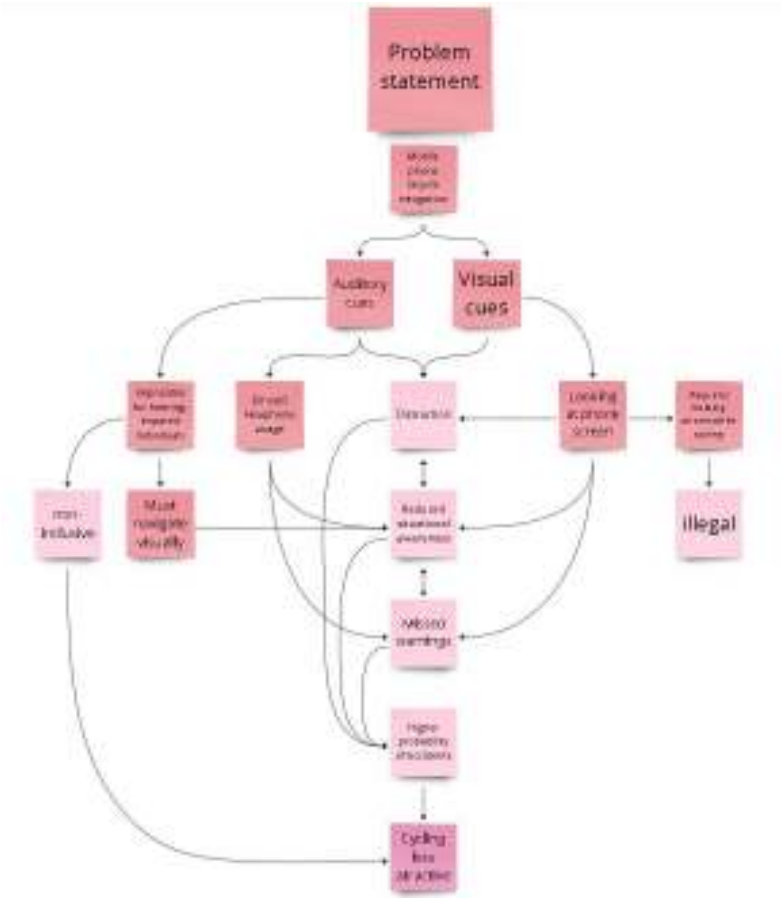


Figure 2: Problem statement

2.3 Design goal

To tackle the aforementioned issues associated with traditional bicycle navigation systems, the design goal is to design a user-friendly bicycle navigation system using just haptics to provide directional cues, thus benefiting both accessibility and safety when navigating on a bicycle [7]. Figure 3 presents the decision process of the design goal and the other themes and concepts that were considered.

2.4 Team expertise and complexity

Pursuing this design goal aligns with the team's expertise areas as it combines knowledge and experience in electronics, programming, and user-centered design. The design goal is therefore realistic, but as the team has concluded, considerably challenging. Complexity is high as the system has to be responsive in real-time, durable, and pairable with a mobile phone. Establishing wireless connection and converting navigational data to haptics has proven to be particularly challenging.

2.5 Innovative approach to haptic navigation

One of the most distinguishing features of our design is the use of haptic feedback as the primary navigation cue. While most navigation systems in the market rely on auditory or visual indications, our solution stands out by tapping into the sense of touch. This is not just a novelty; it's a direct response to the challenges posed by traditional navigation systems. [13]

Safety: Auditory and visual cues require cyclists to divert their attention, potentially leading to accidents. Haptic feedback, on the other hand, offers non-intrusive cues that can be felt without diverting attention from the road.

Universality: Unlike auditory cues, haptic feedback is accessible to the hearing-impaired community. This makes our solution inclusive, catering to a wider audience. [5]

Intuitiveness: Early user tests have indicated that haptic cues, once familiarized, become second nature. The feedback's directional (left or right buzz) and intensity can convey a range of navigational commands, including turns and stops.

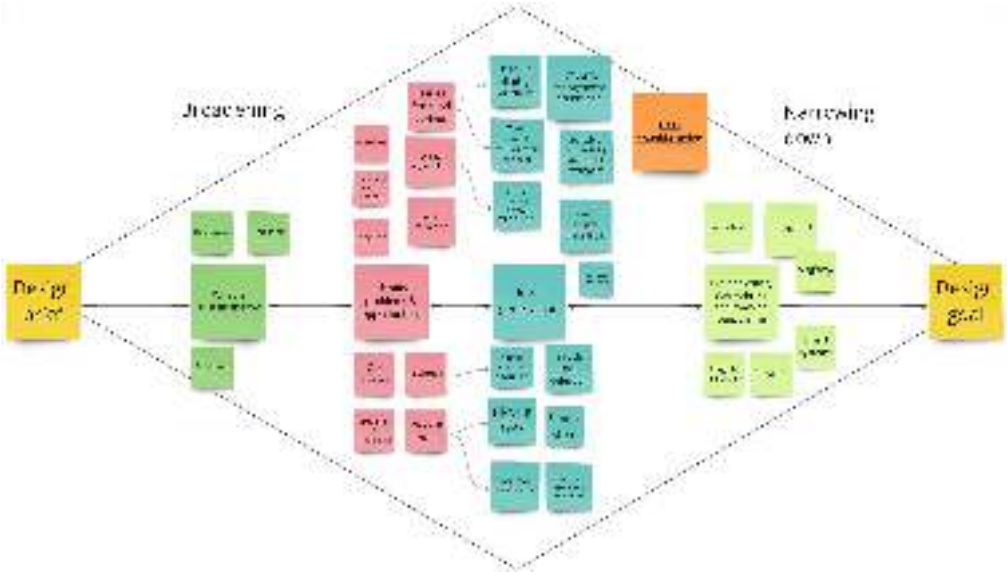


Figure 3: First design stage

3 Functional design and solutions

Using the MoSCoW method, a list of functional requirements is created in this chapter. This prioritizes the functions in order of importance.

3.1 MoSCoW Prioritization:

The design must have:

- A price below 70 euros.
- Clearly noticeable haptic feedback for left and right.
- A smartphone connection.
- A rechargeable battery.

The design should have:

- Indication of distance until turns/actions.
- Durable casing.
- An easy way to attach.
- A functional mobile application.
- Small size.

The design could have:

- An intuitive user interface on the mobile application.
- LED's for extra clarification.
- Waterproofing and dust resistance.
- Made from recycled materials.

The design won't have:

- Voice feedback option.
- Built-in GPS.

The measurable specifications:

- Battery Life: The device should last a minimum of 8 hours on continuous use.
- Vibration Intensity: The vibration motor should have a minimum frequency of 150Hz and a maximum of 300Hz.
- Waterproofing: The device should have an IP54 rating or higher.
- Connectivity Range: Minimum Bluetooth connectivity range of 10 meters.

3.2 Solution encyclopedia:

Price below 70 euros.

- Use only necessary components.
- Opt for multi-functional components.
- Seek cost-effective components.

Clearly noticeable feedback for left and right:

- Use powerful actuators for strong vibrations.
- Test actuators for noticeability.
- Place actuators directly underneath user's hands.

Connection with smartphone:

- Ensure easy setup for the connection.
- Optimize energy consumption for the connection.
- Utilize Bluetooth or Bluetooth Low Energy (BLE) for connectivity.

Rechargeable battery:

- Use a battery that is easily rechargeable.
- Implement a battery management system (BMS) for safety.
- Consider options like Lithium-Polymer and rechargeable AA batteries.

Indication of distance until turns/actions:

- Provide timely announcements for upcoming events.
- Increase the frequency or strength of vibrations as an event nears.

Durable casing:

- Use thick and durable materials.
- Implement structural reinforcements.
- Utilize materials like PLA or printed carbon fiber.

Easy to attach:

- Consider theft-proof attachments like hex screws and bolts.
- Design a universal mounting mechanism for all bicycles.
- Explore options like rubber bands, Velcro, and tie wraps for secure attachment.

Mobile application:

- Design an application to locate and connect with the device.
- Integrate Google Maps for navigation.
- Ensure the app is user-friendly with minimal input requirements, visual hierarchy, clear UI

Waterproofing and dust resistance:

- Use sealed casings.
- Ensure ports, connections, and buttons are sealed or gasketed.
- Opt for water-resistant materials and aim for an IP54 rating or higher.
- Use waterproof charging ports and rubber seals for openings.

LED(s) for extra visual input/clarification

- Linear, circular or array of LED's.
- Integrated (multi-color) LED strip.
- Animations/patterns/blinking of these LED's to communicate instructions.

4 Concept designs

4.1 Vibrating shoe concept

The first idea was to create a device that would attach to a shoe. To signal a turn, this would vibrate at different frequencies (faster if you are closer). The user can navigate through a city without looking at their phone thanks to this device's connection to Google Maps, which converts the directions from the application into vibration.

Figure 4: Shoe concept

4.1.1 Advantages

One of this idea's main benefits is the fact that the device can be used both when riding a bicycle and walking. The vibration motors would be positioned beneath the shoe's sole. As depicted in Figure 4, the other parts of the device would be attached to the exterior of the shoe.

4.1.2 Disadvantages

One of the biggest drawbacks is that walking or cycling generates a lot of vibrations, some of which could interfere with the vibration of the device itself. This could make the vibrations of the device not noticeable. [13] This would defeat the device's sole purpose of making vibrations. A connection between the two shoes is also necessary to indicate which shoe vibrates to indicate the correct direction (left or right). Cables connecting the motors to a single hub could not be used for this connection since then the user's legs would be tied together, which is not very practical. A wireless Bluetooth connection would be another option, but that would exceed our budget.

4.2 Vibrating watch concept

The second concept was to reduce the size of the vibrating device so that it could be worn around the wrist like a smartwatch. This might stop the device's vibrations from being interfered by the vibrations caused by cycling and walking. The functionality of this device would be identical to that of the shoe concept device. This implies that it would vibrate if a turn were approaching, and the vibration frequency would get higher as you got closer to the turn. This device would also obtain navigation information from Google Maps and would transform notifications into directions, which would then be transformed into vibrations.

Figure 5: Watch concept

4.2.1 Advantages

The smaller size of the device would make it easier to wear, which is its biggest benefit. The fact that the user would wear this device on his wrist makes it easier to feel the vibrations, which is another significant benefit. Additionally, simply putting it on the wrist is more intuitive than attaching a device to a shoe, for example.

4.2.2 Disadvantages

To provide directions, the main hub of the device has to be able to connect to the other wristbands. For the device to use Google Maps, it would have to be able to connect to a phone and the wristbands. Unfortunately, the cost of making this would exceed our budget of 70 euros. Another problem with this approach is that this gadget would be required to be extremely small to be worn comfortably on the wrist, which, again, with the resources and budget at hand, is not feasible.

4.3 Bicycle concept

The third concept is to mount the device to a bicycle. The only difference between this approach and the other two concepts is that this one only has one main device. This is feasible due to the device's ability to be mounted in the middle of the steering wheel, from which small vibration motors can be extended to the steer's ends (see Figure 6) to vibrate the left and right hands. This device is also additionally integrated with Google Maps, it provides notifications via vibrations at each end of the steering wheel.

Figure 6: Bicycle concept

Figure 7: Variation of bicycle concept

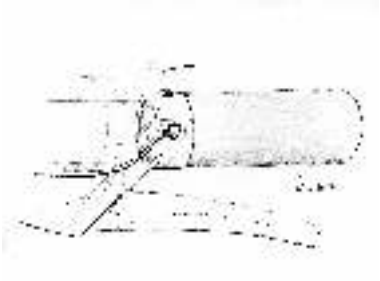


Figure 8: Design option for buzzer attachment

4.3.1 Advantages

The main benefit of this concept is that there is only one main device, which eliminates the need for a Bluetooth connection between the various parts of the device, and only the Bluetooth connection with a smartphone is needed for sending directions obtained from Google Maps. This also helps in lowering the cost of manufacturing the prototype within the limit of materials and budget. Another benefit is that it transmits vibrations directly to the bicycle handlebar. This enables the user to feel them more clearly, so they wouldn't miss a signal due to the bicycle's vibrations. Another point is that there would be more room available on the steering wheel than in a

wristband or on a shoe for the device to be placed. This gives us more creative freedom with designing and is more practical.

4.3.2 Disadvantages

The main drawback of this device is that it only focuses on bicycles and is not suitable for walking. Additionally, the vibrations produced by the vibration motor may not be powerful enough to exceed the vibrations produced by the bicycle itself.

4.4 Application concepts

4.4.1 Google Maps API

The initial idea was to receive navigation details from the user, including the starting point and the end destination directly in our application, and then use the obtained information to find the necessary navigation information such as the route taken, and directions using the Directions API provided by Google Maps. The navigation details would subsequently be transmitted to the device through Bluetooth Low Energy (BLE).

Advantages

Using this approach provides the advantage that the user can solely rely on our application for device use, eliminating the need to switch between our application and Google Maps. As a result, the user's experience is improved.

Disadvantages

The Directions API and all other APIs offered by Google require payment after the free trial with a fixed number of navigation requests, This means that if the product is ever to be used for commercial purposes this is an added cost that would have to be taken into consideration. This also complicates the application as we need to convert the starting point and final destination to latitude and longitudinal coordinates before being able to use it.

4.4.2 Listening to Google Maps notifications

The second proposal aimed to use the NotificationListener Java package to break down Google Maps notifications and transform the data obtained from it into a message that can be decoded by the device. The data would be transmitted to the device via BLE.

Advantages

The advantage of this approach is that it would be cheaper to implement even on a commercial scale as we are not using any paid APIs. In addition, the implementation of the application is made easier by avoiding the need to handle user input of the start and end destinations and convert it into the desired format for obtaining navigation information. This removes a layer of complexity from the coding, allowing it to be completed within the given time frame.

Disadvantages

The disadvantage of this method is that the user must use Google Maps and our application to use the device. Also, when the application reads the notifications, it needs to be able to understand and translate images into usable information, as Google Maps notifications only represent directions in the form of images, not words.

5 Final concept design

Both advantages and disadvantages were taken into account. Certain disadvantages were more significant than others, for instance, the device's applicability was given less weight than its feasibility. When examining the wristband and shoe concepts, it became obvious that they would be challenging to implement because they would require two devices that should be able to communicate both with the other device and a smartphone. This is too difficult to produce and manufacture within the budget limit. The device's necessary space is however another crucial factor. In comparison to the space available on the bicycle device, the space available for the shoe and wristband design is significantly less.

5.1 Final concept

The third concept was eventually selected because its benefits far outweigh its disadvantages. This is not the case for the other two concepts. It can provide haptic feedback and connects to a smartphone to get navigational instructions. Because there is a lot of room to work on the bicycle's steer, the case can be made to be durable. Similarly, after weighing the advantages and disadvantages of the two methods of the application's navigation aspect, the second option was selected as it is both feasible to implement within the given time frame and entirely free of charge.

5.1.1 Innovative technology

There is nothing like this device on the market, making it cutting-edge. Everything available focuses on either voice feedback or a screen. This device is completely different in that regard because it prioritizes vibrations over sound and vision. This is necessary so that you can keep your eyes and ears on the road.

5.1.2 Challenging

Producing the device is difficult because it needs to be able to connect to a smartphone that can read directions from Google Maps. The device will only make up half of the final product because a smartphone app will be required to implement notifications from Google Maps and send them to the device, which will then need to know which side of the steering wheel should vibrate and at what frequency. The difficulty in reading Google Maps arises from the fact that in Google Maps, directions are always represented in the form of images. This makes it impossible to immediately use it and instead is required to convert the image to a form that is understandable by the computer. Similarly, since we are connecting to the device using a BLE connection instead of a regular Bluetooth connection we cannot use the standard Bluetooth scanner present in all mobile phones instead we need to create our own scanner to connect with the device. This added complexity is worth it as the name suggests as a BLE connection consumes significantly less power which is important to keep this product as sustainable as possible and also gives the device a longer battery life.

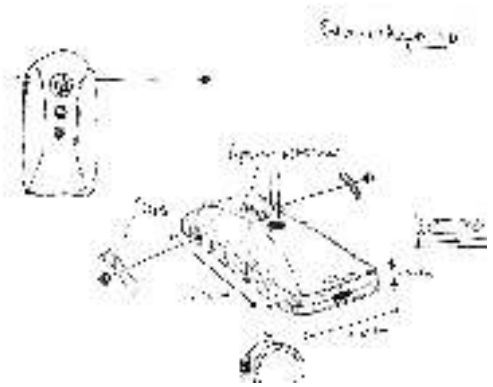
5.1.3 User-friendly

The device should be as simple to use as possible. This means that setting up the device to connect to a smartphone shouldn't be difficult, and the connection should happen automatically when a Google Maps route is launched. When the application functions properly and all necessary permissions are granted, this is possible. When this is the case, connecting is straightforward. The device itself can be left on the steer as it is housed in a durable case and screwed to the bicycle's stem. The actuators are integrated with the steering of the bicycle thanks to their attachment to the handlebars. Because they are too large to fit on a wrist or the outside of a shoe, the other two concepts are not as user-friendly. The application is also designed keeping User-friendliness in mind. It contains several indicators that let the user know if all connections are working perfectly. It contains a shortcut that lets the user easily give notification access and the BLE scanner is also built into the app making it easier for the user to connect to the device.

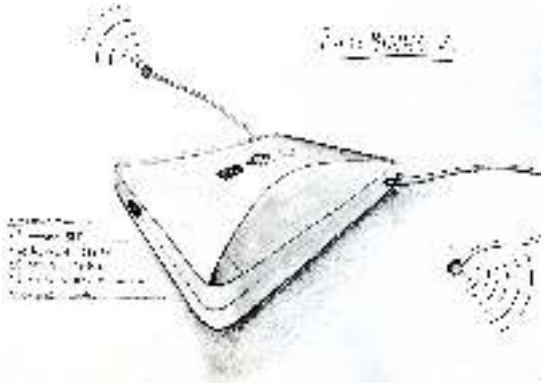
5.1.4 Preliminary design

To move from a sketch to a preliminary design, some initial drawings (see Figure 9a) were made. This was carried out to first make it as user-friendly as possible before modifying it to fit the electronics that need to fit inside the device. The device can be designed nicely while also considering the specifications and dimensions of the electronics inside. The result of this is an aesthetic-looking device that clamps onto a bicycle steer in the preliminary design. The vibrators are connected by long cables that extend to the steer's ends. The device's top features a switch

and a tiny hole with an attached LED light that indicates whether the device is active or not. The device has a USB-C port on the side that can be used to charge it.



(a) Sketches



(b) Preliminary design

Figure 9

(a) Final design

(b) Actual device

Figure 10

6 Technical specification

6.1 Functional Specifications

These specifications include the features that are necessary to help the device achieve the essential technical requirements. They are important to provide a safe, hassle-free experience to the user. They help to create an overall pleasant experience for the user which is an essential aspect to take into consideration to achieve the goal of creating a successful commercial product.

- **Not be a hazard to the user:** The non-conductive casing and insulated wires provide electric insulation eliminating the risk of potential electrical shocks. The device also contains protective measures that ensure the micro-controller current with stable voltage. The device also has protective measures that ensure safe charging of the battery.
- **Intuitive UI:** The pulse Navigation application has an easy-to-use user interface that provides a hassle-free experience to the user. The user can use the application to easily connect to the device via Bluetooth (BLE).
- **Compact casing:** The compact casing makes it lightweight and easy to carry. It does not hinder the cyclist's riding experience.
- **Easy to attach:** The Device is easy to attach with a universal ring that can be placed around the stem. The actuators are attached with the use of Velcro. This makes it easy for the user to attach and detach the haptic feedback system to the handle of the bicycle. It also allows the user to choose in which area of the handle they want to attach the device. This avoids any riding inconveniences that might have been caused by a more rigid device.
- **Durable Structure:** The device will be attached to a bicycle and will be used outdoors. Thus, the device was built with durability in mind. It has a tough 3D-printed case made with Poly Latic Acid (PLA) and has strong wires alongside a durable Velcro strap. The use of a rubber piece in between the attachment piece and the stem of the bicycle protects the device from the biggest bumps.
- **Able to withstand external elements (Dust/ Water Resistant):** The device will be used outside exposing it to the challenges caused by nature. Therefore, the device needs to be able to withstand these factors. The watertight PLA casing keeps the internal components protected. Using the definitions provided by the IP Rating Form from Alpeco [2], we were able to define the IP54 rating for the device by following the testing instructions provided.
- **Indicators that show the status of connection:** The application requires multiple simultaneous permissions and connections to function. The indicators make it easier for the user to check that they have given the application the necessary permissions and connections. Figure 11.

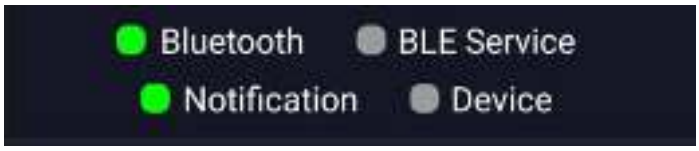


Figure 11

- **Good Battery Life:** The device has an 800 mAh Li-Po battery which provides the device with a battery life of 8 hours of continuous vibrating of both actuators. The use of (BLE) i.e. Bluetooth Low Energy also allows the device to have a long battery life. In real-life situations, only one of the actuators will vibrate when the navigation information prompts an upcoming turn. This means that a noramal user can use the device for several weeks without having to recharge it.
- **Noticeable feedback:** When riding a bicycle it is important to quickly understand navigation cues to make timely responses. The device provides the user with clear, easily decipherable feedback with vibrations at just the right intensity.
- **Rechargeable battery:** The 800 mAh Li-Po battery is rechargeable using a USB-C Battery management system TP4056. This makes the device easily rechargeable and user-friendly because the battery does not have to be replaced or taken out of the device.

- **Shortcuts to access Google Maps:** The application has a button that acts as a shortcut that takes the user directly to Google Maps. It also has a button that allows the user to end Navigation directly through the app without even opening Google Maps. These features generally improve user experience.

6.2 Technical specifications

These specifications include those that are important in the main working of the device. They include the essential features that perform the necessary actions that make it a Bicycle Integrated Navigation and Haptic System.

- **Clear Haptic feedback:** The use of actuators attached to the bicycle handles provides clear and timely Navigation Directions. The Arduino Nano uses the information received to generate an appropriate vibration. The perfect vibrating intensity has been set to make it clearly decipherable at the same time not overdoing it which might make it a nuisance. This tone has been selected with rigorous testing.
- **Able to send Navigation information to the device via Bluetooth:** The Arduino by itself does not have Navigational capabilities thus it is dependent on a mobile phone to provide it with suitable information that it can then use to provide timely feedback to the user. This is accomplished using the Application which has a feature that allows it to connect to the device via Bluetooth. It can then decode the notification from Google Maps and send only the necessary information. The embedded code then allows the Arduino to accept this information and then translate this information which it uses to provide timely haptic feedback to the user.
- **Parse Google Maps Notifications:** The app uses the Android Notification Listener Service to read the Google Maps notifications. It then converts this information into usable data that can be sent to the Arduino. The Application converts the direction images into a bit code which can then be used to determine if the user must go left, right, or straight.
- **Less than 70 euros:** The device costs less than 70 euros due to the use of 3D-printing which makes the case cheap. 35 Euros are spent on the microcontroller and the battery. The remaining electronic parts were inexpensive, keeping the device well within the budget.

7 Detailing

7.1 Pulse Navigation Mobile Application

The four main functionalities of the mobile application include:

- Determining the direction of and distance to the next turn the user is advised to take by Google Maps
- Converting this information into a simple message that can be decoded by the Arduino
- Connecting to the device via Bluetooth Low Energy
- Sending this message to the device every time the app receives an update from Maps

7.1.1 User interface

The user interface was designed to be simple to use, therefore upon opening the app, in the center of the home page, the user is greeted with simple instructions to operate it.

The UI provides information on the current status of the connection to both the device and Google Maps. These status indicators are located at the bottom of the main navigation page and indicate whether the following four main conditions to operate the device are met:

- Bluetooth connection is enabled on the mobile phone
- The phone has access to a Bluetooth Low Energy service
- There is a visible Google Maps notification on the phone
- The Pulse device is powered on and is ready to connect

At the very bottom of the screen, there are three buttons, namely:

- "Notification Settings", which acts as a shortcut for the user to enable the Pulse app to read notifications pushed by Google Maps
- "Scan", which takes the user to the scan page of the app, where they can view all available devices with a BLE connection, and connect to the Pulse device (ESP32)
- "Connect to Previous", which makes it simpler for the user to connect to the device since the previous address is always saved

Last but not least, there are two test buttons at the sides of the screens, which will be discussed in more detail in the testing section of the report.

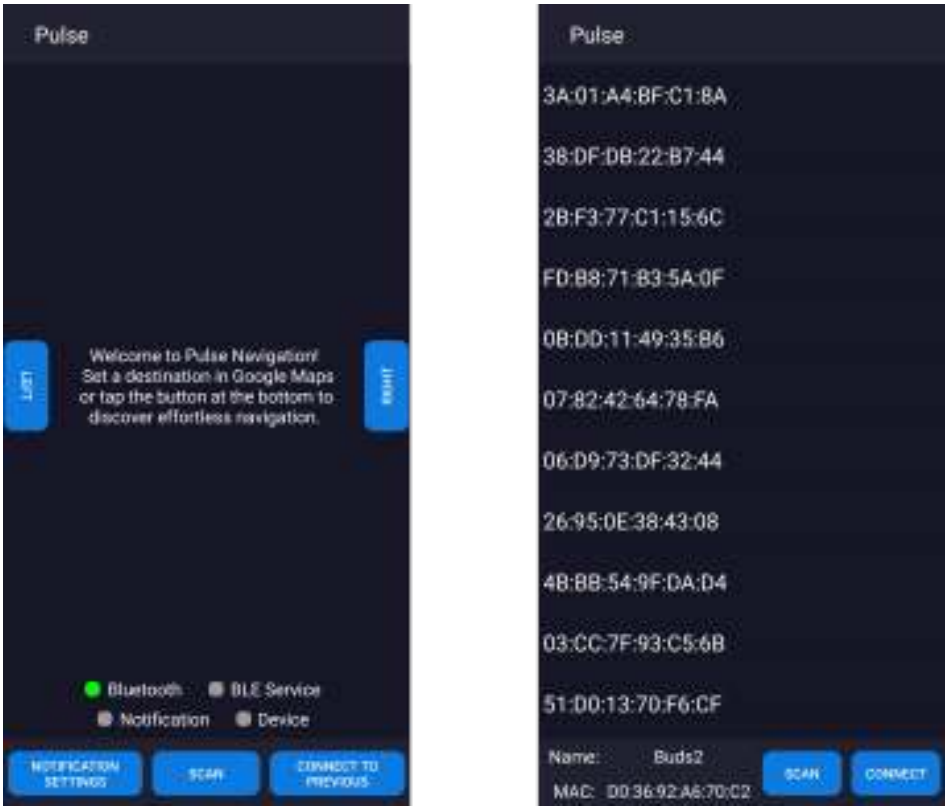


Figure 12: Home page (left) and Scan page (right)

7.1.2 Notification system

The notification system was built using a simple Java library called NotificationListener, which was used to extract all the data available in the notification set by Maps. After fetching the notification data, it is converted into a short string that can be simply sent to the Pulse device through BLE, in the following way:

1. The distance to the next turn is converted to meters (since for long distances Maps displays the distance in kilometers).
2. The actual direction is obtained from decoding the icon that is displayed inside the notification
   (a) The icon is first converted to a bitmap
   (b) The bitmap is converted into a byte array
   (c) the byte array is converted into a hexadecimal number, which is then converted into a string
   (d) The code checks if the hex code of the icon matches any of the already known codes
   (e) If the code is known, the program returns the respective direction, otherwise, it returns "unknown"
3. The shortened direction (e.g. "s" for "straight) and distance are combined into a single string, separated by a colon (:). For example, "r:120" means "turn right in 120 meters".

7.1.3 Bluetooth connection

The application utilizes Bluetooth Low Energy (BLE), which is a power-efficient version of the classic Bluetooth technology designed for short-range communication between devices. It is commonly used for applications where low power consumption is critical. In our case, BLE is used to send our message (explained in the Notification system section) to the Pulse device.

7.2 Case Design

The main device and the attachment part were the two components that were designed separately. The primary device should be able to do the following tasks: house the electronics, have an on/off switch, an LED light slot,

and a charging port. Together with being as small as possible, the device should also look good. The first sketches (Figure 9a) were used to make the first design for the 3D model.

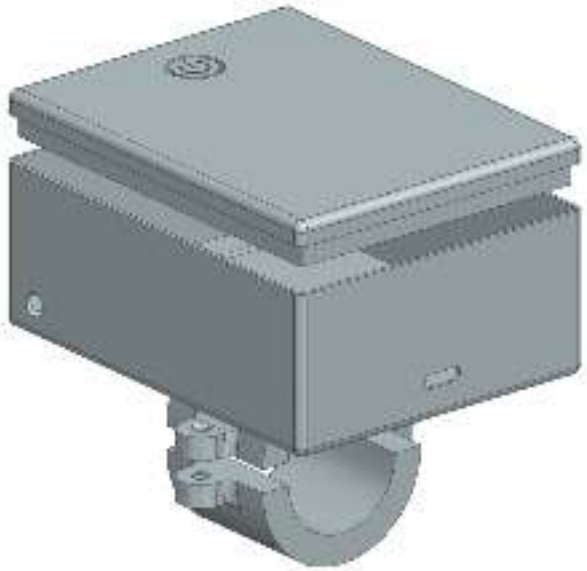


Figure 13: First 3D design

The attachment of a bicycle bell served as the inspiration for the design element visible at the bottom of the device. The actuators' attachment part was made using the same concept.

(a) Actuator attachment

(b) Bicycle bell

Figure 14

The actuator attachment was designed for attaching the actuator to the bicycle's handlebars. Figure 13 illustrates how this operates as the same concept for the main device. There are two half circles positioned around the bicycle's steer. Two bolts and nuts are used to join these two components. This method of fastening the components to the steer is simple but secure.

7.3 Battery Research

To ensure the device operates efficiently, particularly in terms of energy consumption, it's imperative to understand the power requirements and battery life. Below is a detailed exploration.

7.3.1 Battery Life Calculation

The average current consumption of the ESP32-S3 is around 40mA when active. Additionally, the motors use around 60mA when active. Given these values, the estimated battery life can be calculated as:

$$\text{Battery Life} = \frac{\text{Battery Capacity}}{\text{Average Consumption}} = \frac{800\,\text{mAh}}{100\,\text{mAh}} = 8\,\text{hours}$$

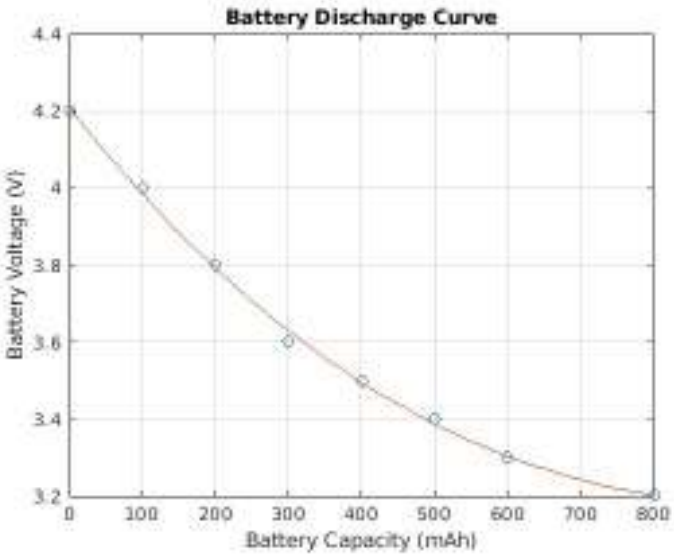
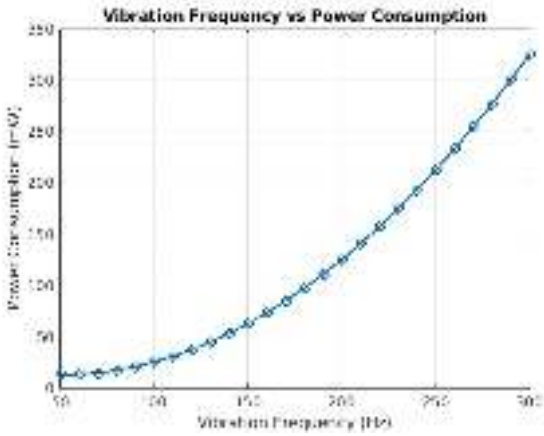


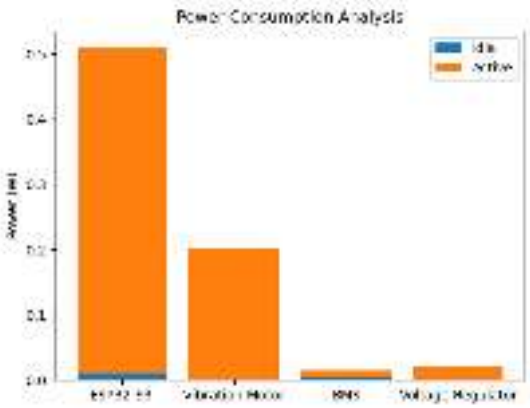
Figure 15: Battery Discharge Curve

7.3.2 Voltage Regulation

The LM1117T 3.3V is employed to ensure that the ESP32-S3 consistently receives a stable 3.3V supply. This is crucial as the battery voltage might fluctuate between 3.2V (when discharged) and 4.2V (when fully charged). (This was removed for the final prototype after real-world testing, but should be explored further in production).



(a) Vibration Frequency vs Power Consumption



(b) Power Consumption Analysis for Components (ESP32-S3, Vibration Motor, BMS)

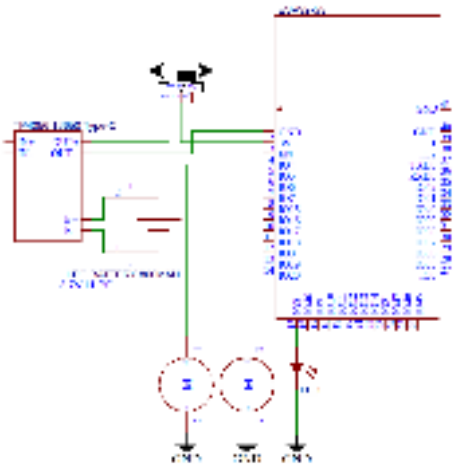
Figure 16: Combined Figures of Vibration Frequency vs Power Consumption and Power Consumption Analysis

8 Realization

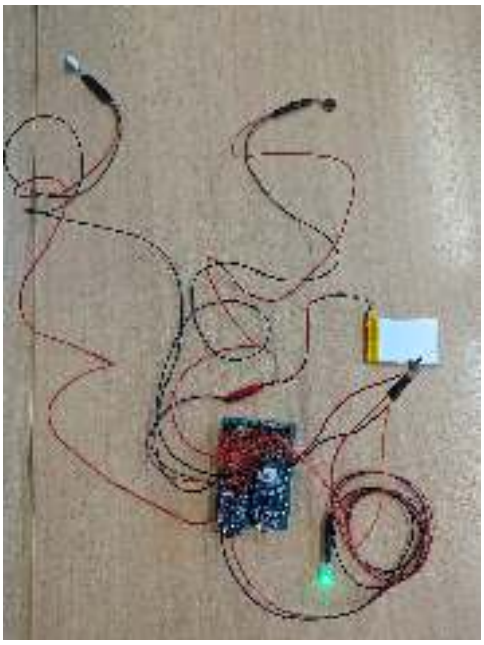
8.1 Electronics

8.1.1 Circuit Design

The circuit for Pulse was meticulously designed to ensure efficient power management and robust communication. The integration of the ESP32-S3 microcontroller with Bluetooth capabilities is central to the design. The microcontroller interfaces with the haptic motors and the battery management system to provide accurate haptic feedback. The circuit diagram can be referenced in 17a.



(a) Schematic Representation of the Circuit



(b) Physical Realization of the Circuit

Figure 17: Comparative Figures of the Circuit's Schematic and Physical Representation

8.1.2 Testing

The circuit was first prototyped on a breadboard. This allowed for easy modification and iterative testing. The voltage levels, current draw, and power consumption were analyzed using a multimeter and an oscilloscope.

To determine the efficiency of the voltage regulator, the following formula was used:

$$\text{Efficiency} = \frac{\text{Output Power}}{\text{Input Power}} \times 100$$

Given that the ESP32-S3 consumes around 40mA when active and the motors consume about 60mA when active, the power consumption was:

$$P = V \times I = 3.3V \times 100mA = 330mW$$

The battery's discharge rate, the motor's vibration frequency against power consumption, and the individual component's power consumption were plotted and analyzed. These can be referenced in 15 respectively.

The circuit's reliability was tested under various conditions, such as varying temperature, humidity, and external interference, to ensure the device's robustness in outdoor scenarios.

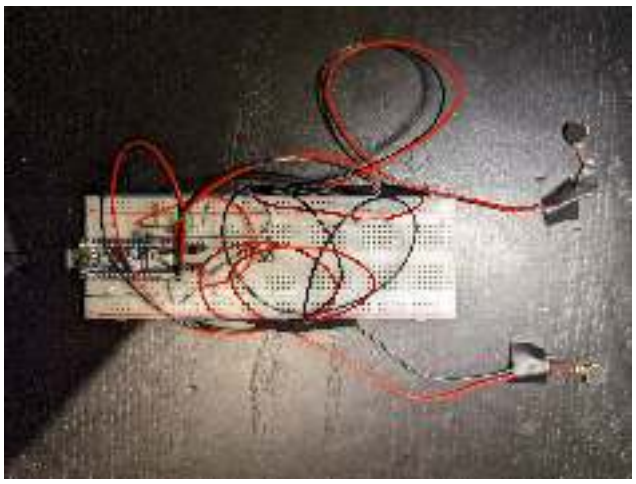


Figure 18: Breadboard Prototype

8.2 Embedded Code

The embedded code is responsible for the core functionality of the device, translating the Bluetooth notifications into haptic feedback for the user. This code is written in C++ for the Arduino platform and specifically targets the ESP32-S3 microcontroller.

8.2.1 Key Functionality

The software primarily listens for incoming BLE signals from the paired mobile device. Upon receiving a message, it decodes the navigation information and triggers the appropriate haptic feedback on the device.

For instance, if the navigation information indicates a left turn, the left-side actuator will vibrate. If it indicates a right turn, the right-side actuator will vibrate. The intensity and duration of the vibration can be adjusted based on the distance to the turn, providing more immediate feedback as the turn approaches.

8.2.2 Notification Listener

The notification listener is a critical component of the code. It uses the NotificationListener library to fetch the Google Maps notifications on the paired mobile device. Upon receiving a new navigation update, it processes this data, converts it into a simple message, and sends it to the Pulse device via BLE.

8.2.3 Haptic Feedback Control

Upon receiving a BLE message, the code processes the message to determine the type of haptic feedback required. It utilizes the Arduino's built-in functions to control the actuators. The duration and intensity of the feedback are controlled by Pulse Width Modulation (PWM), allowing for nuanced feedback based on the navigation data.

8.2.4 Safety Features

To avoid redundant feedback or overloading the user with too many vibrations, there's a built-in delay mechanism. If a new BLE message arrives too soon after the previous one, the new message is ignored, ensuring the feedback remains clear and comprehensible.

For a detailed look at the embedded code, please refer to Appendix A.2.1 A.2.

8.3 Assembly

The assembling of the final product was simple after the electronics were soldered as shown in 17b. The casing of the device is made of three separate parts. The main device, the lid, and the attachment ring.

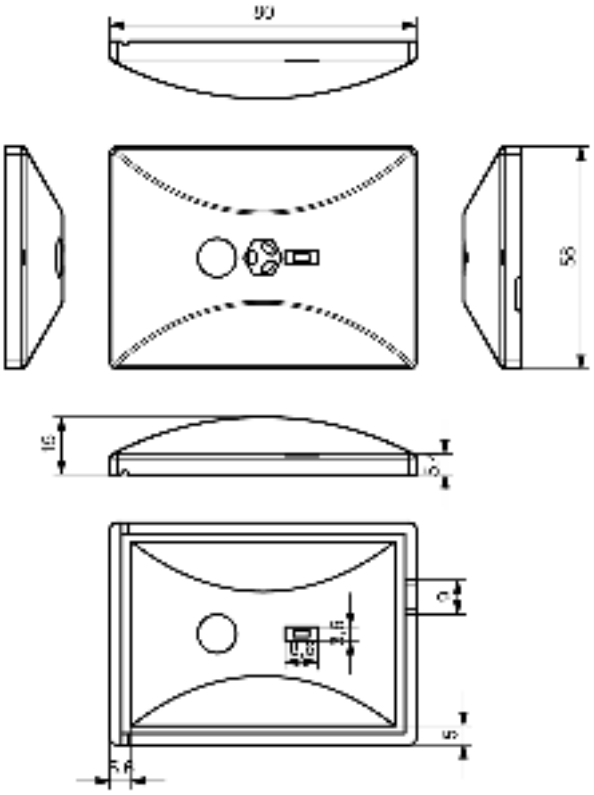


Figure 19: Technical drawing lid

The device has a round and aesthetic lid with a few key features. A circle has been cut out on top of the device to accommodate the LED light. The product's logo appears next to it. Adjacent to it is a slot designed to place the on/off switch in. A tiny half-circle can be seen at the top of Figure 19. This is where the wire to the actuator is supposed to go. The USB C port's half-shape is visible on the left. These shapes are divided in half because the main box's lid and the main box are keeping the cables and USB-C port in place when closed. The decision was made because, once the device was soldered together, the cables would not have been able to pass through the holes. The use of two half circles solves this issue because they allow the cable and USB-C port to fit between the main device and the lid.

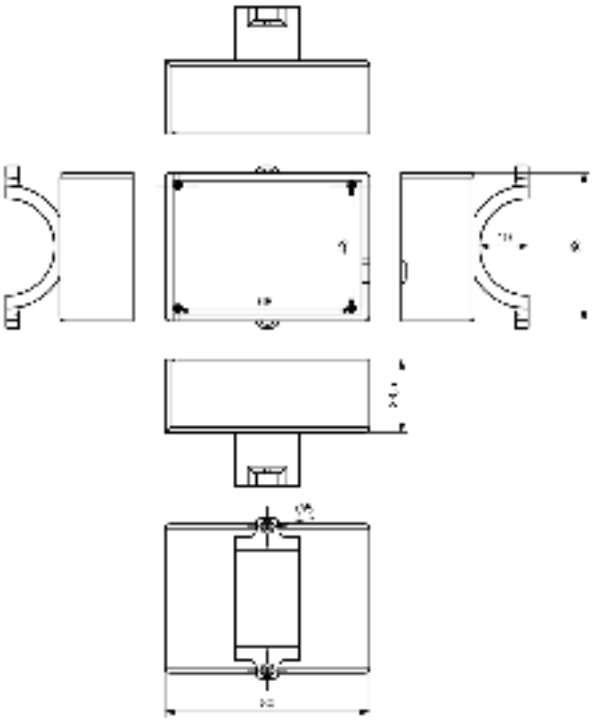


Figure 20: Technical drawing main device

The main device also displays the half circles and the half shape of the USB-C port. The electronic breadboard fits perfectly inside the main component, which is shaped like a box. To guarantee that there is room for the battery underneath, there are a few supports designed to raise the breadboard (these are shown as the black circles shown in Figure 20). The device's bottom portion is where it is attached to the bicycle's steering stem. This has some holes on the sides and is shaped like a half circle. These are designed to accept the placement of bolts and nuts for attaching them to the other attachment part.

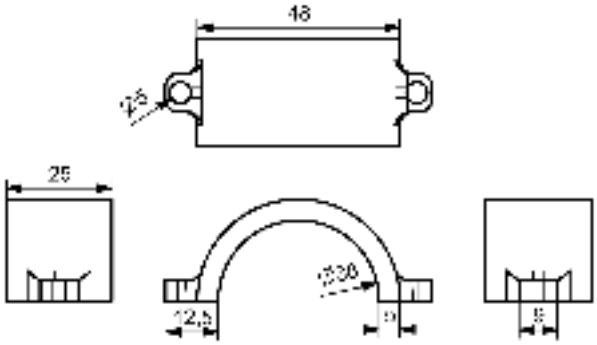


Figure 21: Technical drawing attachment

This part of the attachment is the bottom piece. Its form is identical to that of the main device's attachment part. This facilitates bolted connections between components to seamlessly attach the device to the stem of a bicycle.

Figure 22: Final device

The final device looks like it does in the figure above. The cables are going to be positioned on both the left and right sides of the main box. The figure does not include them. As shown in Figure 22, bolts are used to attach the device. The whole device as shown in the technical drawings and in the figure above is 3D printed from PLA (Poly Lactic Acid) to ensure that the main device is waterproof except for the holes that are made in the device. PLA was not only chosen because of the waterproof characteristic but also because it is light and cheap.

8.3.1 Plan for production

1. Firstly, all the electronic parts should be soldered accordingly Figure 17b and all the 3D printing parts should be printed.
2. The perfboard should be placed in the main device after which the lid should be placed on top and sealed in place.
3. The main device should be placed around the stem of the bicycle with the attachment part under the stem (see Figure 22). It should be secured with the use of bolts and nuts.
4. The actuators should be placed at the ends of the steer and they should be secured by wrapping the Velcro grip around the steer. This ensures that the actuators are right under the hands and this protects the actuators as well.

8.4 Parts List

Item Name	Supplier	Price per Unit (€)	Number of Units	Total Cost (€)
Arduino Nano ESP32 - ESP32-S3	tinytronics.nl	22.5	1	22.5
Small Vibration DC Motor 2.5-4V	tinytronics.nl	1	2	2
Rechargeable Battery (Li-Po 3.7V 800mAh)	tinytronics.nl	12	1	12
Battery Management System (BMS) - TP4056 USB-C	tinytronics.nl	2.5	1	2.5
LM1117T 3.3V Voltage Regulator	tinytronics.nl	1.5	1	1.5
Perfboard for assembly	tinytronics.nl	3	1	3
Switch	tinytronics.nl	0.20	1	0.20
Miscellaneous Components	tinytronics.nl	0	N/A	0
3D Printing	N/A	5	1	5
Bolts and Nuts	N/A	0.5	1	0.5
			Total	**49.20**

Table 1: Bill of Materials (BoM) for the Pulse project

9 Test plan

Each part of the final prototype was put to the test, to ensure the proper intended functionality was achieved. This was carried out independently so that several people could work on the testing at the same time. There are four parts of the device. The hardware, the software component that retrieves directions from Google Maps, the software component that transmits the directions to the device via Bluetooth, and the code that decodes the Bluetooth signals and transmits electrical signals to the vibration motors.

9.1 Hardware Testing

9.1.1 Bluetooth Connectivity

To ensure the reliability of the Bluetooth connection, the Arduino was tested for its ability to establish and maintain a connection with the mobile application. This involved testing the range, connection stability, and reconnection capabilities. Tools like the Serial Monitor in the Arduino IDE were instrumental in monitoring the BLE connection status.

9.1.2 Vibration Motor Testing

The vibrators, responsible for haptic feedback, were tested for their response time, vibration intensity, and energy consumption. A multimeter was employed to measure the current drawn by the motors during operation, ensuring that they operated within the specified limits.

9.1.3 Battery Performance

The battery's longevity, charge retention, and efficiency under different loads were evaluated. This involved running the device continuously and monitoring the battery voltage over time using a voltmeter.

9.2 Application Testing

All the main functionalities of the mobile application listed in section 7.1 were tested (determining direction and distance, converting information into a simple string, establishing a BLE connection, and sending the message to the Pulse device).

Also, the communication between the app and the physical device has been tested several times.

9.2.1 User interface

The user interface provides information on the connection status of both the device and Google Maps, which were really helpful tools for debugging during development.

Moreover, in previous versions of the app the UI included several more debug elements such as text displaying the current instruction given by Maps, the encoded message sent to the Pulse device through BLE, and other data obtained from the Google Maps notification.

9.2.2 Notification system

To test the connection between the smartphone and the Pulse device, we have added a test button on each side of the screen on the home page ("left" and "right"). The basic functionality of these buttons is to send "l:0", or "r:0", respectively (which are simple directions that can be decoded by the Arduino, as explained in section 7.1). These buttons were useful for finding problems both in the connection part of the mobile app, as well as the Arduino code that decodes the encoded message received from the phone.

9.2.3 Bluetooth

Apart from the test buttons mentioned earlier, for debugging purposes, two-way communication is established between the smartphone and the Pulse device upon connecting. Even though in theory only the phone is required to send information to the Arduino, giving general feedback on the status of the message sent by the smartphone is good practice.

9.3 Integration Testing

9.3.1 System-wide Bluetooth Communication

This phase focused on the connection between the hardware and software components. The main objective was to ensure that the directions extracted by the app were accurately transmitted to the Arduino, which then activated the appropriate vibrator. This involved real-world testing, where routes were set on Google Maps, and the device's haptic feedback was monitored for accuracy.

9.3.2 End-to-End System Testing

The device, in its entirety, was affixed to a bicycle to simulate real-world navigation scenarios. Initial tests were conducted in confined spaces, predominantly within the university campus, navigating turns around buildings, pathways, and open spaces to assess its adaptability in familiar terrains.

However, to ensure comprehensive testing and to simulate various scenarios without physically moving, we utilized a 'Fake GPS Joystick' application. This software allowed us to emulate the bicycle's movement across different terrains and environments on Google Maps. By using this approach, we could simulate conditions such as busy streets, open roads, and places known for potential Bluetooth interference, without physically being present at those locations.

Such virtual simulations, coupled with real-world tests, provided a holistic evaluation of the device's performance. It ensured that our design was not only theoretically sound but practically robust, capable of operating seamlessly in diverse conditions and settings.

10 Design evaluation

Achieving the design goal was challenging, but the team is convinced that the end product is very close to the intended design goal, especially considering the limited scope of the project. The product meets all the "must haves" and some of the "should "and "could haves".

As demonstrated, instructions from the actuators were clearly noticeable due to their powerful output and their placement directly underneath the user's hands. This sub-goal has therefore been met. The app was particularly challenging to develop, but the team was very content with how it turned out. It integrates magnificently with Google Maps, it is easy to establish a Bluetooth connection, the user interface is somewhat user-friendly, but most importantly, it is functional. If it were to be introduced to the market, it would require some visual refinement and a few bugs to be fixed, but the fundamentals are there.

The team has tried to keep the device as small as possible and for now, succeeded at that. The casing is cramped with electronics and cables, but using PCB electronics, it would have been possible to cut down the size significantly more. The battery is well integrated and as proposed, safe and rechargeable. Simple calculations have also proven battery life to be more than sufficient, as it allows for 8 hours of continuous buzzing, which realistically, will never happen. Real-life battery life is therefore more than enough.

Using the current iteration of Pulse on their bicycle, someone can realistically navigate to any destination using just haptics. Pulse could therefore address the safety and inclusivity concerns of traditional bicycle navigation systems. For now, the team's mission has thus been achieved. Nevertheless, Pulse is still far from perfect. In the following sections, areas of improvement will be discussed.

10.1 Waterproofing

For obvious reasons, a consumer-grade iteration of Pulse would have to be waterproof, and should at least have an IP65 rating, meaning that it is dust-tight and can resist light-pressure water spray. Though the device is made of water-resistant plastic, it does not meet this requirement. The casing has several unprotected openings for the actuator cables, the LED (which we covered with transparent tape during testing), the charging port, and the on/off switch. Waterproofing the device would require changing the charging port, using waterproof cable housings (which would also look more aesthetic), and creating protective covers for some openings. Sadly, the team did not manage to develop these waterproof measures, though the device could withstand some rain with the use of tape.

10.2 Easy detachment and attachment

The attachment of the device for the demonstration was rather straightforward, as the diameter of the ring matched the diameter of the bicycle stem. The screw tightened the ring to secure the device. Nonetheless, this means that Pulse is currently not suitable for different stem sizes and shapes. Moreover, it would require a screwdriver to attach and detach the device. On top of that, cables can not be taken off, and the buzzers on the handlebars are intended to be placed underneath the handlebar tape, making it impossible to detach Pulse with ease. To tackle this issue and improve user-friendliness, a combination of a universal mounting system as shown in **??**, a system that requires a mere twisting motion to attach or detach a device, could be used in combination with detachable cables. This would allow Pulse to be detached and to be charged while the buzzers are still mounted on the bike (alternatively, the buzzers can be attached with Velcro and can thus be taken off easily). Lastly, a design with a "base plate" for Pulse to attach on, in combination with rubber bands or tie wraps, would allow Pulse to be mounted securely on different stem types and sizes, increasing versatility.

10.3 Build quality

The build quality of the final product was relatively lacking. The material and the edges are rather rough and tape was used to keep the top and bottom half together and to cover the LED opening. This looks unprofessional and unfinished. It would have been better to use crews or glue to attach components. Glue would have looked the best, but that would have meant that after closing up the device, it would have been impossible to open it up again. Especially in a fast-paced process like this one, this would have hindered troubleshooting and adjustment-making.

Options for the casing design were extensively explored, as shown in Figure 24, but ultimately, it was chosen to stick with something simple. Thus, material choice and aesthetic-wise, there is still ample room for improvement. Improving the design and the refinement of our product would make Pulse more appealing to potential consumers.

11 Individual contributions

11.1 Student 1

11.1.1 Concept phase

The student, specializing in mechanical engineering, started by working out concepts and ideas with the use of sketches and sources. Following that, one final concept was selected with the whole group after the conceptual designs were set aside. This idea was turned into a useable design. An Industrial Designer helped create the first Computer-Aided Design (CAD) by providing the concept sketches. Many adjustments were required to this design. Thus, the design process took a while. Initially, the design was not final because the dimensions were only estimates. This was an effective collaboration between the students because the industrial designer's sketches were developed from the technical knowledge of the mechanical student. For the midterm presentation, the preliminary design was printed. This was also done to give the group an idea of how the case would appear and fit together. After the student of electrical engineering completed the last electrical components, the CAD designs were improved. A sketch of all the locations and dimensions was created after the electrical components were measured.

11.1.2 Final design

The final component casing was designed using this sketch to place the support and holes in the right places. This was taken into consideration because the casing needed to be strong, fitting, and elegant. The final designs were printed. There are made Technical drawings from the final design with dimensions. The final designs were sent to another student so that they could be used for renders and animations for the video.

11.1.3 Presentation and report

The midterm presentation tasks were distributed, with the mechanical student being in charge of the presentation. The presentation was made using the design goals that were established earlier in the project as well as the rubric given by the lecturers. The images from the initial sketches and design were included. The report was broken up into sections, and those who worked on one part of the project also worked on that section in the report. This indicates that the mechanical engineering student wrote the majority of the sections regarding the designing and final designs.

11.2 Student 2

11.2.1 Concept phase

The industrial designer's task was to develop and explore the aesthetics of the casing and how it would be integrated with the hardware components. Initially, he was involved with concept development and ideation, as was the rest of the team. During this stage of the process, he advocated for a more open-minded approach to idea generation, as taught during his studies. During the first weeks, he worked on design explorations, functionalities, several iterations, and visualizations to present the concept to the audience and the team. He did this primarily by creating sketches and design drawings from several perspectives. As the project progressed, collaboration with the mechanical engineer became increasingly more significant. In the end, the final design to continue with and its functionalities were, as chosen by the team, kept rather minimalist, as complexity was limited due to the scope of the project.

11.2.2 Final design

For the final design, the designer passed on his drawings to the mechanical engineer, who made a CAD model from them, suitable for 3D printing. A 3D model intended for design communication with the mechanical engineer was made as well. This was his first time collaborating with dedicated engineers, and therefore a valuable experience to explore what his future role in a design team might be.

11.2.3 Other work

The designer also wrote and recorded video scripts and did report writing, highlighting and validating the user-centered considerations of the product.

11.3 Student 3

11.3.1 Concept phase

The student specializing in computer science naturally took on the task of developing the mobile application that would operate the physical device. Since none of the group members were especially experienced with building Android applications with Java or Kotlin, even with the occasional help of other students this task proposed a great challenge. As a result, he had to start over multiple times and try different approaches to the problem. Both main methods, (extracting navigation data from the Google Maps notification / using an API such as the Google Maps Navigation API) had their strengths and weaknesses, it was difficult to decide which road to take.
Luckily, in the end, he found a feasible solution by converting the direction icon provided by Google Maps into a hexadecimal string that can easily be interpreted by the smartphone application and also the Pulse device. After designing the user interface of the app using Figma, and finding the right Java libraries and tools, he was able to create a functional and reliable mobile application.

11.3.2 Final design

Student 3 played a crucial role in creating the digital tool required to operate the physical device. Upon finishing developing the mobile app, the project had been completed and the cooperation between the different parts of the device was functioning as expected.

11.3.3 Other work

The tasks for the midterm presentation were assigned such that his job was to record and edit the product video and to create a simple mockup of the mobile app that would be developed later on. As for the final presentation, he also recorded and edited the final video, as well as produced many 3D renderings of the device, took part in designing part of the physical device, and contributed to writing the report,

11.4 Student 4

11.4.1 Concept phase

Being specialized in electrical engineering, Student 4 took charge of the core electronics and the connectivity of the device. At the onset, he evaluated the feasibility of the electronic components required for the device, focusing on the navigation and haptic feedback mechanism. A significant part of the process involved selecting the right microcontroller, ensuring it could handle BLE (Bluetooth Low Energy) connections and have sufficient processing power. He also played a crucial role in determining the battery life, and charging mechanism, and ensuring power efficiency for the device. As well as soldering the final prototype.

Simultaneously, as the team lead, he took on the responsibility of ensuring the smooth progression of the project. He laid out a clear timeline, distributing tasks based on each member's expertise and ensuring synchronization between mechanical, design, software, and electrical components.

11.4.2 Final design

Student 4's expertise was pivotal in finalizing the circuit design, ensuring that all components fit within the designated space, and ensuring seamless integration with the mechanical and design components. He also played an instrumental role in ensuring safety features, especially concerning the battery and its charging mechanism.

Besides the technical aspects, he collaborated closely with the mechanical engineer and designer to finalize the CAD models, ensuring that the electronics fit perfectly within the given space. Any modifications required were communicated, and iterative designs were developed until a perfect fit was achieved.

11.4.3 Prototyping and Testing

Using a breadboard, he assembled a prototype of the electronics to test its functionality. This included ensuring stable BLE connections, accurate haptic feedback response, and efficient power consumption. Once the prototype was deemed successful, he transitioned the design to a perfboard, making it more compact and ready for integration into the device.

11.4.4 Other work

Being involved in multiple facets of the project, Student 4 also contributed to report writing, emphasizing the electronic components and their integration. He also took an active role in presentations, ensuring that the technical aspects of the device were communicated effectively. His leadership and electrical engineering skills were pivotal in realizing the vision of the project and ensuring its successful completion.

11.5 Student 5

11.5.1 Concept phase

This student specializing in mathematics helped the computer scientist in developing the application since this student was familiar with Java. This student initially worked on the navigation aspect of the mobile application and tried several methods to successfully read Google Maps notifications. This student unfortunately was not able to successfully able to create a notification reader. The computer science student was able to successfully debug and create an app that read Google Maps notifications. The next issue was the fact that directions are only represented as pictures in Google Maps notifications instead of text. Thus making it harder to convert the notifications into usable information. This student did research on how to convert on how to convert pictures into usable information. Using the knowledge gained from the attempts made previously in making an application that reads Google Maps notifications, this student found a way to convert this information into bitmap and then a hexadecimal string. This information is much easier to work with and is then used to compare already known hexadecimal strings. The computer scientist and electrical engineer then worked together to complete the BLE connection part of the application.

11.5.2 Final Presentation

This student made the slides used in the final presentation using the rubric given on Canvas. This student also helped to film and act in the video which was then edited by a student who was familiar with video editing.

12 Statement about use of AI

In this project, artificial intelligence was used. For instance, while editing the report, OpenAI [12] was used to troubleshoot issues in Overleaf. The bibliography did not work probably in the beginning but with the help of OpenAI, we managed to fix it. The same problem occurred with some usepackages that were not correct. Luckily, OpenAI found the mistakes and fixed them. OpenAI was also used to write Arduino code. Since it provided completely incorrect and non-functional codes, it became evident that it was not suitable for this purpose. OpenAI was also used for the mobile application's code, but this was also unsuccessful. Instead, utilizing several open Github repositories was far more beneficial. To better explain the concept ideas, some sketches were rendered using Vizcom AI [14]. We can attest that no text in the report was written using AI.

13 References

References

[1] *Effects of mobile phone use on bicycling*. 2010. (Accessed on 17/9/2023).

[2] ALPECO. Ip rating reference chart - alternative pest control. https://www.alpeco.co.nz/downloads/IP%20Rating%20Form.pdf, Oktober 2023. (Accessed on 20/10/2023).

[3] Rebecca Karstens Brandt, Sonja Haustein, Marjan Hagenzieker, and Mette Møller. Cyclists' handheld phone use and traffic rule knowledge. *Transportation research part F: traffic psychology and behaviour*, 86:121–130, 2022.

[4] K. Buchholz. Where cyclists are going places. https://www.statista.com/chart/25156/share-using-bike-for-transportation-regularly/, June 2021. (Accessed on 17/9/2023).

[5] Suneela Garg, Chetana Prakash Deshmukh, Meghachandra M Singh, Amod Borle, and Blake S Wilson. Challenges of the deaf and hearing impaired in the masked world of covid-19. *Indian journal of community medicine: official publication of Indian Association of Preventive & Social Medicine*, 46(1):11, 2021.

[6] A. Giesa. Navigating through haptics and sound. https://www.diva-portal.org/smash/get/diva2:1481601/FULLTEXT01.pdf, 2019. (Accessed on 17/9/2023).

[7] Ch Goldenbeld, M Houtenbos, E Ehlers, and D De Waard. The use and risk of portable electronic devices while cycling among different age groups. *Journal of safety research*, 43(1):1–8, 2012.

[8] R. Buecher J. Pucher. Cycling towards a more sustainable transport future. https://www.tandfonline.com/doi/full/10.1080/01441647.2017.1340234, 2017. (Accessed on 13/9/2023).

[9] Forough Jafari. Technology-assisted navigation in public spaces for hard of hearing people. 2019.

[10] NOS Nieuws. oven en slechthorenden minder gezond door gebrekkige medische communicatie. https://nos.nl/artikel/2306256-doven-en-slechthorenden-minder-gezond-door-gebrekkigemedische-communicatie, Oktober 2019. (Accessed on 24/9/2023).

[11] The World Commission on Environment and Development. Brundtland report. https://www.are.admin.ch/are/en/home/media/publications/sustainable-development/brundtland-report.html, 1987. (Accessed on 24/9/2023).

[12] OpenAI. Openai. https://openai.com/chatgpt, Oktober 2023. (Accessed on 5/10/2023).

[13] Aaron Raymond See, Jose Antonio G Choco, and Kohila Chandramohan. Touch, texture and haptic feedback: A review on how we feel the world around us. *Applied Sciences*, 12(9):4686, 2022.

[14] Inc. Vizcom Technologies. Vizom ai. https://www.vizcom.ai/, Oktober 2023. (Accessed on 21/10/2023).

A Appendix A: Code

A.1 Android Application

A.1.1 MainActivity.java

```
package com.example.carplay_android;

import static com.example.carplay_android.javabeans.JavaBeanFilters.*;

import androidx.annotation.RequiresApi;
import androidx.appcompat.app.AppCompatActivity;
import androidx.localbroadcastmanager.content.LocalBroadcastManager;

import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.content.SharedPreferences;
import android.graphics.Color;
import android.net.Uri;
import android.os.Build;
import android.os.Bundle;
import android.os.IBinder;
import android.os.PowerManager;
import android.provider.Settings;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

import com.clj.fastble.data.BleDevice;
import com.example.carplay_android.services.BleService;
import com.example.carplay_android.services.NotificationService;


public class MainActivity extends AppCompatActivity {

    private BleService.BleBinder controlBle;
    private ServiceConnToBLE serviceConnToBLE;

    private Button buttonOpenNotification;
    private Button buttonScanNewDevice;
    private Button buttonConnectToOld;
    private ImageView imageViewBTStatus;
    private ImageView imageViewBleStatus;
    private ImageView imageViewNotificationStatus;
    private ImageView imageViewDeviceStatus;
    private TextView deviceName;

    private Button buttonTestLeft;
    private Button buttonTestRight;

    private BleDevice deviceUsed;

    private Boolean deviceStatus = false;
    public void onReceive(Context context, Intent intent) {
        deviceStatus = intent.getBooleanExtra(getFILTER_DEVICE_STATUS(), false);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        init();

        buttonOpenNotification.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {//open the settings for turn on notification
                startActivity(new Intent(Settings.ACTION_NOTIFICATION_LISTENER_SETTINGS));
```

```
            }
        });

        buttonConnectToOld.setOnClickListener(new View.OnClickListener() {//connect to previous device
            @Override
            public void onClick(View view) {
                if(deviceUsed == null){
                    CharSequence text = "No previous device";
                    int duration = Toast.LENGTH_SHORT;
                    Toast toast = Toast.makeText(getApplicationContext(), text, duration);
                    toast.show();
                }else{
                    controlBle.connectLeDevice(deviceUsed);
                }
            }
        });

        buttonScanNewDevice.setOnClickListener(new View.OnClickListener() {//scan new device
            @Override
            public void onClick(View view) {
                Intent intent = new Intent(getApplicationContext(), BleScanPage.class);
                startActivity(intent);
            }
        });

        buttonTestLeft.setOnClickListener(new View.OnClickListener() {//scan new device
            @Override
            public void onClick(View view) {
                try {
//                    if (deviceStatus) { //idk if this works remove the if statement if it doesnt
                    controlBle.sendDirection("l:0");
//                    }
                } catch (Exception e) {
                    Log.e("error", "Unable to test, connect device" );
                }
                Log.d("LEFT", "LEFT");
            }
        });

        buttonTestRight.setOnClickListener(new View.OnClickListener() {//scan new device
            @Override
            public void onClick(View view) {
                try {
//                    if (deviceStatus) { //idk if this works remove the if statement if it doesnt
                    controlBle.sendDirection("r:0");
//                    }
                } catch (Exception e) {
                    Log.e("error", "Unable to test, connect device" );
                }
                Log.d("RIGHT", "RIGHT");
            }
        });
    }

    private void init(){
        askPermission();
        initComponents();
//        isNotificationServiceRunning();
        initBroadcastReceiver();
        initService();
    }

    private void askPermission(){
        String[] permissions = {
                "android.permission.BLUETOOTH",
                "android.permission.BLUETOOTH_ADMIN",
                "android.permission.ACCESS_FINE_LOCATION",
                "android.permission.ACCESS_COARSE_LOCATION",
        };
        //requestIgnoreBatteryOptimizations();
        requestPermissions(permissions, 200);
```

```
140     }
141
142     private void initBroadcastReceiver(){
143         IntentFilter intentFilter;
144         LocalBroadcastManager localBroadcastManager = LocalBroadcastManager.getInstance(
    getApplicationContext());
145
146         intentFilter = new IntentFilter(getFILTER_DEVICE_USED());
147         ReceiverForDeviceUsed receiverForDeviceUsed = new ReceiverForDeviceUsed();
148         localBroadcastManager.registerReceiver(receiverForDeviceUsed, intentFilter);
149
150         intentFilter = new IntentFilter(getFILTER_BT_STATUS());
151         ReceiverForBTStatus receiverForBTStatus = new ReceiverForBTStatus();
152         localBroadcastManager.registerReceiver(receiverForBTStatus, intentFilter);
153
154         intentFilter = new IntentFilter(getFILTER_BLE_STATUS());
155         ReceiverForBleStatus receiverForBleStatus = new ReceiverForBleStatus();
156         localBroadcastManager.registerReceiver(receiverForBleStatus, intentFilter);
157
158         intentFilter = new IntentFilter(getFILTER_NOTIFICATION_STATUS());
159         ReceiverForNotificationStatus receiverForNotificationStatus = new
    ReceiverForNotificationStatus();
160         localBroadcastManager.registerReceiver(receiverForNotificationStatus, intentFilter);
161
162         intentFilter = new IntentFilter(getFILTER_DEVICE_STATUS());
163         ReceiverForDeviceStatus receiverForDeviceStatus = new ReceiverForDeviceStatus();
164         localBroadcastManager.registerReceiver(receiverForDeviceStatus, intentFilter);
165     }
166
167     private void initService(){
168         serviceConnToBLE = new ServiceConnToBLE();
169         Intent intent = new Intent(this, BleService.class);
170         bindService(intent, serviceConnToBLE, BIND_AUTO_CREATE);
171         startService(intent);//bind the service
172         requestIgnoreBatteryOptimizations();
173     }
174
175     class ServiceConnToBLE implements ServiceConnection {
176         @Override
177         public void onServiceConnected(ComponentName name, IBinder iBinder){
178             controlBle = (BleService.BleBinder)iBinder;
179         }
180         @Override
181         public void onServiceDisconnected(ComponentName name){
182             initService();
183         }
184     }
185
186     @RequiresApi(api = Build.VERSION_CODES.M)
187     public void requestIgnoreBatteryOptimizations() {
188         boolean isIgnored = false;
189         PowerManager powerManager = (PowerManager) getSystemService(Context.POWER_SERVICE);
190         if (powerManager != null) {
191             isIgnored = powerManager.isIgnoringBatteryOptimizations(getPackageName());
192         }
193         if(!isIgnored){
194             try {
195                 Intent intent = new Intent(Settings.
    ACTION_REQUEST_IGNORE_BATTERY_OPTIMIZATIONS);
196                 intent.setData(Uri.parse("package:" + getPackageName()));
197                 startActivity(intent);
198             } catch (Exception e) {
199                 e.printStackTrace();
200             }
201         }
202
203     }
204
205     class ReceiverForBTStatus extends BroadcastReceiver{
206         @Override
207         public void onReceive(Context context, Intent intent) {
208             if(intent.getBooleanExtra(getFILTER_BT_STATUS(),false)){
209                 imageViewBTStatus.setColorFilter(Color.GREEN);
210             }else{
```

```
                imageViewBTStatus.setColorFilter(0x9c9c9c);
            }
        }
    }

    class ReceiverForBleStatus extends BroadcastReceiver{
        @Override
        public void onReceive(Context context, Intent intent) {
            if(intent.getBooleanExtra(getFILTER_BLE_STATUS(),false)){
                imageViewBleStatus.setColorFilter(Color.GREEN);
            }else{
                imageViewBleStatus.setColorFilter(0x9c9c9c);
            }
        }
    }

    class ReceiverForNotificationStatus extends BroadcastReceiver{
        @Override
        public void onReceive(Context context, Intent intent) {
            if(intent.getBooleanExtra(getFILTER_NOTIFICATION_STATUS(),false)){
                imageViewNotificationStatus.setColorFilter(Color.GREEN);
            }else{
                imageViewNotificationStatus.setColorFilter(0x9c9c9c);
            }

        }
    }
    class ReceiverForDeviceStatus extends BroadcastReceiver{
        @Override
        public void onReceive(Context context, Intent intent) {
            if(intent.getBooleanExtra(getFILTER_DEVICE_STATUS(),false)){
                imageViewDeviceStatus.setColorFilter(Color.GREEN);
            }else{
                imageViewDeviceStatus.setColorFilter(0x9c9c9c);
            }
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        unbindService(serviceConnToBLE);
    }
}
```

A.1.2 NotificationService.java

```
package com.example.carplay_android.services;

import static com.example.carplay_android.javabeans.JavaBeanFilters.*;

import android.app.Notification;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.drawable.BitmapDrawable;
import android.graphics.drawable.Drawable;
import android.graphics.drawable.Icon;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.service.notification.NotificationListenerService;
import android.service.notification.StatusBarNotification;
import android.util.Log;

import androidx.annotation.LongDef;
import androidx.localbroadcastmanager.content.LocalBroadcastManager;

import com.clj.fastble.BleManager;
```

```
import com.example.carplay_android.R;
import com.example.carplay_android.utils.BroadcastUtils;
import com.example.carplay_android.utils.DirectionUtils;

import java.io.ByteArrayOutputStream;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.Objects;
import java.util.Timer;
import java.util.TimerTask;


public class NotificationService extends NotificationListenerService {

    private BleService.BleBinder controlBle;
    private ServiceConnToBle serviceConnToBle;
    private Boolean deviceStatus = false;
    private Timer timerSendNotification;
    private Boolean ifSendNotification = false;
    private static String[] informationMessageSentLastTime = new String[7];

    public NotificationService() {}

    @Override
    public void onCreate() {
        super.onCreate();
        init();
    }

    @Override
    public void onNotificationPosted(StatusBarNotification sbn) {
        super.onNotificationPosted(sbn);

        if (isGMapNotification(sbn)) {

            Notification not = sbn.getNotification();

            Bundle bun = not.extras;

            String hex = (String) "";
            try {
                Object largeIcon = bun.get("android.largeIcon");
                Icon icon = (Icon) largeIcon;

                Drawable drawable = icon.loadDrawable(this);
                Bitmap bitmap = ((BitmapDrawable) drawable).getBitmap();

                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                bitmap.compress(Bitmap.CompressFormat.PNG, 100, baos);

                byte[] bitmapBytes = baos.toByteArray();
                Log.i("bitmapBytes", Arrays.toString(bitmapBytes));
                // shorten byeteArray
                MessageDigest md = null;
                try {
                    md = MessageDigest.getInstance("MD5");
                } catch (Exception ignored) { }

                byte[] MD5Bytes = md.digest(bitmapBytes);

                for (byte key : MD5Bytes) {
                    hex = hex + Integer.toHexString(Math.abs(key));
                }
            } catch (Exception ignored) { }


            Log.e("hex", hex);

            String[] beginCodes = {
                   "2be7863432a727174291335104b3120"
            };
            String[] straightCodes = {
                    "770a543365614577b5963224c959"
```

```
            };
            String[] rightCodes = {
                    "124f4c726d254f207476c5b5b177b",
                    "52185e477fa2227526f2d1e1f627a8",   //slight right
                    "120a543365614577b5963224c959"      //sharp right
            };
            String[] leftCodes = {
                    "7a7374491f4242527c297e5464756810",
                    "2c603e264355203f4737327b48b1c5",   //slight left
                    "fae7863432a727174291335104b3120"   //sharp left
            };
            String[] uturnCodes = {
                    "450a543365614577b5963224c959"
            };
            String[] endCodes = {
                    "fc301e264355203f4737327b48b1c5"
            };

            String direction = (String) "x";
            String directionText = (String) "unknown";
            if (Arrays.asList(beginCodes).contains(hex)) { direction = "b"; directionText = "Lets go!"; }
            if (Arrays.asList(straightCodes).contains(hex)) { direction = "s"; directionText = "Continue straight"; }
            if (Arrays.asList(rightCodes).contains(hex)) { direction = "r"; directionText = "Turn right"; }
            if (Arrays.asList(leftCodes).contains(hex)) { direction = "l"; directionText = "Turn left"; }
            if (Arrays.asList(uturnCodes).contains(hex)) { direction = "u"; directionText = "Make a U-turn"; }
            if (Arrays.asList(endCodes).contains(hex)) { direction = "e"; directionText = "You have arrived!"; }


            String distanceString = bun.get("android.title").toString();
            Log.i("distanceString", distanceString);

            Double distance = 0.0;
            try {
                if (!distanceString.split(" ")[1].equals("m")) {
                    distance = Double.valueOf(distanceString.split(" ")[0]) * 1000;
                } else {
                    distance = Double.valueOf(distanceString.split(" ")[0]);
                }
            } catch (Exception e) {
                distance = 0.0  ;
            }

            Log.d("distance", String.valueOf(Math.round(distance)));

            String message = direction + ":" + String.valueOf(Math.round(distance));

            Log.i("message", message);
            Log.d("directionText", directionText);

            if (deviceStatus) {
                controlBle.sendDirection(message);
            }
        }
    }

    @Override
    public void onNotificationRemoved(StatusBarNotification sbn) {
        super.onNotificationRemoved(sbn);
        Log.d("Notification", "removed");
    }

    public static void cleanLastTimeSent() {
        Arrays.fill(informationMessageSentLastTime, "");
    }


    private boolean isGMapNotification(StatusBarNotification sbn) {
        if (!sbn.isOngoing() || !sbn.getPackageName().contains("com.google.android.apps.maps
```

```
    )) {
170             return false;
171         }
172         return (sbn.getId() == 1);
173     }
174 
175 
176     private void init() {
177         Arrays.fill(informationMessageSentLastTime, "");
178 
179         initService();
180         initBroadcastReceiver();
181         setSendNotificationTimer();
182         BroadcastUtils.sendStatus(true, getFILTER_NOTIFICATION_STATUS(),
    getApplicationContext());
183         DirectionUtils.loadSamplesFromAsserts(getApplicationContext());
184     }
185 
186     private void initService() {
187         serviceConnToBle = new ServiceConnToBle();
188         Intent intent = new Intent(this, BleService.class);
189         bindService(intent, serviceConnToBle, BIND_AUTO_CREATE);
190         startService(intent);//bind the service
191     }
192 
193     private void initBroadcastReceiver() {
194         LocalBroadcastManager localBroadcastManager = LocalBroadcastManager.getInstance(
    getApplicationContext());
195         ReceiverForDeviceStatus receiverForDeviceStatus = new ReceiverForDeviceStatus();
196         IntentFilter intentFilterForDeviceStatus = new IntentFilter(getFILTER_DEVICE_STATUS
    ());
197         localBroadcastManager.registerReceiver(receiverForDeviceStatus,
    intentFilterForDeviceStatus);
198     }
199 
200     private class ServiceConnToBle implements ServiceConnection {
201         @Override
202         public void onServiceConnected(ComponentName name, IBinder iBinder) {
203             controlBle = (BleService.BleBinder) iBinder;
204         }
205 
206         @Override
207         public void onServiceDisconnected(ComponentName name) {
208         }
209     }
210 
211     private class ReceiverForDeviceStatus extends BroadcastReceiver {
212         @Override
213         public void onReceive(Context context, Intent intent) {
214             deviceStatus = intent.getBooleanExtra(getFILTER_DEVICE_STATUS(), false);
215         }
216     }
217 
218     public void setSendNotificationTimer() {
219         if (timerSendNotification == null) {
220             timerSendNotification = new Timer();
221             TimerTask timerTask = new TimerTask() {
222                 @Override
223                 public void run() {
224                     ifSendNotification = true;
225                 }
226             };
227             timerSendNotification.schedule(timerTask, 10, 2000);
228         }
229     }
230 
231     @Override
232     public void onDestroy() {
233         super.onDestroy();
234         BroadcastUtils.sendStatus(false, getFILTER_NOTIFICATION_STATUS(),
    getApplicationContext());
235         unbindService(serviceConnToBle);
236     }
237 }
```

A.1.3 BleService.java

```
package com.example.carplay_android.services;

import static com.example.carplay_android.javabeans.JavaBeanFilters.*;

import android.app.Service;
import android.bluetooth.BluetoothGatt;
import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattService;
import android.content.Intent;
import android.os.Binder;
import android.os.Handler;
import android.os.IBinder;
import android.util.Log;

import androidx.annotation.Nullable;


import com.clj.fastble.BleManager;
import com.clj.fastble.callback.BleGattCallback;
import com.clj.fastble.callback.BleMtuChangedCallback;
import com.clj.fastble.callback.BleReadCallback;
import com.clj.fastble.callback.BleWriteCallback;
import com.clj.fastble.data.BleDevice;
import com.clj.fastble.exception.BleException;
import com.example.carplay_android.utils.BroadcastUtils;

import java.util.List;
import java.util.Timer;
import java.util.TimerTask;
import java.util.UUID;


public class BleService extends Service {

    private Timer timerBTState;
    private BleDevice bleDeviceConnectTo;


    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return new BleBinder();
    }

    @Override
    public void onCreate() {
        super.onCreate();
        setBTCheckTimer();
        BroadcastUtils.sendStatus(true, getFILTER_BLE_STATUS(), getApplicationContext());


    }

    public void setBTCheckTimer() {// check the bt state every second
        if (timerBTState == null) {
            timerBTState = new Timer();
            TimerTask timerTask = new TimerTask() {
                @Override
                public void run() {
                    BleManager.getInstance().init(getApplication());
                    boolean status = false;
                    if (BleManager.getInstance().isSupportBle()) {
                        if (!BleManager.getInstance().isBlueEnable()) {
                            BleManager.getInstance().enableBluetooth();
                            //check again see if BT is enabled
                            status = !BleManager.getInstance().isBlueEnable();
                        } else {
                            status = true;
                        }
                    }
                    BroadcastUtils.sendStatus(status, getFILTER_BT_STATUS(),
    getApplicationContext());
```

```
                }
            };
            timerBTState.schedule(timerTask, 10, 1000);
        }
    }

    public class BleBinder extends Binder {
        public BleService getService() {
            return BleService.this;
        }
        public void setMtu(BleDevice bleDevice){
            BleManager.getInstance().setMtu(bleDevice, 200, new BleMtuChangedCallback() {
                @Override
                public void onSetMTUFailure(BleException exception) {
                    Log.d("a", "MTUFailed");
                }

                @Override
                public void onMtuChanged(int mtu) {
                }
            });
        }

        public void connectLeDevice(BleDevice bleDevice) {
            BleManager.getInstance().connect((BleDevice) bleDevice, new BleGattCallback() {
                @Override
                public void onStartConnect() {

                }

                @Override
                public void onConnectFail(BleDevice bleDevice, BleException exception) {
                    Log.d("s", "Connect failed");
                }

                @Override
                public void onConnectSuccess(BleDevice bleDevice, BluetoothGatt gatt, int
    status) {
                    Log.d("s", "Connect success");
                    BroadcastUtils.sendStatus(true, getFILTER_DEVICE_STATUS(),
    getApplicationContext());
                    bleDeviceConnectTo = bleDevice;
                    NotificationService.cleanLastTimeSent();

                }

                @Override
                public void onDisConnected(boolean isActiveDisConnected, BleDevice device,
    BluetoothGatt gatt, int status) {
                    Log.d("s", "Disconnected");
                    BroadcastUtils.sendStatus(false, getFILTER_DEVICE_STATUS(),
    getApplicationContext());
                    connectLeDevice(bleDeviceConnectTo);
                }
            });
        }

        public void sendDestination(String information){
            String DESTINATION_UUID = "beb5483e-36e1-4688-b7f5-ea07361b26a8";
            sendToDevice(information, DESTINATION_UUID);
        }
        public void sendEta(String information){
            String ETA_UUID = "ca83fac2-2438-4d14-a8ae-a01831c0cf0d";
            sendToDevice(information, ETA_UUID);
        }
        public void sendDirection(String information){
            String DIRECTION_UUID = "dfc521a5-ce89-43bd-82a0-28a37f3a2b5a";
            sendToDevice(information, DIRECTION_UUID);
        }
        public void sendDirectionDistances(String information){
            String DIRECTION_UUID = "0343ff39-994e-481b-9136-036dabc02a0b";
            sendToDevice(information, DIRECTION_UUID);
        }
        public void sendEtaInMinutes(String information){
```

```
            String ETA_DISTANCE_UUID = "563c187d-ff17-4a6a-8061-ca9b7b70b2b0";
            sendToDevice(information, ETA_DISTANCE_UUID);
        }
        public void sendDistance(String information){
            String ETA_DISTANCE_UUID = "8bf31540-eb0d-476c-b233-f514678d2afb";
            sendToDevice(information, ETA_DISTANCE_UUID);
        }
        public void sendDirectionPrecise(String information){
            String DIRECTION_PRECISE_UUID = "a602346d-c2bb-4782-8ea7-196a11f85113";
            sendToDevice(information, DIRECTION_PRECISE_UUID);
        }

        private void sendToDevice(String informationMessage, String uuid) {
            String uuid_service = "4fafc201-1fb5-459e-8fcc-c5c9c331914b";
            byte[] data = informationMessage.getBytes();

            BleManager.getInstance().write(
                    bleDeviceConnectTo,
                    uuid_service,
                    uuid,
                    data,
                    false,
                    new BleWriteCallback() {
                        @Override
                        public void onWriteSuccess(int current, int total, byte[] justWrite)
    {
                            Log.d("1", "Success to send");
                        }

                        @Override
                        public void onWriteFailure(BleException exception) {
                            Log.d("1", "Failed to send");
                            new Handler().postDelayed(new Runnable() {
                                @Override
                                public void run() {
                                    sendToDevice(informationMessage, uuid);
                                }
                            },250);
                        }
                    });
        }
    }


    @Override
    public void onDestroy() {
        super.onDestroy();
        BroadcastUtils.sendStatus(false, getFILTER_BLE_STATUS(), getApplicationContext());
    }
}
```

A.2 Embedded Microcontroller

A.2.1 main.cpp

```
#include <Arduino.h>
#include <NimBLEDevice.h>

NimBLEServer *pServer;
NimBLECharacteristic *pCharacteristic;
bool deviceConnected = false;

const int ledPin = 2;
const int rightBuzzerPin = 3;
const int leftBuzzerPin = 4;

#define SERVICE_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define DIRECTION_UUID "dfc521a5-ce89-43bd-82a0-28a37f3a2b5a"

unsigned long lastActionTime = 0;

class ServerCallbacks: public NimBLEServerCallbacks {
    void onConnect(NimBLEServer* pServer) {
        deviceConnected = true;
```

```
        digitalWrite(ledPin, HIGH);
    };

    void onDisconnect(NimBLEServer* pServer) {
        deviceConnected = false;
        digitalWrite(ledPin, LOW);
    }
};

class MyCallbacks : public NimBLECharacteristicCallbacks {
    void onWrite(NimBLECharacteristic *pCharacteristic) {
        unsigned long currentTime = millis();
        if (currentTime - lastActionTime < 10500) {
            return;
        }
        lastActionTime = currentTime;

        std::string value = pCharacteristic->getValue();
        char direction = value[0];
        int distance = std::stoi(value.substr(2));

        int buzzDuration = 1000; // default 1 second buzz
        if (distance > 0 && distance <= 50) {
            buzzDuration = 500; // half-second buzz for very close distance
        } else if (distance > 50 && distance <= 100) {
            buzzDuration = 1000; // 1 second buzz for mid-distance
        } else if (distance > 100) {
            buzzDuration = 2000; // 2 second buzz for far distance
        }

        switch (direction) {
            case 'b':
                digitalWrite(leftBuzzerPin, HIGH);
                digitalWrite(rightBuzzerPin, HIGH);
                delay(buzzDuration);
                digitalWrite(leftBuzzerPin, LOW);
                digitalWrite(rightBuzzerPin, LOW);
                break;
            case 's':
                delay(buzzDuration);
                break;
            case 'r':
                if (distance == 0) {
                    // Special case for r:0
                    buzzDuration = 10000; // 10 second buzz
                }
                digitalWrite(rightBuzzerPin, HIGH);
                delay(buzzDuration);
                digitalWrite(rightBuzzerPin, LOW);
                break;
            case 'l':
                if (distance == 0) {
                    // Special case for r:0
                    buzzDuration = 10000; // 10 second buzz
                }
                digitalWrite(leftBuzzerPin, HIGH);
                delay(buzzDuration);
                digitalWrite(leftBuzzerPin, LOW);
                break;
            case 'u':
                digitalWrite(leftBuzzerPin, HIGH);
                digitalWrite(rightBuzzerPin, HIGH);
                delay(buzzDuration);
                digitalWrite(leftBuzzerPin, LOW);
                digitalWrite(rightBuzzerPin, LOW);
                delay(500);
                digitalWrite(leftBuzzerPin, HIGH);
                digitalWrite(rightBuzzerPin, HIGH);
                delay(buzzDuration);
                digitalWrite(leftBuzzerPin, LOW);
                digitalWrite(rightBuzzerPin, LOW);
                break;
            case 'e':
                digitalWrite(leftBuzzerPin, HIGH);
```

```
                delay(buzzDuration);
                digitalWrite(leftBuzzerPin, LOW);
                delay(500);
                digitalWrite(rightBuzzerPin, HIGH);
                delay(buzzDuration);
                digitalWrite(rightBuzzerPin, LOW);
                break;
            default:
                break;
        }
    }
};

void setup() {
    pinMode(ledPin, OUTPUT);
    pinMode(leftBuzzerPin, OUTPUT);
    pinMode(rightBuzzerPin, OUTPUT);

    NimBLEDevice::init("ESP32_BT");
    pServer = NimBLEDevice::createServer();
    pServer->setCallbacks(new ServerCallbacks());

    NimBLEService *pService = pServer->createService(SERVICE_UUID);
    pCharacteristic = pService->createCharacteristic(
                        DIRECTION_UUID,
                        NIMBLE_PROPERTY::READ |
                        NIMBLE_PROPERTY::WRITE |
                        NIMBLE_PROPERTY::NOTIFY
                    );

    pCharacteristic->setCallbacks(new MyCallbacks());
    pService->start();
    pServer->getAdvertising()->start();
}

void loop() {
    if (!deviceConnected) {
        digitalWrite(ledPin, !digitalRead(ledPin));
        delay(500);
    } else {
        delay(10);
    }
}
```

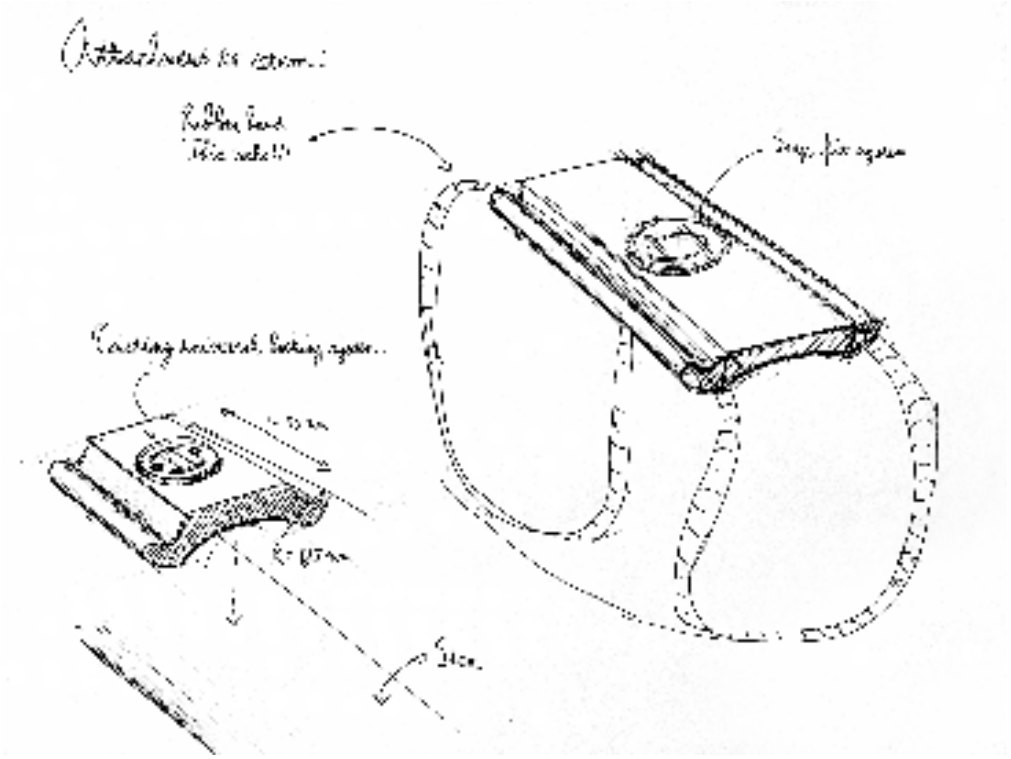


Figure 23: Universal mounting system

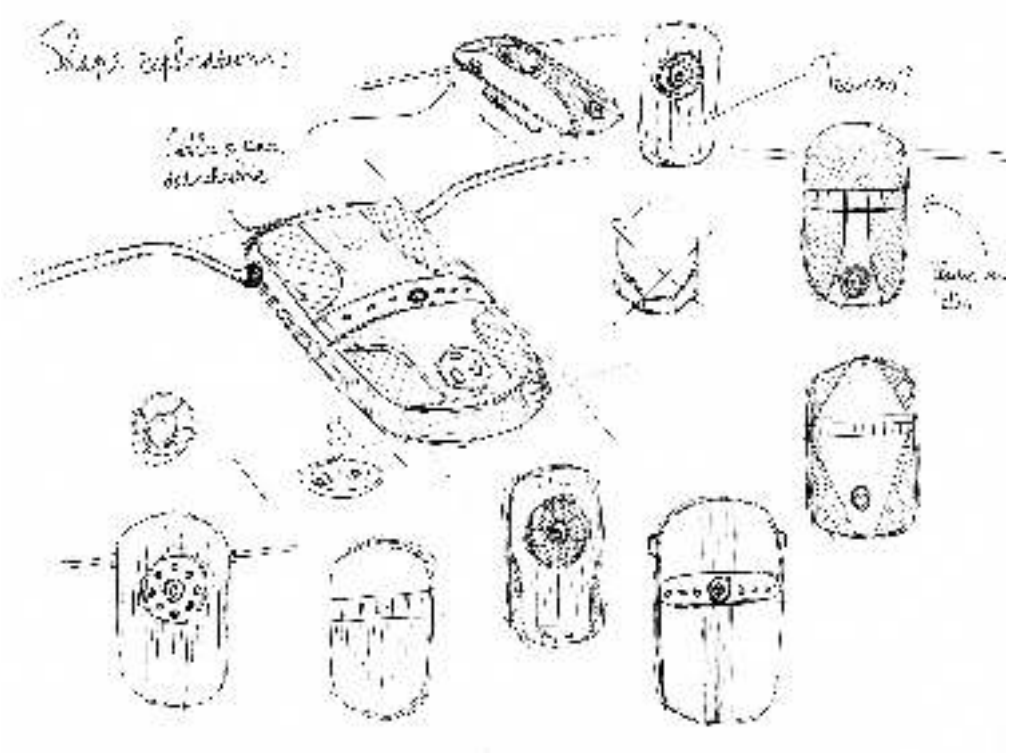


Figure 24: Shape explorations

Figure 25: The device installed on a racing bicycle

Figure 26: Rendering of the inside of the device