GuideMe, a “smart” white cane approach

LAURENZ BAUMGART
University of California – San Diego, laurenz.baumgart@hotmail.de

ANOOP GUNAWARDHENA
University of California – San Diego, agunawar@ucsd.edu

RICK TRUONG
University of California – San Diego, rickvtruong@gmail.com

ALBERT ZHONG
University of California – San Diego, azhong@ucsd.edu

In this paper, we introduce GuideMe, a “smart” white cane that improves upon the typical white cane, making it safer for the visually impaired to cross a crosswalk. Our system comprises two primary devices: Raspberry Pi with sound sensors attached to the white cane and an android app with google maps API and a voice interface. With the combined capabilities, the system interacts over speech input and output, detects nearby crosswalks, determines whether cars are approaching, and detects if the user is walking onto the street. These features would allow visually impaired users to make more confident and informed decisions on when they should cross the street, ultimately improving their safety, independence, and quality of life. From testing, we see great potential in our device and avenues to improve it for more widespread use.

CCS CONCEPTS • Hardware • Software and its engineering • Mathematics of Computing

Additional Keywords and Phrases: Visually impaired people, navigation, directional microphones, GPS, Raspberry Pi, Google Maps API, geofencing, crosswalks, Google Assistant, publish-subscribe servers

1 INTRODUCTION

Crossing a crosswalk is inherently a very visual task. To better illustrate this, we can break down the process of crossing a street into its subtasks. One would first have to detect and identify a nearby crosswalk, check for approaching cars, and maintain alignment while crossing to ensure they do not walk into the street. This process is simple for people with sight, but what for the visually impaired?

Crossing a crosswalk for visually impaired people is much more complex and dangerous [1]. They must rely on good infrastructures to identify crosswalks, such as tactile paving and sound feedback, their ears to listen to approaching cars, and their human compass to walk straight. However, sometimes the required infrastructure is not good enough or does not exist. Furthermore, external noise sources could disturb the visually impaired, but also multiple other factors, for example, weather and street layout, can influence the noise reflection and thus make it harder to distinguish those sources and listen for approaching cars [2].

This is where GuidMe comes in. GuideMe's goal is to help visually impaired people to cross streets more safely by extending the human ear even in challenging environments. The system extends a simple white cane with a sound detector and provides a mobile application.

The sound detector can distinguish multiple sound sources, directions, and locations and project them onto a 3D environment. In the 3D environment, moving objects can be distinguished from stationary objects, and other objects, like planes, can be filtered out. The mobile application provides a voice interface that uses Google Assistant to interact with the visually impaired person back and forth. Furthermore, the application retrieves the current user location and knows about crosswalks nearby, programmed into the application, with the help of geofences. Together, these features would allow visually impaired users to make more confident and informed decisions on when they should cross the street, ultimately improving their safety and quality of life.

1.1 Structure of the Paper

In this paper, we start off by going into greater detail on our motivations and the problem space, such as by detailing previous works and their shortcomings. Next, we will elaborate on our design idea by reviewing its different iterations. This leads us to the most lengthy and essential part of our paper, which describes the overall system of our device, such as its architecture, the integrated technologies, and the implemented features and how they were implemented. After that, we will go over what tests we performed and our evaluation of these tests, then the collaborative efforts of our team to bring this device to life. Finally, we will conclude with a final retrospective of our device and what can be done in the future to improve it.

2 MOTIVATION AND BACKGROUND

Our primary motivation for the implemented prototype is to help visually impaired people to cross crosswalks more safely and independently. One of our team members once saw a visually impaired person having trouble crossing a street and helped this person to cross. In this specific case, too much external noise disturbed the person making it impossible to distinguish them and filter out approaching objects. This motivated him to develop a system that removes this problem. As one can imagine, the ability to not walk safely and independently can be devastating and reduce our target group's quality of life.

We found that crossing a street creates a considerable obstacle for visually impaired people. According to this source [1], locating of a crosswalk is by far the most challenging part, even if there are audio systems attached to traffic lights. In our case, crosswalks usually do not have audio systems, making crossing even harder. Furthermore, unlike traffic lights,

crosswalks do not show car drivers a signal indicating if a person wants to cross. Those two problems are exactly what we want to tackle with our prototype and help the visually impaired to cross crosswalks more safely. Furthermore, we were able to conduct an interview with a visually impaired person, and he confirmed what we found during our research (A.1). According to him, the right timing on when to cross the street and check whether it is safe to cross seems to be the biggest problem.

Although it's common in many countries worldwide, there is still no good standard for tactile paving [3]. There are even documented cases where the paving leads people into obstacles such as trees or ledges, making it outright dangerous instead of helpful [4]. All of this decreases the confidence of the visually impaired to travel alone. It is essential for the visually impaired to have good guidance when navigating around because over 60% of blind people travel more than five times a week outside their own residences. On top of that, almost half of blind travelers travel an unfamiliar route which requires systems to make them feel comfortable and safe [12]. All this research and our interview motivated us to create our prototype that should serve as a basis for further development and eventually bring a system to market that benefits the visually impaired.

2.1 Related Work

With the development of the white cane in the 1930s, the first steps towards more independent navigation for the visually impaired were made [5]. Over the last 70 years, researchers have been trying to implement electronic-aided systems to help with navigation for the visually impaired [6]. The first sensor systems were developed around the 1960s to detect nearby objects with the help of ultrasound sensors and radar because this technology was on the rise during and after World War 2 [7]. Since then, many researchers have been trying to improve those systems and try to reduce their errors.

Multiple applications were developed between 2010 and 2014, but we could not find similar applications that were developed after 2020. Most of these applications were deprecated, or the last updates were made years ago. A list of related GPS applications can be found in this source [8]. One application that comes close to our purpose is BlindSquare. However, this application helps with orientation around known places but does not approach the problem of crossing streets. But as mentioned before, more than half the travels outside of a visually impaired person's home happen in unknown environments.

A Korean researcher made another aid for blind people to aid in crossing traffic crosswalks. It works by scanning for a pedestrian signal light and determining when it shows the walk signal. The device succeeded in having a 96.2% color interpreting accuracy. However, overall, it only managed to get a 74.4% overall accuracy, showing room for improvement [9]. Furthermore, this system is also not helping with crossing crosswalks, as they do not indicate when it is safe to cross.

Stanford researchers created an improved white cane version with sensors attached to it that allow it to "self-navigate" [10]. It works by using similar sensors to the ones used in self-driving vehicles to detect when obstacles are in front of the user and adjusts itself accordingly via a spinning wheel mechanism. When the wheel spins, the user feels a "tug" that tells them how to move to avoid the obstacle. It succeeded in a way that with 5 minutes of training, blind people increased their walking speed by 38%. Finally, this article outlines many apps made with the visually impaired in mind [11].

To name a few systems above, there is still a need for a system that helps users cross crosswalks that are not marked by a guided audio system. The Stanford study doesn't help in this regard, as it only improves what the white cane can already do in detecting obstacles right in front of it. The Korean study, while similar, is narrower in scope in that it only works with crosswalks with a pedestrian signal, thus limiting the crosswalks it can support. Finally, the discontinued apps showcase the need for further improvement in helping the visually impaired. These sentiments were also echoed by a

visually impaired person whom we interviewed. As far as he knows, he is unaware of any successful commercial product out there that could help him cross crosswalks more safely.

3 DESIGN

The three features we want to implement with our prototype should inform the user about…

1. …nearby crosswalks
2. …approaching cars
3. …about when he exits to the crosswalk onto the street

To develop our suggested features, our prototype must be able to fetch a current user's location, required for features 1 and 3, interact through a voice interface, needed in all features, and detect approaching cars with sensors, required in feature 2. In the following, we will elaborate on our ideas and decisions on how to get and provide the required information.

3.1 Prototype Requirements

When developing a system that is put onto a white cane, there are several requirements that we think are essential to consider

- **System start-up time**
  - The system should be ready to use in under one minute when picking up the white cane and starting to navigate. Otherwise, it is another obstacle than actual help.
- **Usability**
  - A voice interface should provide feedback and transcribe user input to make it as seamless as possible.
- **Cost**
  - The system cost should be less than a white cane which costs between $15 and $50.
- **Weight**
  - A slim white cane can weigh as little as 7 ounces. Therefore, the attached sensors should not weigh more than the white cane, or it might make it harder to use.
- **Energy consumption**
  - Our system will for sure consume energy, and we want to keep the energy consumption as low as possible. Otherwise, a user would have to carry multiple external chargers.

3.2 Fetch the Current User Location

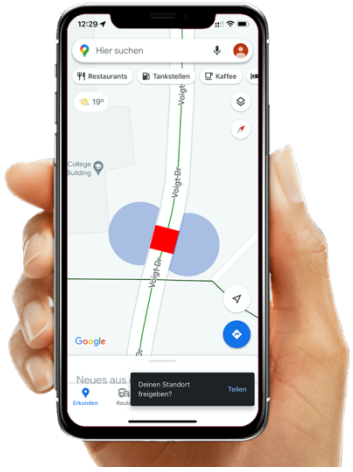


Figure 1: Geofence around a nearby crosswalk

This part of the prototype was clear from the beginning. With the help of an android application and the GPS sensor in an android device, we want to fetch the current user location. Nowadays, every android phone supports GPS, and this sensor is available through the Google Maps API. As seen in figure 1, the blue semicircles mark the geofence of a nearby crosswalk. The red area denotes the actual crosswalk. As soon as the user enters the blue semicircle, the system can notify them about it. If the user exits the red bar, the system knows that the user left the crosswalk onto the street. After some research, we found that Google also provides a geofencing client that is compatible with the Google Maps API. This service helps to implement features 1 and 3.

3.3 Interact Through Voice Interface

For the voice interface, we had multiple solutions in our minds. Our voice interface needs to be actively invoked, and it needs to listen to user input, transcribe the input to text and perform actions, like sending a message to a server, based on what the user said.

The first idea was to create a Voiceflow application and deploy it to Amazon Alexa. As soon as the user is near a crosswalk, we want the voice assistant to speak it out loud and ask the user if he wants to cross. However, Alexa is a speaker performing actions based on user activation. In our case, the user does not know that he is around a crosswalk and will not actively talk to Alexa, which is unsuitable for our purpose. Furthermore, the hardware weighs around 12 ounces which is more than double the weight of a white cane and is therefore too big and too heavy.

That is why we had to look for another voice interface and found Google Assistant to be a perfect match for our purpose. We found that it can be actively invoked to output text which is precisely what we need. Furthermore, the assistant can listen to voice input and transcribes it into text. With Google Assistant already being available on our android device, we can also reduce the hardware size of our system and make it easier to use.

3.4 Detect Approaching Cars

First, we decided to implement a time window in which the PI senses its environment for surrounding cars. This not only helps to reduce the system's energy consumption but also makes it more precise. If the sensors were always active, they

could detect approaching obstacles, although the user does not want to cross anything. This helps to have clear communication and prevent confusion.

The initial idea to detect approaching cars was to implement it with the introduced ultrasound sensors from the lecture. But the available ultrasound sensors are limited in range and can only detect nearby obstacles about 2 meters away. Our system needs to detect cars further away from the crosswalk in a range of around 10 to 30 meters.

After discussions with Chen and our Professor, we found that directional microphones could be a great alternative and decided to use them. We even found directional microphones compatible with a raspberry PI, allowing us to quickly work with the extracted signals. The combined devices weigh less than 70 grams and fulfill our described weight requirements.

3.5 Inter-Device Communication

Another essential part of our prototype is to enable communication between devices. First, this is necessary to activate the PI if the user wants to cross the crosswalk. After the PI finishes listening for approaching cars and analyzing its environment, it sends a message back to the android device. First, we wanted to develop a standard TCP connection but got recommended to implement it via a publish-subscribe service. We found a service provider called Pubnub that allows easy integration of its service into our architecture. The service can be found here: https://www.pubnub.com/. Our PI and Android devices subscribe to our channel and listen to messages. When one device publishes a message, the receiving device reacts to those messages.

3.6 System composition on an improvised white cane

Over time, these initial designs proved too cumbersome in some aspects, leading to many design changes. For example, the Amazon Echo proved entirely unnecessary, and our final product had only two devices attached to the cane: a raspberry pi and an android phone (as can be seen in figure 3 below)/ Our current architecture and dataflow are listed in the Architecture section in Section 4: System Development.

Figure 3: Our final functional prototype of GuideMe. Attached are a raspberry pi with directional microphones and an android device.

Figure 4: Close-up of the raspberry pi + microphone sensor

4 SYSTEM DEVELOPMENT

Developing GuideMe took the joint effort of our entire team to write over three thousand lines of code in total, using multiple devices, APIs, and sensors. The android app was fully developed with Java in android studio, and the raspberry pi was coded in Python. In this section, we outline the architecture of our system, going into detail on the networking and I/O, our technologies used, such as our two devices, and finally, our four main features of our system.

4.1 Architecture

Our final dataflows are as follows:

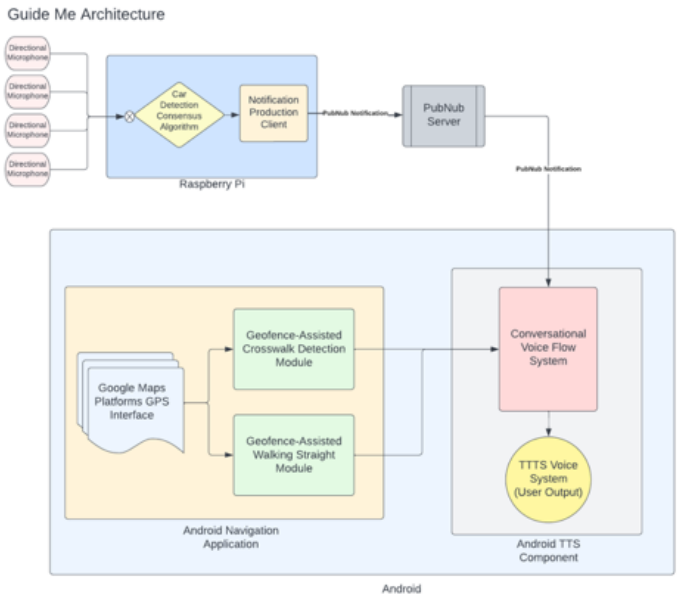


Figure 5: A high-level diagram of GuideMe System Architecture and its core components

Crosswalk Detection Dataflow

Initially, the Google Maps Platform Interface module will, through the Google Maps API, collect user location data and street layout data for the user.

The user location and street layout data will be directed to the Geofence-Assisted Crosswalk Detection Module, which will use Google's Geofence API to classify whether the user is approaching a crosswalk and send the data internally to the Android TTS Component, specifically to the Conversational Voice Flow System module.

Once the Conversational Voice Flow System module receives data from the Geofence-Assisted Crosswalk Detection Module that there should be a notification raised to the user, the Conversational Voice Flow System module will send a signal to the TTS Voice System which will alert the user of an approaching crosswalk.

Detect incoming cars/safety to walk Dataflow

Four Directional Microphone Sensors will initially collect automobile velocity and proximity information about the user when crossing a street.

The automobile velocity and proximity information will be directed to our Raspberry Pi, where it will run its Car Detection Consensus Algorithm on the data collected by our Directional Microphone Sensors to accumulate the different velocity and proximity values across all Directional Microphone Sensors.

The Car Detection Consensus Algorithm module will send its data to the Notification Production Client module, which will classify whether there exist one or more automobiles of pressing velocity and proximity.

The Notification Production Client will then send its classification data to a PubNub Server through a secured PubNub channel to forward to the Android TTS Component, specifically to the Conversational Voice Flow System module.

Once the Conversational Voice Flow System module receives data from the Raspberry Pi signifying that there should be an alert raised to the user, the Conversational Voice Flow System will send a signal to its adjacent TTS Voice System, which will notify the user of a moving object.

Walking Straight Dataflow

Initially, the Google Maps Platform Interface module will, through the Google Maps API, collect user location data and street layout data for the user.

The user location data and street layout data will be directed to the Geofence-Assisted Walking Straight Module, which will use Google's Geofence API to determine whether the user is deviating from a crosswalk the user is crossing and send the data internally to the Android TTS Component, specifically to the Conversational Voice Flow System module.

Once the Conversational Voice Flow System module receives data from the Geofence-Assisted Walking Straight Module that there should be a notification raised to the user, the Conversational Voice Flow System module will send a signal to the TTS Voice System which will alert the user that they are deviating unsafely from a crosswalk.

For our networking and I/O applications in GuideMe, we used a PubNub server for inter-device communication. We deployed one PubNub server with two PubNub channels on our Android TTS Component and one PubNub client on Raspberry Pi and Android Navigation Application to allow communication between each component. As detailed on the PubNub website, PubNub communication works as follows,

"The PubNub Data Stream Network believes in a protocol-independent open mobile web, meaning that we will use the best protocol to get connectivity through any environment. PubNub has used a variety of protocols over time, like WebSockets, MQTT, COMET, BOSH, SPDY, long polling, and others, and we are exploring architectures using HTTP 2.0 and others. The bottom line is that PubNub will work in every network environment, and has very low network bandwidth overhead, as well as low battery drain on mobile devices."

Data sent from one module to the other is temporary and not persistent (except that stated in Google Maps and PubNub's Privacy and Terms of Service agreements), so no database is needed. As soon as GuideMe stops running, user data will be deleted.

We are not planning to process or store individual information for the current system. However, for user data sent between components, PubNub ensures the confidentiality, integrity, and authenticity of user data sent within PubNub channels.

4.2 Technology Used

We have three major components (getting noise data using directed microphones for determining if cars are approaching, getting location data through google maps API to set up geofencing, and finally, getting user input through a speech recognizer and generating audio output through text-to-speech). These components are spread amongst our two devices, a raspberry pi for the noise data and an android device that runs an app that handles google maps API, user input, and audio output. To allow these two devices to communicate, we use a Pubnub server.

4.2.1Raspberry Pi

We used a Raspberry Pi for car detection. This was done using a ReSpeaker 4-Mic Array and ODAS Studio (both below). When triggered, the Raspberry Pi's expected functionality is to run ODAS Studio, which would be configured to activate the microphones and collect data from them. The data prepared by ODAS (in JSON form) is to be parsed and then used

upon further numerical analysis to determine the exact source direction of audio energy inferred from the data. Then the PI calculates the total magnitude of acoustic energy that is felt by it for each direction and also determines the rate of change(power) of these energy sources, the volume of the authorities and their power are then used to determine if a source is dangerous or not.

4.2.1.1 Respeaker 4-Mic Array

This device was chosen because it had equidistant microphones, all on the corners of its frame. This was very attractive to us as it gave us the advantage of multidirectional analysis and made localization of sound a much smoother process as relative audio energy could be used to calibrate direction.

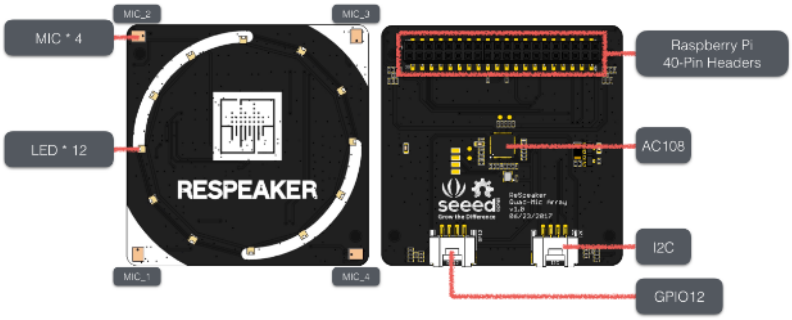


Figure 6: Labelled Image of the Respeaker 4-Mic Array, via Seeed Studio wiki (https://wiki.seeedstudio.com/ReSpeaker_4_Mic_Array_for_Raspberry_Pi/)

4.2.1.2 ODAS Studio

ODAS Studio was chosen because it is a powerful audio energy analysis and offers compatibility for multi-microphone systems similar to ours. ODAS Studio works by analyzing the input from all the microphones and then using their relative distance from each other to localize sources of potential sound. Its settings can also be tweaked to sample less or more samples of incoming energy to offer varying degrees of localization. However, the localization data returned by ODAS only makes sense relative to the microphones, as seen below. Therefore, further analysis was required to determine if any specific energy reading inferred an object to the left, or the right, and more importantly, if the object was approaching. Odas Repo for reference

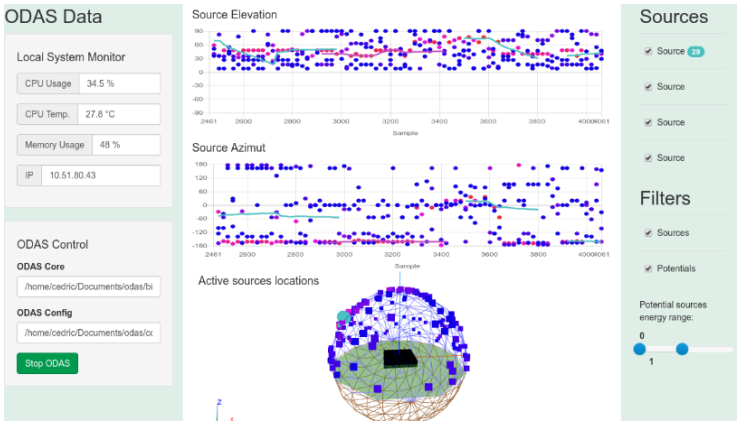


Figure 7: Screenshot of our ODAS Studio data

4.2.2Android Application

Our Android Application has two main sections, with one unit handling parts relating to user location and another handling the more interactive features for the user, namely the audio outputs that alert the user and user inputs.

4.2.2.1 Android Navigation Application

The navigational component of our Android application, the Android Navigation Application, contained the Crosswalk Detection and Walking Straight feature of GuideMe. Google Maps Platform was crucial in both features' development. Within Google Maps Platform lies a suite of open-source APIs that developers, ranging from enterprise to freelance developers, can use to develop navigational-based applications. The Google Map Platform APIs which were particularly important for the development of GuideMe were the Maps SDK for Android and Geolocation API.

For GuideMe's Android Navigation Application UI, the Maps SDK for Android was responsible for rendering the map seen upon application launch. All streets, roads, buildings, natural landscapes, and other geographical features rendered on the application are made visible through this SDK. User location data is also tracked, and our user's location is made visible through a series of API calls by this SDK.

For GuideMe's Crosswalk Detection feature, the Maps SDK for Android was coupled with the Geolocation API to generate the voice notification produced when a user approaches a crosswalk. This is made possible through a series of implementation details listed in section 4.3.1.

For GuideMe's Walking Straight feature, the Maps SDK for Android was coupled with the Geolocation API to generate the voice notification produced when a user walks across and deviates from a crosswalk. This is made pos through a series of implementation details listed in section 4.3.4.

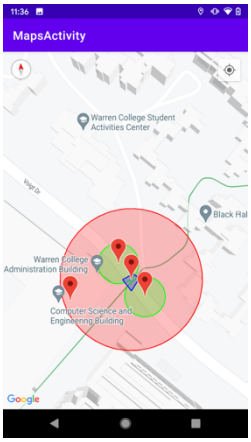


Figure 8: The above screenshot depicts both types of GuideMe Geofences. The region encompassed by the red region is a Crosswalk Detection Geofence. When Google Maps Platform detects a user entering the region encompassed in red, a GeofenceTransition event will be triggered and a message sent to the Android TTS Component for eventual voice output "There is a crosswalk detected, do you want to cross?". When a user leaves the region encompassed in red, a GeofenceTransition event will be triggered and a message will be sent to the system (not outputted) notifying the user of the event for debugging purposes.

4.2.2.2 Android Text to Speech

For the more interactive sections, we needed to integrate two main libraries, namely android TextToSpeech and android SpeechRecognizer, into our code. However, due to a change in the way android handles permissions past android SDK 30, this required changes to our android manifest file. From there, we had to implement a text-to-speech object and its methods, such as onInit. Due to the complexity of doing so, we created our own additional packages to make our code more scalable in the future if needed. Thus, with the text-to-speech setup, our app could output spoken messages to the user. Furthermore, after following similar practices for our speech recognizer, we could also allow the device to perform speech recognition using input from the user's microphone.

4.2.3Pubnub Server

Our Pubnub server is what is responsible for handling all of our device-to-device communication. Once connected, the subscribe key allows a device to receive messages, and the publish key allows it to send them. Thus, this allows for each device to send out messages to alert one of our other devices to act, and for each device to receive messages to act on. For example, in the car detection feature of our system, when the raspberry pi detects that there are no longer any cars and that it is safe to cross, it will publish the "safe to cross" message in our channel. From there, when our android application, which is subscribed to the channel, receives the message it can act on that message to tell the user that it is safe to cross, as well as activate our walking straight feature.

4.3 Features

4.3.1Crosswalk Detection

As mentioned in the Technology Used section, GuideMe's Crosswalk Detection feature makes use of the Maps SDK for Android and Geolocation API in order to generate the voice notification produced when a user approaches a crosswalk. Initially, the Maps SDK for Android keeps track of our user's current location data after getting the proper permissions to access the user's location data for his/her android. Once the user consents to allowing Google Maps Platform to collect his/her location data under the Google Privacy and Terms of Service agreement, the Maps SDK will generate an instance of a My Location object on the map which will refresh the user's location data (longitude, latitude, address, time, location error-bound, etc.) periodically.

Subsequently, the Geolocation API enabled the creation and monitoring of Geofences on the application's map. According to Google's Geolocation API documentation,

> "Geofencing combines awareness of the user's current location with awareness of the user's proximity to locations that may be of interest."

With the Geolocation API, we were able to set a geofence at any latitude and longitude with a specified trigger radius. Our current implementation of GuideMe's Crosswalk Detection feature consists of creating a geofence at coordinates (32.8818788, -117.2331303), right near the crosswalk at the UCSD CSE Building, with a geofence trigger radius of 35 m. This geofence functionality encoded two geofence transitions, which are defined under the Geolocation API. The first geofence transition in this geofence is the Geofence.GEOFENCE_TRANSITION_ENTER event, which is triggered upon user entrance of the area encompassed within the geofence. Upon Geofence.GEOFENCE_TRANSITION_ENTER, an Android TextToSpeech ObservableObject instance will update its message field to store a "There is a crosswalk detected, do you want to cross?" String which will be processed and outputted by the Android TTS Component. The second of geofence transitions encoded in this geofence is Geofence.GEOFENCE_TRANSITION_EXIT. Upon Geofence.GEOFENCE_TRANSITION_EXIT, a few messages specifying the trigger of that event will be Toast-ed to the UI and the Android console for debugging purposes.

4.3.2Alerting the User & User Input

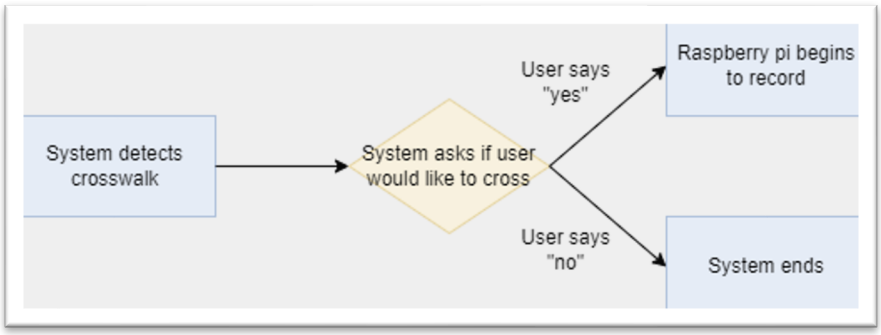


Figure 9: Example flowchart of how our system works with user input

As mentioned in the previous section on our Android Application, users can interact with our device based on outputs from our app. The outputs we have outputted through text-to-speech are as follows:

- "crosswalk detected, would you like to cross?" - As mentioned earlier and as seen in figure [#], this message is sent out when the location part of our app detects the user entering a geofenced area. Once it is received, our text to speech will output this message to the user, and then activate our speech recognizer, which listens for if the user says either "yes" or "no". If the user says "yes", the app will send a "record" message to the raspberry pi to tell it to turn on its sensors to begin listening for cars. If the user says "no", our system will output "understood" to inform the user that it heard the user's intent, and then end.
- "there are still incoming cars, please wait" - This message is sent out if the raspberry pi sends out a "wait" message when it detects incoming cars during the listening timeframe (further elaborated on in the next section on car detection). This, in turn, will have our text-to-speech output this message.
- "there are no cars detected, it is safe to cross now" - This message is sent out if the raspberry pi sends out a "safe to cross" message when it detects incoming cars during the listening timeframe (further elaborated on in the next section on car detection). This, in turn, will have our text-to-speech output this message.
- "you are not walking straight, please return to the crosswalk for your safety" - This message is sent out when the location part of our app detects that the user is exiting the geofenced area around the crosswalk (further elaborated on in a future section on walking straight).

Thus, this feature audibly alerts the user when each other feature is being activated to make our system more intuitive and user-friendly. This gives the user full transparency of what is happening behind the scenes so that they understand precisely what is happening at any given time, making our system more intuitive. Furthermore, this also makes our system more user-friendly as it allows the user to communicate their intents with the system in case they don't currently need some of its features (such as if they don't actually plan to cross the detected street, then there is no need to activate the car detection feature).

4.3.3Car Detection

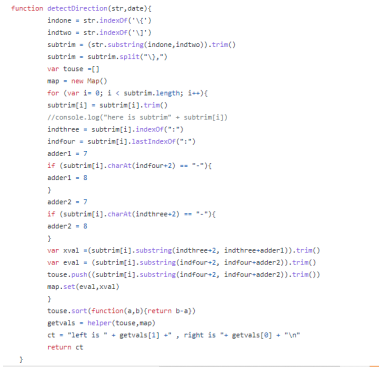
As inferred in the previous section, the Raspberry Pi and its tools are used for car detection.

- Upon receiving affirmation from the user, the Pi launches ODAS Studio to start analyzing incoming audio energy, however as previously mentioned the localization done by ODAS Studio is relative to the source, therefore each spike in audio energy recorded from the surroundings is assigned a coordinate:


```
"timeStamp": 1884,
"src": [
    { "id": 30, "tag": "dynamic", "x": 0.763, "y": 0.296, "z": 0.574, "activity": 0.000 },
    { "id": 0, "tag": "", "x": 0.000, "y": 0.000, "z": 0.000, "activity": 0.000 },
    { "id": 66, "tag": "dynamic", "x": -0.223, "y": 0.379, "z": 0.898, "activity": 0.000 },
    { "id": 67, "tag": "dynamic", "x": -0.174, "y": -0.923, "z": 0.332, "activity": 0.882 }
]
```


Figure 10: Our Screenshot of Data from ODAS Studio

- So after running ODAS Studio for 10 seconds and piping the data in JSON form into a file, the Pi then parses through this file and mathematically processes the o-ordinates to determine if the energy spikes are to the left or right of the Pi. (of concern when crossing the road). After this is done it divides the spikes into the either a right or a left source.



```
function detectDirection(str,date){
        indone = str.indexOf('\{')
        indtwo = str.indexOf('\}')
        subtrim = (str.substring(indone,indtwo)).trim()
        subtrim = subtrim.split("\},")
        var touse =[]
        map = new Map()
        for (var i= 0; i < subtrim.length; i++){
        subtrim[i] = subtrim[i].trim()
        //console.log("here is subtrim" + subtrim[i])
        indthree = subtrim[i].indexOf(":")
        indfour = subtrim[i].lastIndexOf(":")
        adder1 = 7
        if (subtrim[i].charAt(indfour+2) == "-"){
        adder1 = 8
        }
        adder2 = 7
        if (subtrim[i].charAt(indthree+2) == "-"){
        adder2 = 8
        }
        var xval =(subtrim[i].substring(indthree+2, indthree+adder1)).trim()
        var eval = (subtrim[i].substring(indfour+2, indfour+adder2)).trim()
        touse.push({subtrim[i].substring(indfour+2, indfour+adder2)).trim())
        map.set(eval,xval)
        }
        touse.sort(function(a,b){return b-a})
        getvals = helper(touse,map)
        ct = "left is " + getvals[1] +" , right is "+ getvals[0] + "\n"
        return ct
    }
```

Figure 11: The screenshot of our code for determining the sound direction

- After the data has been parsed and its direction determined, the Pi then has to compute if the source(i.e left or right) is dangerous or not, it does this by analyzing both the average magnitude of the energy spikes for each source and also each sources rate of change of energy(changing power), after comparing these values to thresholds, the Pi concludes if the source is dangerous.We consider the power of a source to be equally important to the magnitude of its energy because power implies acceleration(explained below) which if present indicate that the source is moving rather than stationary, which matches the profile of a moving car.

```
def helper(left, right, leftdiff, rightdiff):
    leftot = 0.0
    righttot = 0.0
    leftacc = 0.0
    rightacc = 0.0
    for i in range(len(left)):
        leftot = leftot+ left[i]
       # print("left is " + str(left[i]))
        righttot = righttot + right[i]
        leftacc = leftdiff[i] + leftacc
        rightacc = rightdiff[i] + rightacc
    rightavg = righttot/float(len(right))
    #rightaccavg = rightacc/float(len(rightdiff))
    #leftaccavg = leftacc/float(len(leftdiff))
    leftavg = leftot/float(len(left))
    #print("leftot is " + str(leftot))
    #print("leftavg is" + str(leftavg))
    #print("rightavg is" + str(rightavg))
    #print("leftaccavg is" + str(leftaccavg))
    #print("rightaccavg  is" + str(rightaccavg))
    if(rightavg >= 0.5 or leftavg >= 0.5):
        if rightavg >= 0.5 :
            if rightacc >= 0.0:
                print("danger")
                return
        if leftavg >= 0.5:
            if leftacc >= 0.00006:
                print("danger")
                return
    print("safe to cross")
```

Figure 12: Our Screenshot of our code for determining if it's a car approaching

- the changing power of a source can be related with its acceleration through the formula below, which infers the two are proportional. Thus, we decided to analyze the changing power of a source because that implies the acceleration of the source (provided its mass stays the same). And if a source has an increasing power rate, it is

accelerating toward the microphone. If its amplitude (Magnitude of energy) is high enough, this qualifies it to be a dangerous vehicle.

$$a = \sqrt{\frac{P}{2mt}}$$

Figure 13: Formula for what was described above

- Once the Raspberry Pi has concluded that there is a potentially dangerous source of sound nearby, it publishes a PubNub message to the channel indicating danger and then restarts the process all over. However, if no danger is detected, then the Pi publishes a PubNub message indicating that it is safe. All PubNub messages received from the Pi will then be read to the User by Phone using the aforementioned text-to-speech service.

4.3.4Walking Straight

As mentioned in the Technology Used section, GuideMe's Walking feature uses the Maps SDK for Android and Geolocation API to generate the voice notification produced when a user walks across a crosswalk and deviates unsafely from the crosswalk.

Once again, the Maps SDK for Android keeps track of our user's current location data after getting the proper permissions to access the user's location data for his/her android.

GuideMe's Walking Straight feature was implemented using Google's Geolocation API in a different fashion than that of GuideMe's Crosswalk Detection feature. Our current implementation of GuideMe's Walking Straight feature consists of creating two geofences on top of and below the crosswalk near the UCSD CSE Building at coordinates (32.8819543, -117.2331900) and (32.8817980, -117.2330663), respectively. Instead of a geofence trigger radius of 35 m as was implemented in GuideMe's Crosswalk Detection geofence, both Walking Straight geofences have a trigger radius of 20 m as we decided the bounds of the side of the crosswalk were narrower than the radius we felt necessary to encode for crosswalk detection. These geofences both encoded two geofence transitions: Geofence.GEOFENCE_TRANSITION_ENTER and Geofence.GEOFENCE_TRANSITION_EXIT. The first geofence transition in this geofence is the Geofence.GEOFENCE_TRANSITION_ENTER event. It is triggered upon user entrance of the area encompassed within the geofence. Upon Geofence.GEOFENCE_TRANSITION_ENTER, an Android TextToSpeech ObservableObject instance, will update its message field to store a "not walking straight detected" String which will be processed and outputted by the Android TTS Component. The second geofence transition encoded in this geofence is Geofence.GEOFENCE_TRANSITION_EXIT. Upon Geofence.GEOFENCE_TRANSITION_EXIT, a few messages specifying the trigger of that event will be Toast-ed to the UI and the Android console for debugging purposes.

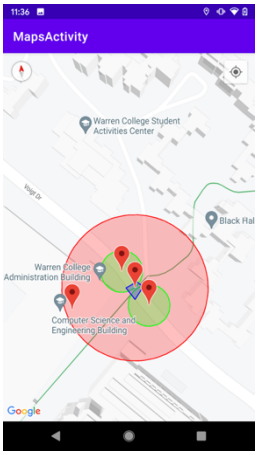


Figure 14: The above screenshot depicts both types of GuideMe Geofences. The region encompassed by the green region is a Walking Straight Geofence. When Google Maps Platform detects a user entering the region encompassed in green, a GeofenceTransition event will be triggered and a message sent to the Android TTS Component for eventual voice output "not walking straight detected". When a user leaves the region encompassed in green, a GeofenceTransition event will be triggered and a message will be sent to the system (not outputted) notifying the user of the event for debugging purposes.

5 TESTING AND EVALUATION

GuideMe required constant testing to ensure its functionality week by week. Furthermore, the final stretch required hundreds of man-hours to iron out all the kinks that arose during testing. However, at the end of all the testing, we were rewarded with a fully functional system that was modular and simple to use and addresses all of our initial goals.

5.1 Testing

Testing first began with testing each individual feature independently. In fact, even the two parts on the android application, the location and the user input/audio output parts, were done on their own separate apps at first. Both apps were tested on a pixel 4 android virtual device running SDK 30. Testing the location application required running it on the virtual device to see if the google maps API was loading, a feat that was much easier said than done. Testing the user input and audio output required creating an extra button that would either turn on the microphone or have our text to speech to say a custom phrase. However, while this worked fine for text to speech, testing the microphone on the virtual device required multiple extra steps in the settings of the device, and even then it was inconsistent on whether or not it will work. After exhaustive testing with both an actual android device and the virtual device, we concluded that this inconsistency with whether or not it will work was an unavoidable complication with using the virtual device. Testing the raspberry pi involved playing noise near the microphones to see what the data output was like, and then testing our algorithm from there. After testing with speaking near the pi, clapping near the pi, and playing a car noise near the pi, we were able to get the pi to accurately detect when a dangerous object is approaching, namely a car.

Once we were able to have each app and the raspberry pi's systems working, we began to work on having each part communicate with each other by integrating with our Pubnub server. This required a custom built Python script in order to publish any inputted message we had to simulate the communication between the apps and the pi. We also shifted the

custom button's functionality in the audio output app to instead publish a message to Pubnub, and for the app to read whatever messages it sees in the subscribed channel.

Once everything was integrated with Pubnub, we were able to start with the most extensive and lengthy phase of testing: full integration. The first step in doing so was merging our two android apps into one, cleaning up the code in the process. Initial testing of this merged app was done by manually setting certain locations for a test user to ensure that merging the two apps didn't break any functionality. Once that was confirmed, we were ready to begin field testing at the CSE building crosswalk. At first, we only tested the android application since the crosswalk detection is what kicks off our entire system. However, we found that there were some issues with the location accuracy, requiring a large geofence in order for our program to work. To try to figure out the issue with the location accuracy, we first compared our application's shown location with the shown location of another phone running google maps side by side as we walked along the building we were testing at. From there, we were able to tell that our application was slower at updating the user's location. To make sure that this wasn't simply due to hardware reasons, we then ran google maps on our android device to compare with the other phone. In that test, the shown user location updated at roughly the same time each time, showing that it was indeed an issue with google maps API on our application. Upon further research, we determined that this was likely due to the type of API key we were using for our application being for free use. With that in mind, we were then able to begin testing our system with the raspberry pi included. To do so, we went to the crosswalk multiple times and allowed our full system flow (crosswalk detection, then user input, then car detection, then walking straight) to run its course. However, these tests had troubling results: many of our systems were failing for no explicable reason. The crosswalk detection was inconsistent, and there were a few trials in which the raspberry pi even crashed. The root cause of these issues turned out to be that the area we were testing was a dead zone, and thus with poor gps data and internet connectivity, our systems were unable to perform their tasks. Upon testing with a strong 5G hotspot, all of our issues vanished and our tests had everything running smoothly the way they should.

5.2 Evaluation

When we began, the main problem we set out to solve was providing a tool for the blind or visually impaired in cities that would help them cross streets more confidently and safely. Evaluation of our work showed that our fully functional system was perfectly able to achieve this goal. Our system, for the most part, runs flawlessly. Our crosswalk detection feature is able to run smoothly, with the only hitches being due to the fact that our GPS location accuracy isn't the greatest (which in turn is due to the free version of google maps API not being as fast to update). Since the location isn't particularly fast to update, we have been forced to create large geofencing areas. This in turn leads to the occasional incident in which our system detects the crosswalk too early due to the large area. Even more rarely is when even the large area isn't enough due to this lag in updating the location, leading to it being late. While this issue has been found to be minimized with the use of a strong hotspot, it still has to be kept in mind. The text to speech also outputs anything it needs to in a timely and informative manner, and the speech recognizer is able to process the main commands the user needs (namely stating their crossing intent). Furthermore, the raspberry pi sensors are able to reliably pick up the noise from cars and distinguish it from other potential sources of noise such as foot traffic. While it is not completely foolproof with its current design, the conditions of something being able to trigger a false positive are exceedingly rare. Finally, the walking straight feature is also fully functional, albeit with similar issues as the crosswalk detection due to the not fully reliable google maps API.

Our system is not only functional, but easy to use as well. While creating our system, we made sure to design our system to be user friendly and intuitive. To that end, we believe that we were successful in this endeavor. For example, our system requires minimal setup by the user: all they need is the turn on the tool itself. In fact, this is especially true since the android

application itself can run in the background. From there, our system activates automatically upon detection of a crosswalk, and always communicates what it is doing to the user so that they know what is going on. Furthermore, since our tool builds off an already existing tool our target audience should be familiar with, the white cane, without changing the way it is used, our target users would have a minimal learning curve with our device. Finally, since our system is not reliant on the presence of certain external markers (unlike the Korean Crosswalk study being reliant on crosswalk signs [8]), our tool allows for full independence and confidence when using our device. Since the user doesn't have to rely on infrastructure that is not always readily available, such as guiding sound corridors, they are now much more free in where they can go safely and confidently.

While we may have a fully functional and easy to use product already, we also made sure to make it modular and extendible for future improvements. We can easily add more speakers and sensors to our device, such as a lidar sensor if we want to improve our car detection. In fact, we can even integrate more devices if needed due to the fact that our Pubnub server can support multi-channel communication. In that way, the only hardware limitation is simply how much space we have on the cane itself before it becomes too unwieldy. On the software side of things, our android application allows for adding more features and patches to our navigation component, and can even be ported to wear OS for even more ease of use or to make more space for other devices on the cane. This is because our code implementation follows software engineering best practices to be more scalable. As mentioned earlier, we made our own custom implemented packages that can be more easily called by as many features as needed. While doing so required a lot of up front effort on our end, the payoff in the future would be well worth it. Thus, in that way, our implementation is both testable and maintainable for the foreseeable future, as well as ethical.

6 COLLABORATION

6.1 Structure of the Team

One of our undergraduates on the team, Albert, was in charge of handling everything involving voice input and output for the user. This was due to him having the least coding knowledge due to his background in design instead of computer science, and since our device required little to no user interface designing due to its intended audience being the visually impaired, the least technical task was delegated to him instead. During the first week of progress, we used voiceflow in order to set up Amazon Alexa to be able to communicate with the user. However, by the end of that week it was found that our original plan for the voice interface utilizing an echo dot wouldn't suit our needs due to its inability to handle push notifications as well as the clunkiness of trying to integrate an echo dot onto our cane. This led us to instead swap to using a google assistant as this would allow our voice interface to be done on the same device as our location related work. This also came with its own separate set of issues, namely the fact that google actions had a long deploy time making it difficult to test. It was also discovered in a later week that there were issues with integrating Pubnub with voiceflow as it required a paid version of voiceflow as well as additional expertise. Furthermore, since our voice interface was made to be greatly simplified over those weeks, we decided to shift to using google text to speech and voice recognition starting with our penultimate sprint. Although there were issue with the text to speech engine not initializing at first, we were able to get it resolved by the final sprint in time for the full integration. From there, cleaning up the code to make it more scalable and integrate it with the location app was done by Laurenz, our graduate student in the team who was also our scrum master.

Anoop was the Pi developer, his responsibilities included, setting up the microphones on the pi, integrating in ODAS studio, localizing sources of sound, determining if a source of sound was dangerous or not, and communicating output messages to the PubNub channel.Weeks 5 and 6 saw him explore many different localization strategies with the

microphones before settling on ODAS Studio before the second project report. However during week 7 we realized that the localization provided by ODAS alone was not optimal for our purpose, so part of week 7 was spent parsing the json data prepared by ODAS and organizing them into left and right categories. Towards the third project meeting, around week 8 is when we started testing sound with the pi. This included ambient noise found at a crosswalk (birds, jets, people, talking, and walking), vehicles approaching the crosswalk and vehicles halted at the crosswalk waiting for pedestrians to cross. However towards the end of week 8 integration became more of a priority for all developers rather than complete functionality. So efforts on testing the pi were halted and directed towards integrating the pi with the phone, ensuring that PubNub supported communication worked seamlessly, this was what was presented for the penultimate project report at thanksgiving. For the last two weeks of development Anoops time was divided between finishing his exploration into energy levels, building a mathematical model to determine if a source was dangerous, implementing PubNub responses and testing the system to see if it works as a whole. By week 9 Anoop had derived a 2 fold magnitude and acceleration based comparison model to detect dangerous sources and had established thresholds that were reliable to a great extent. Week 10 and finals week was spent testing to see if said model and thresholds were accurate by testing the Pi at the crosswalk, then testing the entire system with all devices to see if it worked as planned.

Rick was GuideMe's principal Android developer. His responsibilities included determining which navigation services were suitable for GuideMe's development at low costs, deciding Google Maps Platform was the most suitable navigational service for GuideMe's development and budget, registering a Google Maps Platform account to access Google Maps Platform APIs, developing the Android Navigation Application UI, programming GuideMe to ask users for consent to use their location data, implementing GuideMe's Crosswalk Detection feature, and implementing GuideMe's walking straight feature. In weeks 5 and 6, Rick spent time researching what navigational services were offered which suited GuideMe's needs and one which was preferably offered for free as GuideMe's team budget was informally determined to be relatively low. Google Maps, Waze, and a few other navigation platforms are compatible with Android development, but the one which seemed to have the most documentation and history of Android application development was the Google Maps SDK for Android. After concluding that Google Maps Platform best met GuideMe's developmental needs, Rick registered a Google Maps Platform account with a free 90-day trial to access Google Maps Platform APIs used in the Android Navigation Application. In Week 7, Rick set up a Google Maps Activity Android project which imported Google's Maps SDK for Android built-in UI, which now acts as GuideMe's Android Navigation Application UI. He also implemented asking users for consent on their location data through the Google Maps SDK for Android API which enabled the instantiation of a My Location instance on the application's map, depicted as a blue circle on the application UI. An issue that Rick had run into attempting to complete the aforementioned tasks was proper set-up and configuration of the Android Google Maps Activity project with his Google Maps Platform account credentials and API keys. He resolved the issue through enabling all API offered by the Google Maps Platforms on his API key and referenced that in the proper location in AndroidManifest.xml. Throughout Week 8 and Week 9, Rick developed GuideMe's Crosswalk Detection feature. Rick spent most of Week 8 following the Geolocation documentation and integrating it with the project's existing Maps SDK for Android code. This consisted of researching how to create a GeofencingClient instance, creating a basic geofence with no geofence transition events encoded at a specific (longitude, latitude) and geofence trigger radius, and learning Pending Intents which allowed for the GeofencingClient to communicate with the Geolocation API. There were numerous issues involved in this process which Rick had to resolve which included learning all the aforementioned implementations details. In Week 9 and 10, Rick spent the entirety of Week 9 researching how to encode geofence transitions in geofences in order to finish implementing Crosswalk Detection and in Week 10, finish implementing Walking Straight as both features relied on geofences exhibiting different geofence behavior upon trigger. Throughout the latter half

of Week 10 up until the first half of Week 11, Rick worked with Laurenz and Albert to help integrate GuideMe's Android Navigation Application with GuideMe's Android TTS Component which finished his technical project responsibilities. Additional project responsibilities included helping organize the product demo and helping write the Project Final Report.

Laurenz worked as the Scrum Master for the entire team. At first, Laurenz helped to further develop the team's initial idea. Here it was crucial that the prototype could be developed within the given time frame and resources. Furthermore, he helped with narrowing the target group of the developed prototype, keep it specific and actionable. After that, Laurenz developed the critical milestones for the prototype development to keep track of the team's progress and make it measurable. During the development of the prototype, Laurenz's main responsibilities were to coordinate team meetings, organize sprints and report the current status of the project to our supervisors. Due to his experience as an android developer, Laurenz was also helping out with the integration of the developed applications, the server infrastructure, the text to speech feature and speech recognition. Several dependency issue arose during the integration wich he helped fixing in order to make the application work.

6.2 Problems/Issues and How They Have Been Solved

6.2.1GPS Inaccuracy From Powerbank

When testing the prototype, we found that that the GPS accuracy of the phone was affected by the presence of the powerbank used to power up the raspberry pi on the cane. Since the phone and the cane had to be in somewhat close proximity to each other, this affected most of our test runs. We speculate that the powerbank was a source of electro-magnetic interference that affected the accuracy of the phones GPS, we were eventually able to overcome this problem by extending the cable connecting the Pi and its powerbank and then putting the powerbank in the testers pocket as opposed to being on the cane.

6.2.2Microphone Systematic Error

The Microphone showed to have a systematic error when sampling, no matter the amount of noise in the surroundings the microphone would always record high levels of audio energy for a period of roughly 10ms whenever it started recording. This skewed a lot of data points when our Pi developer was exploring the ranges of audio energy for thresholds when designing the algorithm. The solution was to exclude all data gathered in the first 10ms of sampling, and this change was made to our algorithm.

7 CONCLUSION AND FUTURE WORK

7.1 Improved Reliability

7.1.1Vehicle Detection

When expanding the capability of vehicle detection something important to consider is electric vehicles, our microphones were unable to distinguish noise made by electric vehicle engines from other sources with a degree of certainty. Now, this is a flaw as there are an increasing number of electric vehicles on the road today. However, my research into alternative methods of vehicle detection show that infra-red detection would categorize sources based on their energy output which would be better for identifying cars in general than sound. (Lopez 2015).

7.1.2Crosswalk Generalizability

Our Prototype was designed for a specific crosswalk, however in order to make the product commercially viable, more crosswalks have to be added, this can be done with satellite imagery and using a powerful geo-location tool similar to the Google Maps API, to dynamically recognize crosswalks as the user is traveling. This is possible as it requires very little implementation on top of what we have already done.

7.2 Make the system ready to attach to a white cane

Our presented prototype can already be attached to a white cane, including a power bank to keep it running. However, the system needs further development to prepare it for commercial use. Our hardware currently includes multiple sensors and ports that aren't used, like the HDMI port, the USB ports, and the LAN port. Those are currently not used by our system. Furthermore, the Repseaker mic-array includes LED lights that are not used. Those LED lights require quite a lot of space and move the microphones apart. If we remove those lights, the system width could be less than a half inch. This would allow reducing the size and weight of our hardware even further. The idea is to have a microcontroller that is run by a small battery, and our algorithm for car detection is flashed to the operating system.

7.3 Closing Thoughts

With GuideMe, we successfully created a tool that helps the blind or visually impaired more safely and confidently cross a street. Since our tool doesn't rely on any external factors, such as crossing lights, our system is fully independent. It thus gets around the issues found in our research, namely problems with our current infrastructure not being fully accommodating. This makes it so that the user doesn't have to rely on audio information at crosswalks, for example, as they can be more self-sufficient with the use of GuideMe. Furthermore, our system has been designed to be as easy to use as possible for our target audience. Using the white cane as a base means the user doesn't have to learn to use any new tool, mainly because the added functionality to the cane requires minimal setup. Most things are done automatically, and those that aren't are activated based on the user's spoken intent, such as crosswalk detection activating if and only if the user says they would like to cross.

Thus, the question becomes: where do we go from here? The most obvious improvement is to improve our GPS location accuracy. Currently, the user location can be slow to update due to inaccuracies with google maps API, and to get around it, we were forced to create large geofencing areas in our system. While these two factors often cancel out, this occasionally leads to times when the crosswalk detection feature is a little early. If we can find a way to improve our user location, such as finding a faster version of the google maps API, we can have a more reliable crosswalk detection and walking straight feature. Another avenue of exploration would be to expand this android application to be compatible with Wear OS. This would make our application even more accessible and convenient to use compared to a phone due to the better portability of a watch.

Furthermore, it can improve the design of the cane, as it no longer has to hold a bulky phone on it, freeing up space for other devices or sensors. For example, a lidar sensor can be attached to the raspberry pi to improve the accuracy of our car detection feature. On the other hand, instead of adding more sensors, we can enhance the sleekness of the cane by shrinking the size of our microphone sensor.

Finally, one avenue we can continually improve is scaling the system. Currently, our system is manually set for demo purposes for one specific test crosswalk. Since it works flawlessly for that crosswalk, it should be able to work for basically any crosswalk. However, how would we do that? To start, we can continue to hard code and test locations around the

UCSD campus to see how it works at that scale. From there, we can expand throughout San Diego to reach a wider audience. Yet, doing so manually would require much tedious manual labor. Thus, a route to explore would be to develop an algorithm to mark crosswalks automatically. This could be in the form of a machine learning algorithm that can scan google maps API and learn the distinctive features that show a crosswalk on the map.

All of these potential changes would be doable with our current team and resources in mind. However, what if we had more time, expertise, resources, and workforce? GuideMe's development achieved many things, and what we achieved was done with four students, three undergraduates, and one graduate, in five weeks. Much of what we did was outside the realm of expertise for most of us, especially for one who is a student in design, not computer science. Despite all of this, we were able to create a robust tool utilizing geofencing for crosswalk detection and walking straight and directional sensors for detecting cars at a particularly given crosswalk. Thus, with more time, expertise, resources, and workforce, the possibilities for expansion become endless. Improving location accuracy and integrating a lidar scanner would become child's play, and the scale issue can go beyond much beyond just one city. It is easy to imagine that with enough backing, GuideMe can become a widely available patented and commercialized tool for blind people everywhere to be able to use. Our vision for GuideMe was always to create a helpful tool for the blind/visually impaired community to travel more independently and safely. Thus, allowing GuideMe to reach as many hands as possible to improve the quality of life of the visually impaired would be the ultimate goal of our team.

REFERENCES

[1] Carole Martinez. 2022. Blind and Visually Impaired Pedestrians: What Are Their Difficulties When Crossing the Street? https://www.inclusivecitymaker.com/blind-visually-impaired-pedestrians-difficulties-crossing/

[2] Ken Thompson. 2022. Can I Cross the Street? Considerations for a Blind Pedestrian – An Interview with Dona Sauerburger, COM https://www.nadtc.org/news/blog/can-i-cross-the-street-considerations-for-a-blind-pedestrian/

[3] J. Lu, K. W. M. Siu and P. Xu. 2012. A comparative study of tactile paving design standards in different countries. 2008 9th International Conference on Computer-Aided Industrial Design and Conceptual Design, 2008, pp. 753-758, doi: 10.1109/CAIDCD.2008.4730674. https://ieeexplore.ieee.org/document/4730674

[4] Kurt Kohlstedt. 2017. Death by Tactile Paving: China's Precarious Paths for the Visually Impaired. https://99percentinvisible.org/article/death-tactile-paving-chinas-precarious-paths-visually-impaired/

[5] Braille Institute. 2022. White Cane Safety Day. https://brailleinstitute.org/white-cane-day#:~:text=White%20canes%20were%20introduced%20in,United%20States%20since%20that%20time

[6] Real S, Araujo A. Navigation Systems for the Blind and Visually Impaired: Past Work, Challenges, and Open Problems. Sensors (Basel). 2019 Aug 2;19(15):3404. doi: 10.3390/s19153404. PMID: 31382536; PMCID: PMC6696419. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6696419/

[7] National Research Council (US) Working Group on Mobility Aids for the Visually Impaired and Blind. Electronic Travel AIDS: New Directions for Research. Washington (DC): National Academies Press (US); 1986. Chapter 6, THE TECHNOLOGY OF ELECTRONIC TRAVEL AIDS. Available from: https://www.ncbi.nlm.nih.gov/books/NBK218025/

[8] Wikipedia contributors. 2022. GPS for the visually impaired. https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=GPS_for_the_visually_impaired&id=1116294719&wpFormIdentifier=titleform#AMA_style

[9] Song, Hanjun & Wachirawit, Ponghiran. 2014. Pedestrian crossing aid device for the visually impaired. https://www.researchgate.net/publication/309173663_Pedestrian_crossing_aid_device_for_the_visually_impaired

[10] Andrew Myers. 2021. Stanford Researchers Build $400 Self-Navigating Smart Cane. https://hai.stanford.edu/news/stanford-researchers-build-400-self-navigating-smart-cane

[11] The vision of children foundation. 2018. Smartphone Apps for the Blind and Visually Impaired. https://www.visionofchildren.org/voc-blog/2018/7/13/smartphone-apps-for-the-blind-and-visually-impaired

[12] Manduchi, R., & Kurniawan, S.H. 2010. Mobility-Related Accidents Experienced by People with Visual Impairment. https://www.semanticscholar.org/paper/Mobility-Related-Accidents-Experienced-by-People-Manduchi-Kurniawan/1cd941a033ec0e6d2390228eebe0ba8c6f9e978a

A APPENDICES

A.1 Interview with Gordon (a visually impaired student that has some visualization left and is not 100% blind)

What are the most significant obstacles you encounter when walking and navigating independently?

The biggest obstacles are streets without sidewalks, and complex intersections such as the ones with separate straight and turning green lights. Another obstacle is when there are branches overhead or on the ground

What are the current external devices that you use for independent mobility?

I use my folding cane, iPhone, Airpods, Google Maps app, One Bus Away app, and Go MTS apps.

Do you think that you need help with crossing a street? If yes, what are your biggest problems/fears?

My biggest problem is when there are complex intersections such as the one described above or when there are not enough cars on the street. This is because I use the cars to know when to cross the street. I am independent on most street crossings.

Are you able to identify crosswalks?

Yes, based on my usable vision

If not, would identifying crosswalks help you?

N/A

What steps are necessary for you to cross a crosswalk?

I have to listen for the parallel surge of cars which means cars that are going straight away from me or straight towards me. Another method is to listen for the perpendicular cars to stop crossing.

Would information about the length of the crosswalk help you?

Maybe, but the timing of when to cross is more important

Would a system that identifies if you walk straight to the other side help you?

Not really, but it might help other people.

Would a system that identifies approaching cars help you?

Maybe

Would a system that helps you align you vertically to the street help you?

Not really, but it might help other people

If an external system gives you feedback, do you prefer voice over haptic feedback, like vibration?

I think audio but it could be good to have options for both

Can you think of a solution that would help you to walk and navigate more safely and independently?

I think if there were glasses with cameras that could detect traffic lights and tell you when to cross, that would be good. They have glasses that connect to cameras so you can make video calls, but I'm talking about Ai recognizing traffic lights.

Do you have any other important information about mobility for the visually impaired?

The biggest thing that helps visually impaired people, for now, would be if there were more accessible crosswalks. That means crosswalks that either tell you when to cross or start beeping when you should cross. Some cities are better than others in terms of how many accessible crosswalks there are.