A Wearable Navigation System for Augmenting Guide Dog Object Detection

by

Tim Zhong

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2019

Author .....
Department of Electrical Engineering and Computer Science
May 24, 2019

Certified by .....
Daniela L. Rus
Professor
Thesis Supervisor

Certified by .....
Igor Gilitschenski
PhD
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

A Wearable Navigation System for Augmenting Guide Dog Object Detection

by

Tim Zhong

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2019, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, I discuss the design, construction and evaluation of a wearable object detection system for blind users to avoid collisions. Blind people rely upon tools such as guide dogs to navigate the world around them. However, even guide dogs are prone to errors that leave their owners prone to unexpected collisions with objects, such as low-hanging branches and street signs, that may cause bodily harm. This work introduces a wearable device that provides object detection capabilities to blind people with guide dogs. The device contains a camera, an embedded computer and a vibration motor to provide feedback when an obstacle is detected. The system uses computer vision techniques to determine whether there are objects in the path of the user, as well as the distance of said objects. A series of experiments was conducted with guide dog users traversing a path of obstacles while using the device to determine the effectiveness of the device at allowing users to avoid collisions.

Thesis Supervisor: Daniela L. Rus
Title: Professor

Thesis Supervisor: Igor Gilitschenski
Title: PhD

Acknowledgments

I would like to thank my advisors Daniela Rus and Igor Gilitschenski, who both consistently helped provide direction/guidance for my work and helped me revise my thesis, as well as helping me keep the gradient. I'd like to also thank Robert Katzschmann, whose previous work in served as an inspiration for the topic of this thesis. He worked closely with me throughout the first half of my MEng and provided plenty of advice, guidance and technical expertise. I'd also like to thank the many members of the MIT CSAIL Distributed Robotics Lab (DRL) Group, all of whom I've greatly enjoyed getting to know during my time working with them.

Finally, I would like to thank the wonderful people at Guiding Eyes for the Blind for providing the support for this work and the inspiration for such an important problem. Their expertise with guide dog navigation and assistance with the project was invaluable, and through this collaboration I learned a lot about the challenges that visually impaired people face. I now have a better understanding of those challenges, and better understand the importance of technological inclusion and innovation for people with disabilities. Special thanks to Benjamin Cawley and Thomas Panek for their close involvement and guidance in the project, and for their help in designing and conducting experiments. Spending time with them in Boston and at the Guiding Eyes campus has been very influential on the trajectory of this research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We wish to design and construct a system that assists blind and visually impaired people in navigation by providing haptic feedback on obstacles in their walking path. With this research, we look to the work in [46], which presented a real-time wearable system for providing vibrational feedback to signal obstacles to users, as guidance. We aim to build upon this device by allowing the device to be used effectively in not only indoor environments, but outdoor environments as well, with a focus on obstacles that are hanging over the user's path (referred to as overhang obstacles). We hope to design a device that can detect these overhang obstacles well, even in different lighting conditions.

Approximately 285 million individuals are blind or visually impaired (BVI), according to the World Health Organization [33]. That number is only growing, with one person losing their vision each minute. Blindness offers difficulties in mobility that people with normal sight do not tend to worry about. While actions such as crossing an intersection, walking down the street without bumping into people and passing through doorways are simple for one with sight, these tasks are much more difficult for BVI people.

They can use tools such as walking canes or guide dogs to help with these tasks, but the tasks still come with their own set of challenges. For instance, a cane user naturally samples their immediate environment with physical contact, but in some situations it can be desirable to avoid contact if possible, e.g. when navigating amongst pedestrians

or in quiet environments. Moreover, walking canes are not effective for detecting overhang obstacles. A guide dog user relies upon the dog to function as their eyes and guide them where it's safe to walk, but guide dogs can get distracted or lose focus fairly easily, and it is left to the user to determine when/if that happens and get the dog back on track. Also, very crowded environments are difficult to navigate with a guide dog, and can still lead to unintentional collisions for the BVI person. In some cases, these collisions may result in bodily harm. Furthermore, since the walking height of BVI people is much higher than their guide dog, some obstacles in the path of the BVI person are not immediately detected by the dog until collision, and must be trained for the dog to spot and avoid. Even well-trained dogs may miss certain obstacles that have not been exposed to them before. These obstacles include objects such as low-hanging tree branches and street signs–both of which may be commonly encountered in every-day life. Thus, blind people are not as willing to travel independently–especially in locations they are not familiar with [16]. Other devices have been created that alert users to obstacles [], however these devices do not work in tandem with the guide dog.

In this thesis, we construct, experiment with and evaluate a device to provide obstacle detection and avoidance to BVI guide dog users in real-time. The device uses a depth camera as well as computer vision techniques and orientation data from an inertial measuring unit (IMU) to extract information about the range and direction of obstacles in the path of the user. The solution provides feedback on presence of obstacles through a vibration motor attached to the handle of the guide dog harness. A large challenge in this project lies in robustly processing sensor feedback from a portable system and relaying that information back to the user in such a way that meets the needs and goals of a BVI person. Specifically, this thesis contributes a system for constructing, experimenting, and evaluating a light-weight mobile device made for augmenting obstacle detection for guide dog users.

Chapter two reviews research related to the project, including blind navigation and mobile object detection, and how they apply to this project.

Chapter three describes how the device was manufactured, as well as design deci-

sions made regarding the device and design challenges encountered when constructing the device.

Chapter four provides a system overview that includes architecture and capabilities of the system.

Chapter five describes the experiments conducted to test the efficacy of the device and presents the results of those experiments, as well as metrics collected to describe the performance and capabilities of the device.

Chapter six provides conclusions from the research, as well as next steps for research. This chapter also describes lessons learned while taking on this research.

Chapter 2

Related Works

Several recent papers have focused on systems that provide navigational capabilities to a blind person [1, 2, 8, 12, 15, 31, 32, 35, 39]. Most of these systems guide blind people through turn-by-turn navigation, using technologies such as GPS [35, 39], RFID tags [2, 8, 13], and Bluetooth low-energy beacons [1, 10, 25]. Others use a combination of sensors, computation, and feedback components, and are used for functions of local frame obstacle detection/avoidance [24, 45, 30, 18, 9]. This prior research serves as a great stepping stone for the purpose of this project. The following sections explore works related to blind navigation, with exploration into different sensors, user feedback, and systems used for obstacle avoidance.

2.1 Sensors and Cameras

There are a couple of systems that fully relied upon a white cane using just a single sonar sensor and vibration motor to detect above-knee obstacles in the direction of the cane, such as the SmartCane [41], UltraCane [34], and the BatCane [20]. A similar system [17] with a single sonar sensor used in tandem with a white cane detected obstacles at chest height, and signaled the user through a vibrating necklace. Another system worn around the neck was a collision-warning pendant for users with hemianopsial [38] was equipped with a video camera that tracked large features [37] and then signaled collisions through audible beeps based on time-to-collision, rather

than proximity. The system was specifically designed for users that had only lost part of their field of vision, and helped to reduce collisions when compared to using no aid, however the audible beeps were reported as distracting.

Tacit [19] was wrist-wearable device that used a single sonar sensor to provide feedback on the distance of obstacles by applying a pressure to the wrist. This required users to have to move around their hands in order to sense the environment, and no tests or user experiments were actually conducted in order to test the efficacy of the device. The Electronic Mobility Cane [5] was equipped with five sonars mounted along a white cane to sense obstacles and alerted a user via audio or one vibration motor at the handle of the cane.

2.2 Vibrotactile Feedback

An overview by [7] compared vibrotactile actuators that can be used in wearable haptic devices. The ActiveBelt [42] mapped GPS (Global Positioning System) directional information to several vibration motors worn around the waist, but no local obstacle information was provided to the user. A head-mounted device by [4] relayed generic information to the user via air puffs to the front of the forehead. Other systems used 1D and 2D arrays of vibration motors worn around the abdomen provided signals to soldiers [27]. Low frequency Pacinian corpuscle stimulation with up to 128 vibrating elements was used for maintaining a stable hover in helicopters [44]. Vibration units placed on the back of a person potentially widened their field of view within a virtual environment [11]. The haptic interface Force Blinker 2 [3] was a handheld stick that indicate d direction by generating a centrifugal force. None of these studies, to our knowledge, performed tests with blind users to verify their effectiveness in a mobility task.

2.3 Systems for Obstacle Avoidance

Besides guide dogs, white canes are the most common tool for blind people to find obstacles and avoid collisions. While very efficient, a user can detect an obstacle only after physically hitting it with the cane. Such a reactive approach is undesirable, especially when the obstacle is a pedestrian. Moreover, white canes have extreme difficulty with obstacles that are elevated. Researchers have developed supportive technologies that allow blind users to detect obstacles with non-contact sensing [29] using WiFi. Systems have also been created that detect and provide information about obstacles (e.g. distance [21, 23, 26, 29, 36], shape [6, 22, 28], or category [21, 36, 47]) to users. They often use laser [26], ultrasonic [23, 40], phone's speakers and microphones [43], or depth sensing [6, 14, 21, 36, 47] for their obstacle detection. However, it is still challenging for blind people to detect and avoid obstacles, in particular in very dense environments. In those scenarios, it is important to have low-latency haptic feedback for the user. Furthermore, the audio feedback may interfere with a blind user's ability to listen to environmental cues necessary for blind navigation. More complex and detailed user feedback also requires focus that may make navigation confusing. Blind users can follow the system feedback to avoid static obstacles such as chairs, desks, and walls. [46] presents a wearable system uses structured light sensors to detect obstacles and provide situational awareness to blind people using vibrations. However, the sensors do not perform well in direct sunlight conditions, an important use case of our device.

While applicable to inanimate objects, some studies assumed that pedestrians, as dynamic obstacles, can avoid a blind user and thus did not focus on supporting pedestrian collision avoidance [28]. Guide dog users may draw more attention than normal blind users, and thus may be avoided more easily by pedestrians.

Chapter 3

Technical Approach

As mentioned previously, guide dog users may suffer collisions with objects such as low-hanging branches and street signs–some of which may cause bodily harm. The goals of this device are to be able to detect objects that are in the path of the blind user to help prevent these types of collisions. There are several ways to design such a device, however after discussion with several blind people and a guide dog company we've taken an initial approach of creating a one-piece device that can be attached to the handle of the guide dog harness.

There are a few additional user requirements for the device that we must satisfy. These include the following:

1. A miniaturized and compact package that is portable and wearable
2. Inconspicuous form, doesn't draw attention
3. Easy to use, requiring minimal training time
4. Long battery life/low power consumption
5. Low cost

Along with the following requirements for device function:

1. High frame rate, low latency, and high reliability

2. Efficient environment recognition and obstacle detection

3. On-board computation of video streams

The device can achieve this using many different types of sensors and extensive computation. However, with these different types of materials bring trade-offs between object sensing, computation, and system usability. The device will need to include sensor, computational and vibrational components so we will focus on small, light-weight devices to aid with the comfortability of the device as we want the device to be as inconspicuous as possible while not in use. Furthermore, the device should be easy to use and effective so we will want to use sensors that are as advanced as possible, and reduce the amount of parts that need to be prepped before use. To that end, we will also need enough computation power on the device to handle sensor data quickly and effectively. We also aim to optimize the load and frequency of computations to maximize the frame rate of the device. Thus, in this work we propose a system that finds a balance between all of these requirements and trade-offs. In this chapter, I will discuss the initial design approach and goals for the device itself, as well as alternative hardware and software options that were considered. We then focus on the methods through which the device was manufactured, as well as how the whole system was implemented end-to-end.

3.1 Hardware

3.1.1 Depth Camera

There were a few directions we could have gone with the sensors, with time-of-flight sensors, structural light sensors and stereo cameras being used in similar studies and offering similar capabilities. TeraRanger offers a fleet of structural light and time-of-flight sensors that are robust in object detection tasks, but perform poorly in direct sunlight. As this is a device that will need to be used outdoors, the need for a device that will work in direct sunlight is paramount. Thus, we will need to look at alternatives to structural light or time-of-flight sensors.

Figure 3-1: The whole device with components connected, placed in the case, and mounted on the guide dog harness handle.

One alternative we can use for our sensor is a stereo depth camera, which has a greater compactness and resolution–both of which are crucial for the best performance in object detection. There are several options within that realm including models in the PMD Pico series, Intel RealSense series, DUO MC series and Stereolabs ZED series. With so many options, the Some things to consider with the camera are the field of view of the camera, the frame rate of the camera, and the relative popularity/ease of use of the camera. The field of view is important because a larger field of view will allow a larger frame for detecting objects. The frame rate of the camera is important because a higher frame rate will allow for quicker object detection when an object enters the field of view. The relative popularity/ease of use of the camera is also important because a more popular and easy to use camera will tend to have more of an online community for issues and bugs regarding the device, as well as a more comprehensive software development kit (SDK).

With all of these factors taken into consideration, the Intel RealSense d435 camera was our camera of choice. The camera features a field of view of $87° \pm 3°$ x $58° \pm 1°$ x $95° \pm 3°$ with a resolution of up to 1280 x 720 and a frame rate of up to 90 frames per second. In addition to having top marks in terms of these specifications, the camera also has an extensive SDK and an extensive online community to consult with when bugs occur. One downside of the camera to note is that it will perform worse in direct sunlight than in shaded lighting conditions. This came up during our experiments using the device.

Before the experiments were conducted, preliminary tests were performed to determine the specifications of the device's object detection capabilities. One case in which we were curious about the performance of the device was with very thin objects. To this end, we define a metric that can describe the distances at which the device is able to detect objects of varying sizes. *First detection distance* is the distance at which the device is able to fully detect an object, without noise in the depth data. We will record this metric for wires of varying diameters so that the size of the obstacle can be described using one number, the diameter of the wire. Each wire is hung up across a hallway and the camera is moved gradually closer to the device until the wire

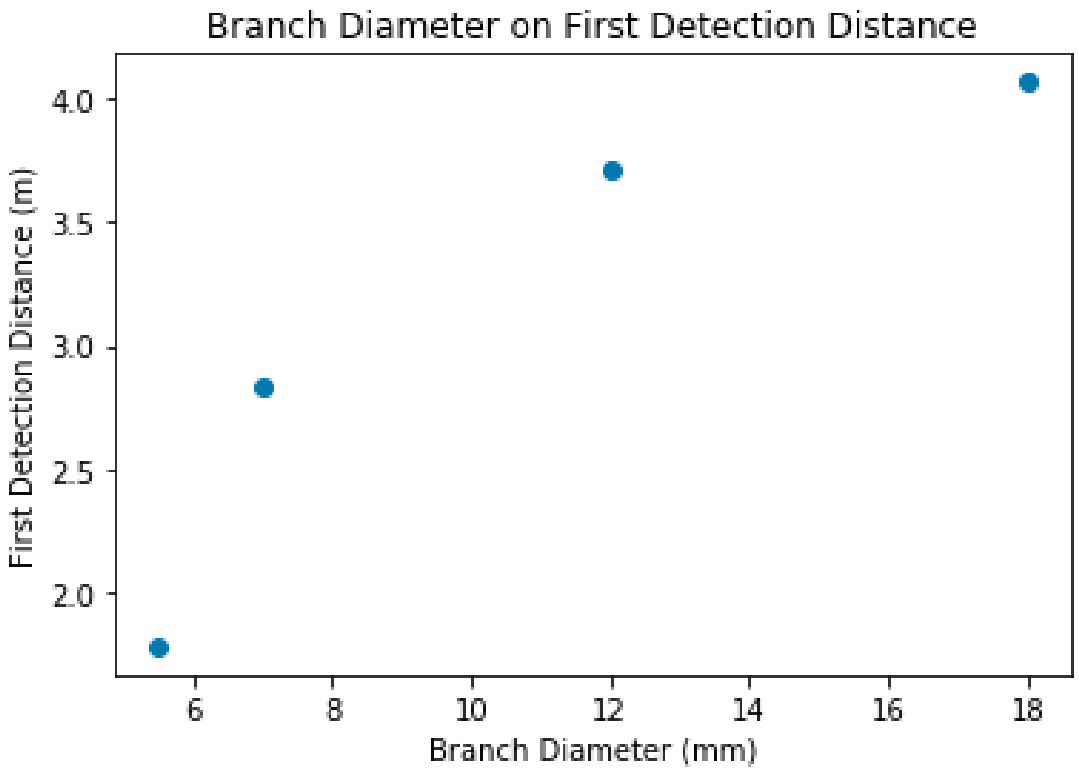


Figure 3-2: First Detection Distance of wires of different diameters

is detected fully. In Figure 3-2 we can see a graph of the first detection distance for wires of different diameters. Notice that the The thinnest wire with a diameter of 5.5 millimeters (about the diameter of a standard ethernet cable or laptop charging cable) had a first detection distance of 1.785 meters. This distance should be sufficient for detecting the obstacles in our experiments and giving the user enough time to stop themselves. Moreover, we assume that real-world obstacles are not as thin as this, and thus will have a larger first detection distance.

3.1.2 Computer

For computation purposes, we needed a computer that could handle a stream of sensor data from a camera and do image processing on that data to find obstacles in the path of the user. Furthermore, the computer must then control vibration motors to send feedback back to the user. There were several options for the computer, as there are many microprocessors out on the market that are able to handle these capabilities,

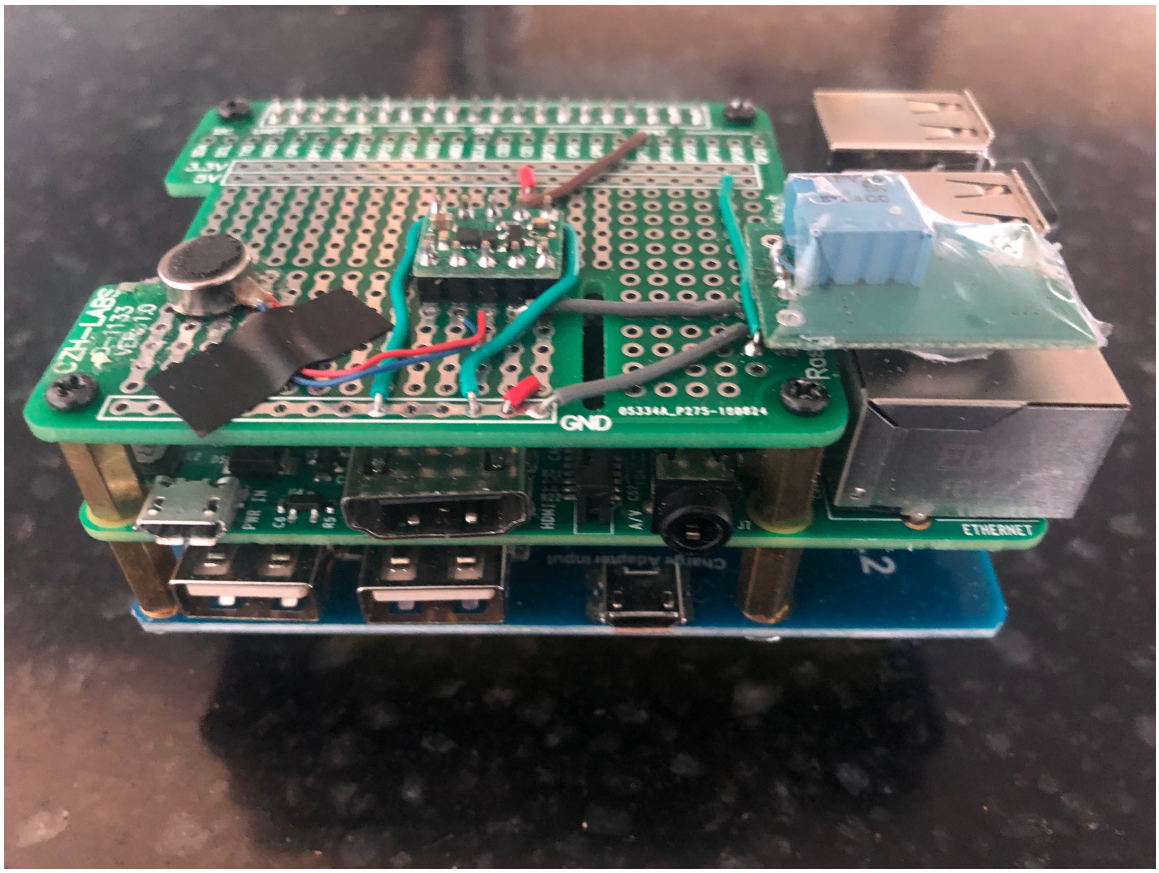


Figure 3-3: 3-layered combination of Raspberry Pi, vibration motor circuit and lithium-ion battery pack. Top layer: vibration motor circuit. Middle layer: Raspberry Pi. Bottom layer: lithium-ion battery pack.

including mbed microcontrollers. Another option is the Raspberry Pi, a single-board computer with the same serial communications protocols that a microcontroller would have, such as UART, I2C, and SPI, as well as general purpose input output (GPIO) pins. It also features several USB ports that make it compatible with the Intel RealSense d435 camera.

With this versatility in being able to control the vibration motors via GPIO, as well as its documented use with the d435 camera, we chose to use the Raspberry Pi. The fact that it also comes as a single board is an important feature too, as it simplifies the manufacturing of the device and limits the amount of soldering that needs to be done. The middle layer of Figure 3-3 shows the Raspberry Pi we used and its HDMI and micro-usb connection ports.

3.1.3 Vibration Motor

The vibrational feedback needs to be able to be take input controls from the Raspberry Pi and vibrate at a level that can be felt by the users. This requires some testing and fine-tuning to determine the optimal level for users. If we set the vibration level too low, the vibrations won't be felt by the user through the case and the handle. Too high, and the vibrations will distract the guide dog and disturb the user. Through experimentation with different sized vibration motors, we determined that a vibrational force of 0.4-0.5 G would be sufficient for use with the device. Thus, we can use a coin vibration motor or rotating mass motor for the device, and control the vibrations through the GPIO pins of the Raspberry Pi.

3.1.4 Circuit

The vibration motor will be connected to the Raspberry Pi's GPIO pins and draw power from the Pi itself. Specifically, the motor must be connected to a pin (pin 13) that offers pulse width modulation (PWM) in order to drive the vibration motor. However, the vibration motor is controlled by a motor driver that takes a voltage of 2.9 volts, while the PWM pins supply voltage at levels of 5 volts. Thus, the voltage will need to be down-regulated to the appropriate voltage levels. This is done with a voltage regulator that is adjusted to take in 5V of input and output 2.9V.

The circuit is soldered onto a breakout board that is connected to the Raspberry Pi using brass standoffs and pin connectors so that it sits on a level directly above the Pi itself. The top layer of Figure 3-3 shows the circuit used to power the vibration motor.

3.1.5 Power Supply

In order for the device to be mobile, it will need to have a portable power supply to power the Raspberry Pi. There are lithium-ion battery packs specifically made for the Raspberry Pi that are compact, and can attach firmly with screws to the bottom of the Pi using brass standoffs. The battery pack connects to the Pi using micro-usb,

and can be seen in the bottom layer of Figure 3-3. A second power supply is used to supply power to the camera to avoid a current drop that would cause the Pi to freeze. For this power supply we use a portable USB battery pack.

This second power supply connects to a USB hub that allows the depth camera to stream data to the Pi while still drawing power from an outside power instead of the Pi itself.

3.1.6 Case

The device as a whole will need a case to house and protect the components, as well as mount the device to the guide dog's harness handle. To make the case, we created a design in SolidWorks and 3D-printed the case in plastic using a Fortus 400 3D printer. The plastic filament used is sturdy enough to withstand impact and protect the internal components in case the device is dropped. A mechanism to easily mount and unmount the device from the harness handle will be crucial for blind users. Allowing the components to be easily removed from the case will also be important for prototyping.

Container

The components that the case needs to house has four moving parts. One is the combined Raspberry Pi with vibration motor circuit, and lithium -ion power supply. Second is the depth camera, third is the USB hub, and fourth is the additional power supply for the camera. Due to them having the same relative shape, the combined Pi/vibration motor component and second power supply can be housed in the same cube-like container (which will be the main container as it takes up the most space). The other two components will have their own container. Since we want the components to be easily removed from the case, each separate container will have an open top. To minimize the number of holes in the device, the components will be oriented such that sides that need access to connecting cables will be towards the open top (this will not eliminate the need for extra holes, however).

The main container will have an open top to allow access to the Pi's USB ports, as well as holes on the side and bottom to allow access to the HDMI/power ports and power switch, respectively. The USB hub is longer and thinner than the other components so it will be in its own compartment next to the main container, with a hole at the bottom for access to its micro-USB port, and an open top so that the USB hub can connect to the camera. The camera is an odd shape resembling a pill with one flat side, so a regular rectangular prism shaped container will not work. For this compartment we added side supports to keep the camera in place, and took out part of one side of the compartment to provide a window for the depth camera lens.

Mount

The case will need to be mounted to the handle of the guide dog's harness in a way that it can be mounted and unmounted easily. Specifically, the case will be mounted on the cylindrical rod (14 millimeters in diameter) of the handle, and will need to be held securely in place so that the device doesn't rotate while in use. For the initial design, an adjustable bike clamp was to be used to secure the case to the handle. The clamp itself would be connected to a protrusion on the back of the case using screws, with nuts to secure them in place. The adjustable nature of the clamp made sure that the case would be held securely in place. However, using the bike clamp would've required the whole handle to be disassembled to attach the device.

To avoid this, we design a plastic clamp similar to a two piece shaft collar which will be attached to the case using screws TODO: insert figure?. The plastic clamp is two parts that wrap around the rod and interlace with each other. They have slots so that the two pieces can be held together with screws, although the pieces interlock snugly enough to be held together without screws. The clamp is then attached to a protrusion at the back of the case using two screws.

In the final iteration of the mount's design, one piece of the clamp is printed directly on the case to move the device's center of gravity closer to the handle, making it much easier to be held securely without rotating. Figure 3-4 shows the two parts that went into the final mount design.

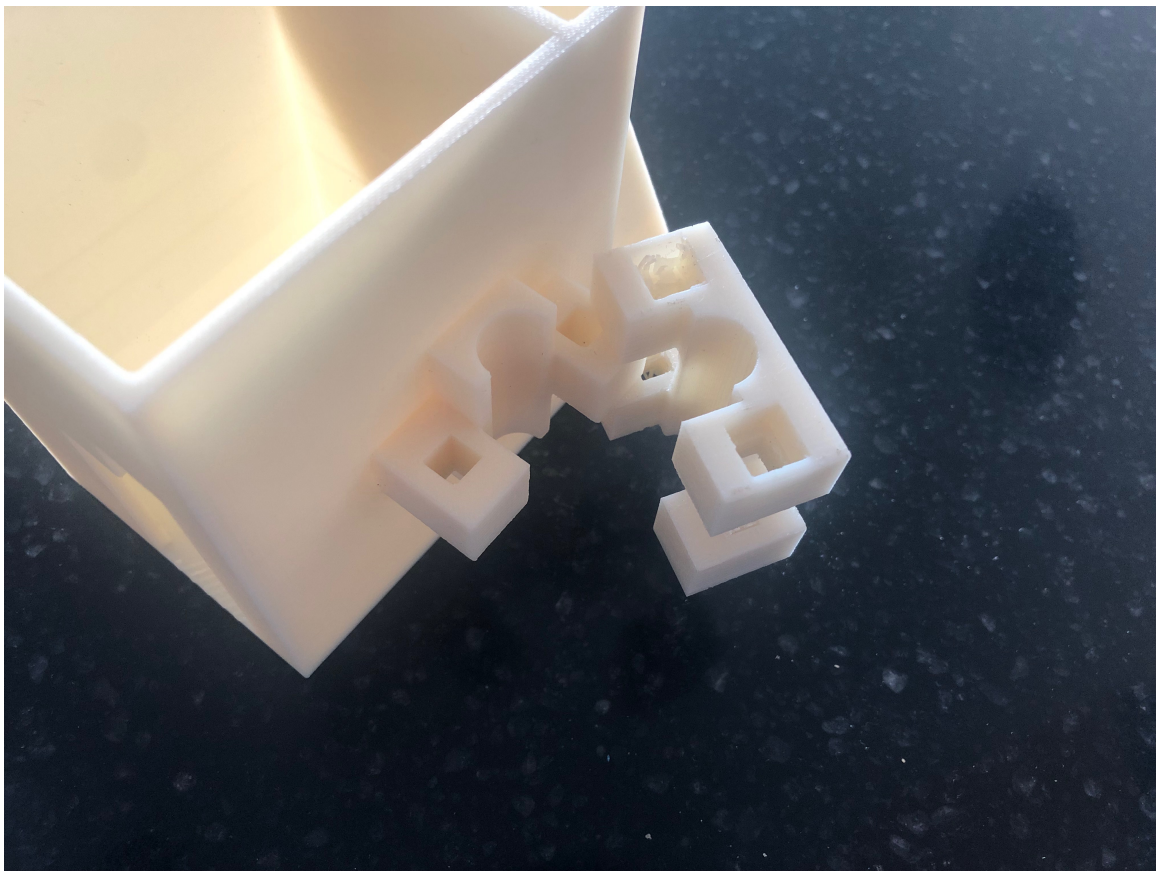


Figure 3-4: Two part plastic shaft collar printed and used as mount for the device

Camera Angle

Since the case is attached to the rod of the harness, the case will be aligned parallel to the harness rod. When a guide dog is leading a blind user, the harness rod will be at a certain angle to the ground (theta). In order for the camera to be able to detect objects in the user's path, it must be oriented at an angle so that it can actually see the path. Thus, testing must be done in order to determine the optimal camera orientation angle.

Initially, it was believed that having the camera parallel to the harness rod would suffice, because we are aiming for a view that is above the height of the dog and not too high that it is above the height of a human (a "target region" for object detection). That, along with the large field of view of the camera led us to this conclusion. However, in initial experiments using the device with guide dogs, it became very clear that this camera angle would not work out.

Measurements were collected from these experiments on the range of angles that the harness would be in during use, and from there a couple different angles were tried. Two cases were designed that tilted the camera's orientation angle forward by 45 degrees and 30 degrees, respectively. Between these two angles, the 30 degree case allowed the camera's field of view to capture the target region more effectively.

3.2 Software

Once the hardware is built, the device will need software developed to handle depth image processing and control the motor. The depth image processing will contain the algorithm for determining the path of the user and object detection within that path. For the development and testing of the object detection algorithm, a separate pipeline was also produced to make the development process simpler.

3.2.1 Operating System

The operating system (OS) for the Raspberry Pi runs on a micro SD card that's inserted into the device. Raspbian and Ubuntu Mate are the two main OS used with the Raspberry Pi, and each has its own reasons for use. While Raspbian is a Debian-based OS commonly used for projects involving controlling devices (such as led lights or vibration motors) using GPIO pin outputs, while Ubuntu Mate is a Ubuntu-based OS that has extensive documented use with the Intel RealSense library. Since both are fairly comparable, we decided to move forward with Ubuntu Mate.

3.2.2 Code

We decide to write the code in C++ using CMake to build and compile. The code will include a main file that contains the object detection algorithm code, as well as the code for controlling the vibration motor. There will also be code for displaying the depth and color video streams, as well as feedback printed to the terminal about if the algorithm detects any object, and what distance away the object is.

Motor Control (GPIO)

The vibration motor's circuit is connected to the PWM pin (GPIO pin 13) and must be controlled within the main C++ file. Normally, the GPIO pins can be controlled using a library called WiringPi, a GPIO access library similar to existing Arduino libraries, intended specifically for use with the Raspberry Pi. Specifically, the library includes a software-driven PWM handler capable of outputting a PWM signal to the specific pin. For our purposes, we used two functions in particular. One is a method that creates a software controlled PWM pin, and will be called once in our program to initialize the pin for control. The other method is one that updates the PWM value on a given pin, which will be called every time we receive a new depth frame.

Object Detection Algorithm

Our goals for the object detection algorithm are that we must take in depth frame data, estimate the path of the user, and then trigger the vibration motor if there is an object detected within that path. The algorithm must also do all of this processing quickly in order to relay the vibration signal back so that the user has enough time to avoid a collision. The software development kit (RealSense SDK) for the Intel RealSense camera includes libraries and methods to allow for real-time processing of their data streams.

In order to estimate the path of the user, we must determine parts of the camera's viewing frame that would constitute the user's walking path. The position of the user in relation to the device is usually fixed, as guide dog users always hold the harness handle in their left hand. Thus if the camera is pointing straight forward on the guide dog, the path of the user will be a person's width to the right of the middle of the viewing frame.

We estimate whether an object is within this width through angle estimation and simple geometry. For a particular pixel at a distance x meters away, we take the vertical line that splits the frame in half (the center of the camera's field of view) and count the number of pixels to the right of the mid-line the pixel is. This number is divided by the number of pixels in a horizontal row divided by two and then multiplied by half of the camera's field of view in the horizontal direction to give us an estimate of the angle. This angle estimation is not exact, but it will suffice for this path estimation. Multiplying the sine of this angle by x should then give us a horizontal offset value for the object, and then that value can be used to determine if an object is within the width of the user's estimated walking path (assuming they keep moving straight forward). This also assumes that the camera's field of view in the horizontal direction is 58 degrees. This of course is only effective under another assumption that the camera has not rotated on the harness rod and is still pointing straight forward, and only is effective in estimating user's walking path width. The height of the user's estimated walking path is harder to estimate due to the handle's movement while the

dog is walking. For that estimation, reducing the viewing frame of the camera in the vertical direction was done to estimate the height of the user's path. For the rest of this section, the estimation of the user's path will be referred to as the "target path".

Now that the target path of the user has been estimated, the depth frame can be analyzed to determine whether there is an object within it. To do this, the individual pixels of the target path as examined and each one is compared to a maximum clipping distance. In initial iterations of the algorithm, the number of pixels within this clipping distance are counted and compared to the total number of pixels of the target path. In the most recent iteration of the algorithm, a cluster finding algorithm is used to determine whether there are clusters of pixels in the path of the user within a certain distance of the user. The assumption here is that a big enough cluster will be indicative of an obstacle in the path of the user. The clusters are found using a breadth first search (BFS) of pixels that are within the user's target path that are within a certain distance. The size of the largest cluster is recorded and then compared to a threshold to determine whether the cluster is big enough to constitute an obstacle. If so, then the vibrations are triggered.

Algorithm 1 Path Planning and Object Detection Algorithm

```
1: procedure PATH PLANNING
2:     left ← 1/2 * width
3:     right ← 8/9 * width
4:     up ← height
5:     down ← 1/2 * height
6: loop over box created by up, down, left and right:
7:     if distance < 2.5 then
8:         if horizontal offset < path width threshold then
9:             map ← pixel
10: procedure OBJECT DETECTION
11: loop over pixels in path planning map:
12:     cluster finding using BFS
13:     record size of largest cluster
14:     if ( thenlargest cluster > cluster size threshold)
```

One aspect of the device that's important to maintain is a high frame rate. If the frame rate is too low, then the object detection latency of the device will be too slow, decreasing the stopping distance that the device allows. We record that the

device has a frame rate of 15 frames per second when the algorithm is running and no obstacles are detected. Since there is a fair amount of computation taking place in this algorithm, we notice a drop in frame rate to about 5 frames a second when there are obstacles detected in the path of the user. This is due to the fact that the algorithm undergoes significantly more computation if it detects pixels in the target path that are within the device's object detection range.

The distance used for this object detection range was determined through preliminary testing on the device, as discussed in section 3.1.1, as well as feedback from previous round of experiments. The first detection distance mentioned in that section confirmed that the depth sensing capabilities of the camera are sufficient enough for detecting objects at a distance, as it is able to detect wires 5.5 millimeters in diameter at a distance of 1.785 meters. In the first iterations of the object detection algorithm the object detection range was set at 1.5 meters. However, this was shown in the experiments to be shallow of a range for the object detection, as the stopping distance allowed to users was too little. This could be attributed to the walking speed of the person as well as the processing time of the algorithm to trigger the vibrations. Thus, the range was increased to 2.5 meters for the most recent iteration.

The algorithm also applies post-processing filters to the depth data to allow for simpler computations which in turn lead to a higher frame rate. The filters used specifically are the decimation and spatial filters. The decimation filter intelligently reduces the resolution of a depth frame, which reduces the number of pixels that the algorithm processes with each iteration. The spatial filter applies edge-preserving smoothing of the depth data, which reduces some of the noise that can occur in the depth data, especially in direct sunlight.

OpenCV and OpenGL

We used both OpenCV and OpenGL for visualization of the detph data. While we initially used only OpenGL, as it provided quick real-time visualization of the processing done by the algorithm. OpenCV was then introduced to allow for use with its computer vision libraries, as well as for offline develpment of the object

detection algorithm.

3.2.3 Dataset Pipeline

Also in CMAKE and C++ To aid with refinement and iteration of the object detection algorithm, we created a dataset pipeline where we create our own datasets by saving video streams recorded by the RealSense camera. Then we can read in the datasets and apply our algorithms to them, and see if it correctly detects objects and triggers vibration signals. This offline nature of the dataset pipeline is particularly useful because it allows us to judge the correctness of our algorithm without having to test it online on the device. This made our testing process easier as we did not have to set up the device every time we made a change to the device. All of the pipeline was written in C++ and built using CMake.

Dataset Creation

For creating our dataset, we will need to For our purposes, the datasets we want to create are ones that emulate the particular use cases that we are interested in (e.g. low-hanging branches/wires, street signs, etc.).

Dataset Reading

Once the datasets have been recorded, we need a pipeline to read in the datasets from their respective files. A bag file is specified containing both a stream of depth images and a stream of color images. The streams are aligned and then can be read and displayed in proper order.

Algorithm Application

After the datasets have been read in and aligned correctly, our object detection algorithm can be applied to the stream frame by frame. The response of the object detection algorithm can be observed and then be evaluated.

3.3 Design Challenges

While designing and manufacturing the device there were several challenges encountered, both on the hardware and software sides. The following sections are constructed similarly to the system design section, with the hardware design challenges being followed by the software design challenges.

3.3.1 Hardware

The following section discusses design challenges encountered with the hardware of the device. Specifically issues that will be discussed pertain to issues with the power supply, certain circuit components breaking down over time, and reasons why case re-designs were necessary.

Power Supply

In the final product of the device, we have two power supplies. One is a lithium-ion battery attached to the under-side of the Raspberry Pi, and the other is a standard USB battery pack. Initially, our plan was to use only one power supply, but issues arose when we started trying to run both the Raspberry Pi and the depth camera from a single power supply. The camera needed to be plugged into the Raspberry Pi, but the camera would draw too much current causing the Raspberry Pi to freeze. Thus, the USB hub was introduced into the system so that the camera could use a separate power source while the Rasperry Pi used its own power source.

The choice of second power supply wasn't trivial, and initially a second lithium-ion battery was used to power the USB hub. However, this ended up taking up too much space and a USB battery pack was used as the second power supply instead. We were able to find a USB battery pack that was the same width as the Raspberry Pi so that it was small enough to fit into the same compartment as the Raspberry Pi. The case was designed so that it would be able to fit both of these components together.

Circuit

Now that the second power supply was in the same compartment as the Raspberry Pi, with the power supply being placed directly on top of the Pi, the circuit components suddenly had a weight sitting on top of them. Initially this was not seen as an issue, but over time the voltage regulator started malfunctioning and wouldn't trigger the vibrations correctly. What made this even more peculiar was that the voltage regulator would work perfectly fine when held down. Thus, a simple solution for this was to wrap a rubber band around the device to hold down the voltage regulator. In the future, this component would need to be replaced outright, but with this setback being presented so late in the year, the rubber band sufficed. In future design iterations the power supply would have its own compartment, and the case would be designed to better protect the circuit components.

Vibrations

While designing the device and determining the type of vibration motors to use, we failed to consider whether the vibrations would be felt through the handle by a wide range of individuals. While users who tested the device's vibrations in lab felt that they were easily noticeable, it became clear in our experiments that the vibrations were not as easy to feel when the device was in motion. Some of the test subjects reported not being able to feel the vibrations at all, even when the device was vibrating at its highest power. Moreover, the vibrations were light enough that the user would need to really concentrate in order to feel them, drawing their focus away from other cues (i.e. the dog getting distracted or audio traffic cues) that they also needed to pay attention to.

This also ended up influencing a decision on the algorithm side for conveying distance of objects to the user. While in previous iterations of the algorithm the distance of objects was conveyed by the strength of vibrations, this feature was thrown out in later iterations due to the varied sensitivity found in our test subjects. This feature could be brought back in the future if stronger vibration motors were used,

or if the vibration motors were attached directly to the user to be felt more strongly.

Case Mount

An intense re-design of the guide dog's harness handle was occurring concurrently to this project, which meant that designing the mount for the device would need to happen without access to the handle itself. Thus, without knowledge of the structure of the harness handle, a bike clamp was initially used as a mount before discovering that the bike clamp could only be put on by dismantling the entire handle. Thus, this design wasn't usable, and another solution needed to be found. Eventually, a two piece plastic shaft collar was designed and used for the final iteration of the device.

3.3.2 Software

There were several issues that arose regarding the coding portions of the project, mostly involving the path planning and object detecting algorithm. There were also issues with the dataset pipeline that made the pipeline difficult to use.

Algorithm

High frame rate is very important for the performance of the device. As noted in section 3.2.2, the frame rate drops significantly when there is an object detected in the user's target path. This is because of the increase in computation that the algorithm does on the whole frame when there are a lot of pixels within the object detection range of the device. This made it necessary for some optimizations with path estimation and object detection of the algorithm. It would be infeasible to check each pixel in every depth frame and determine if it is in the path of the user. Thus, a restricting of the depth frame to an area slightly larger than the user's target path is done first to restrict the number of pixels that the algorithm must process. Also, the BFS done to search for clusters of pixels is only done on pixels that are within the object detection range of the device.

Previous iterations of the device were naive in that they detected objects by

taking the pixels in the user's target path and counting how many of them are within a max object detection range of the user. However, this really only worked for very large obstacles, and would only work effectively when the user was very close to the obstacle. Thinner obstacles were not able to be detected by this algorithm. Thus, a new algorithm was written that looked for clusters of pixels.

Dataset Pipeline

Several issues came up in the dataset pipeline that made it difficult to use. While recording datasets, the device would usually be in use and need to move around a set space to record a useful dataset. Thus, the Pi couldn't be connected to an HDMI monitor, and the recording itself would usually need to be done without a screen. This made it particularly difficult to record datasets of particular edge cases without a little guess work involved. Another interesting thing about recording datasets was the noticeably lower frame rate as opposed to streaming the data live. When the device is live, it would have a frame rate of 5-15 frames per second, however that number would drop to 2 frames per second while recording. This could be resolved by recording the datasets slowly, although this coupled with the inability to watch what the device is recording while it is recording made dataset recording less than ideal.

Once the datasets were recorded, they were stored in bag files. Another issue that arose was the need to have the camera connected when these files were read. We used methoeds found in the Intel RealSense SDK to read the bag files, and there was no workaround documented for being able to read the bag files without the device being connected. This made it difficult because the offline development of the algorithm using this dataset pipeline would need to occur on the Raspberry Pi while it was connected to the camera. Although the pipeline was still usable, and was particularly useful with testing the algorithm offline, these issues made the pipeline less effective than it could have been.

Chapter 4

System Overview

In the previous chapter I discussed design decisions and issues encountered while constructing the device. This chapter provides an overview of the system and its capabilities. Figure 4-1 shows the system architecture and the capabilities of the system. Environment sensing and obstacle detection is achieved using a stereo depth camera, allowing the camera to be effective at detecting obstacles both indoors and outdoors. It does see a performance decrease in direct sunlight.

Figure 4-1: System Overview: The depth data is taken as input and is used to estimate object distances from the user. The data is then filtered to determine what objects are in the user's path. If there is an obstacle detected, then a vibration motor is triggered.

Before the experiments were conducted, preliminary tests were performed to determine the specifications of the device's object detection capabilities. One set of metrics collected was the allowed range of orientation of the device that would still allow the user's estimated path to be visible. As the guide dog user walks, the angle that the handle makes with the ground goes through an oscillation motion with each step. Different guide dog users may also hold their handle at different angles than

others. In addition to this, the device may rotate on the harness handle while in use. Thus, a large allowed range of device orientation would be preferred to maximize the performance of the device.

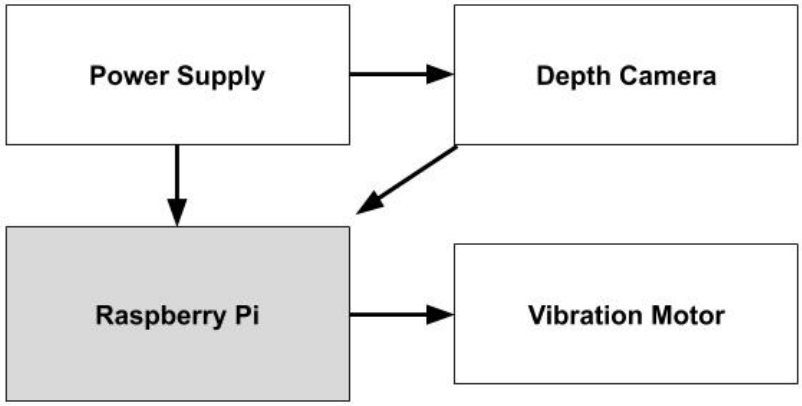


Figure 4-2: Hardware System Architecture: The Raspberry Pi and the Depth Camera draw power from the Power Supply. The Depth Camera sends input to the Raspberry Pi to be processed. Signals are sent to the Vibration Motor if an obstacle is detected.

We record this allowed range of device orientation through two metrics, the allowed harness angle range and the allowed device yaw range. We define the *allowed harness angle range* as the range of angles that the handle could make with the ground (with 0 being set as the handle being parallel to the ground) such that the device is still able to capture the entire estimated path of the user. We define the *allowed device yaw range* as the range of degrees that the device could rotate around the handle (with the device facing straight forward being set as 0) such that the device is still able to capture the entire estimated path of the user. Through testing with the device, we determined the allowed harness handle range to be from 24 to 54 degrees, and the allowed yaw range to be from -9 to 9 degrees. These allowed ranges are more than sufficient for the device to be able to be used with guide dog users.

Figure 4-3 provides a hardware overview of all the system components and figure 4-2 shows the architectural layout for the hardware. The depth camera is connected to the Raspberry Pi using a USB-C cable. The algorithms are implemented in C++

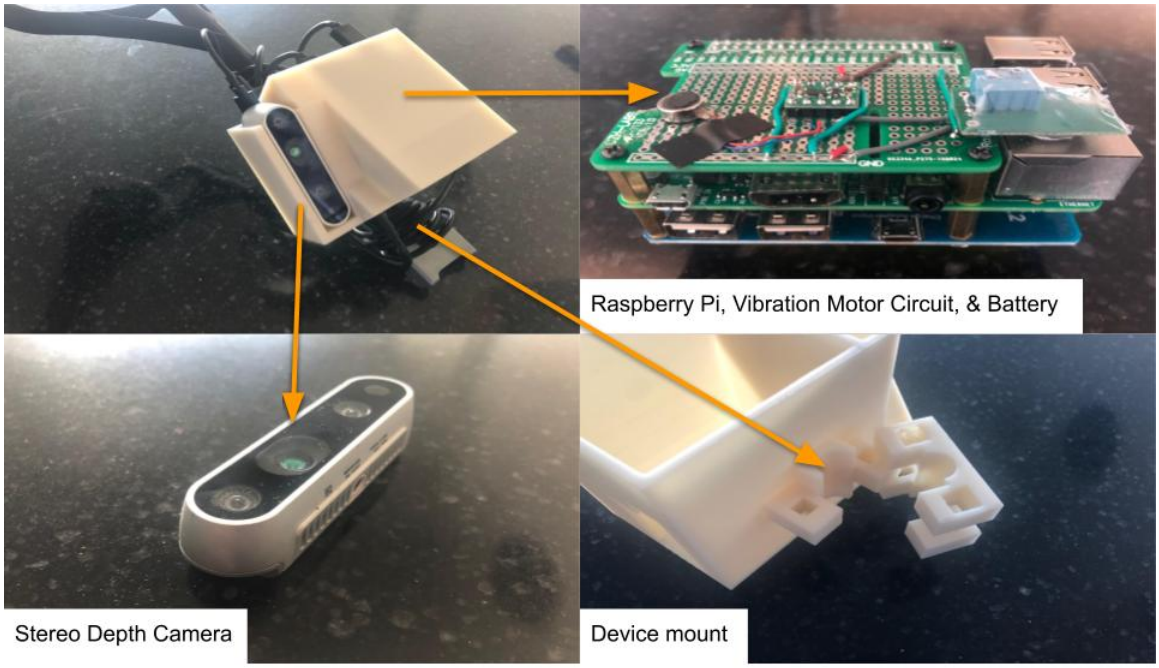


Figure 4-3: Hardware Overview. Top left: the full system mounted on the harness handle. Top right: a 3-layered component containing the Raspberry Pi, vibration motor circuit, and one power supply. Bottom left: the Intel RealSense d435 Depth Camera. Bottom right: the 3D-printed collar shaft used to mount the device on the handle.

with OpenCV and WiringPi, a arduino-like library used for Raspberry Pis. The vibration motors vibrate at a force of 0.4 G, about the same strength of a cell phone vibration.

Chapter 5

Results

Now that the device has been built, it's important to test the effectiveness of the device in real-world situations. The effectiveness can be measured in both an absolute sense (was the device able to detect an obstacle) and in a relative sense (to what degree was the device able to detect objects). For both of these, we will define metrics for measuring the performance of the device. In this chapter we describe the experiments conducted to test the efficacy of the device, as well as the implications of these results.

5.1 Experimentation Methods

In order to properly test the device, we need to observe the device in-use with our target users–specifically observing them using the device with their guide dogs to navigate and avoid obstacles. Before guide dogs are given to their respective owners, they undergo training that involves conditioning the dog to avoid obstacles along a sidewalk, such as traffic cones, barrels, and traffic barricades. In a similar vein, we set up an obstacle course full of over-hanging obstacles that users may encounter in every day life, such as low-hanging tree branches and street signs. Parts of our obstacle course were both indoors and outdoors to simulate some of the changes in environment that a blind or visually impaired (BVI) user may encounter while using the device. Examples of obstacles that were used included caution tape hung over a bridge at chest-level (to emulate a bridge closed for construction) and a tree branch

hung over a sidewalk at chest level. Our test subjects for these experiments are guide dog trainers from Guiding Eyes for the Blind, who have plenty of experience traversing similar obstacle courses. They were targeted as test subjects due to their accessibility and familiarity using guide dogs. The trainers were required to wear blindfolds to better emulate a BVI user.

In Figure 5-1, a blind-folded test subject is shown approaching an obstacle with the device mounted on the harness of the guide dog. The different test cases we wanted to test for were a combination of simple obstacles which a dog should already detect (such as doors) and obstacles that are above the height of the dog and may go undetected. These obstacles are arranged in a controlled path which our users travel through. While they are traversing the path, metrics are recorded on the ability of the device to detect objects, and the stopping distance allowed for users when the device does detect objects. The test subjects indicated when the device was vibrating but saying "vibration" and saying when the vibration ceased by saying "no vibration."

The path that we have users traverse is a 200 meter long loop, with an obstacle set up every 50 meters. This gave users more than enough distance to approach each obstacle, giving a very clear definitive measurement for our target metrics. For our most recent set of experiments we had a total of 8 test subjects who walked through the obstacle course, with each test subject walking through the obstacle course twice. Thus, the experiments had 16 trials, the results of which we will present in the sections below.

5.2 Evaluation

In the experiments, metrics were collected on the ability of the device to detect obstacles, as well as the stopping distance afforded to the users. These metrics test both the absolute object detection ability for different objects, as well as a relative object detection ability.

Figure 5-1: Blind-folded test subject approaching an obstacle with device mounted on guide dog harness

Figure 5-2: Obstacles used for the most recent round of experiments. Top left: Caution tape. Top right: Caution bar. Bottom left: White PVC pipe. Bottom right: Hose.

Table 5.1: Object Detection Probability and Average Stopping Distance (in feet) for all obstacles (n=16)

Obstacle	Object Detection Probability	Average Stopping Distance
Caution Tape	1.00	3.72
Caution Bar	1.00	6.65
White PVC	1.00	6.17
Hose	0.4375	1.39

5.2.1 Object Detection Probability

One metric collected to describe the ability of the device to detect obstacles was the probability of object detection. The metric measures, for different obstacles, the number of times it was correctly detected by the device across all trials. Figure 5-2 shows the obstacles used for the experiments, as well as their respective labels.

Table 5.1 describes the object detection probability across the four obstacles used for our most recent round of experiments. While three of the obstacles were able to be detected consistently, the fourth obstacle had a much lower object detection probability of 0.4375. This discrepancy appears to be due to several key differences between this obstacle and the others used. While the other obstacles were at chest/-torso height, the hose obstacle was at head height. The hose was also noticeably thinner compared to the other three obstacles, making it much harder to detect.

While the object detection probability is good for differentiating which objects were difficult for the device to detect, there was a need to differentiate between objects that the device could detect well. The caution tape, caution bar and white PVC obstacles all had a object detection probability of 1.00. To determine the differences between these objects, the stopping distance was collected across all 16 trials, and an average was computed from these values. In trials where the object was not detected, the stopping distance recorded as 0.0. Table 5.1 also contains the average stopping distance for each of the obstacles.

From this table, we can see that for the obstacles with high object detection probability, the stopping distance that the system afforded was more than sufficient

to keep the users from collisions. From the average stopping distance, we now have a picture of how easily the device was able to detect each obstacle. Specifically, among the obstacles that the device had a high object detection probability for, we can see that the device offered the greatest stopping distance to the caution bar obstacle and offered the least stopping distance to the caution tape obstacle. If we assume that a larger average stopping distance indicates that an object is easier to detect, then this metric offers a clear progression of the detectability of the obstacles we used. The caution bar obstacle is easier for the system to detect than the white PVC obstacle (although they have relatively similar detectability), and the white PVC obstacle is easier for the system to detect than the caution tape obstacle.

There may be several factors that lead to this similarity in stopping distance between the caution bar and the PVC obstacle. The obstacles are both fairly thick and are both located in similar lighting conditions. Meanwhile, the caution tape obstacle is thinner than both of these obstacles, and is located in direct sunlight. These disparities can help explain why the average stopping distance of the caution tape obstacle (3.72) is so much lower than the stopping distance of the caution bar (6.65) and white PVC (6.17) obstacle.

The system also experienced instances where vibrations were triggered without obstacles in the user's path. These false positives happened at the same sections along the obstacle course and all of them contained overhanging tree branches that were in the vision of the camera but too tall to be in the way of the user. In Figure 5-3, an example of a low-hanging tree branch which would trigger a false positive is shown. These false positives could be attributed to the large detection range of the device, as well as the orientation of the device on the handle. The depth data is taken as a distance relative to the device, with our clipping distance set at 2.5 meters. Since the device has a wide field of view in the vertical direction and is angled upwards, the device may notice an object that is 2.5 meters away, but is above the head of the user. The path planning algorithm could have been adjusted downwards to reduce the number of false positives. but that would've led to a greater risk of missing obstacles at a person's head height.

Figure 5-3: Example of a low-hanging branch (circled in red) which can cause false positives

5.3 Results

From the results of the experiments, we can see that the device does particularly well with detecting most objects that it encountered. Moreover, it offered enough stopping distance for users to safely avoid collisions. What it has trouble with is with very thin objects in direct sunlight, though obstacles found in every-day use will usually not be so thin. The device also has a tendency to vibrate even when there are not obstacles in the user's target path if there are low-hanging branches over the user's head. These false positives made users more tentative to stop when they felt the vibrations of the device, as they weren't sure if there really was an obstacle in their path or not. However, some test subjects did mention that the obstacles that led to false positives usually had more intermittent vibration patterns, while the actual obstacles had a more steady and constant vibration pattern. Still, this amount of focus to discern between actual obstacles and likely false positives was a distraction that affected users' ability to focus on their environment.

One aspect that makes this even more of an issue is the fact that it is difficult for a blind user to confirm whether there is an obstacle in front of them or not. The device doesn't convey information about the location of obstacles in the user's path, only

the presence of the obstacle. Thus, it is up to the user to attempt to find the obstacle once they feel the vibration signal. This is a tedious process that can significantly increase the time a user would need to traverse a path. Test subjects did suggest that the device is good for confirming knowledge of obstacles in a somewhat familiar environment, but that it would be difficult to use in unfamiliar environments where knowledge of possible obstacles is unknown.

From questionnaire feedback, test subjects generally agreed that they were satisfied with the object detection capabilities of the system. Specifically, they found that the device was easy to acclimate to, the device's vibrations were easy to interpret, and that using the device was just as comfortable as walking without it. While some felt that the vibrations were too weak, they all agreed that it was effective at detecting obstacles and that they did feel safer using the device.

Chapter 6

Conclusion

In this paper we presented a real-time wearable system, which includes a camera, an embedded computer and a vibration motor that provides vibration feedback to its users. Using depth information from a stereo depth camera, we were able to detect obstacles and send vibrations to the user giving them enough time to stop and avoid collisions.

We conducted experiments with guide dog trainers which involved traversing a path of obstacles using the device while blind-folded. We were able to show that such a device can work effectively to account for mistakes that a guide dog may make, and decrease the number of collisions compared to the use of a guide dog alone.

Previously, guide dogs had to be trained to recognize and avoid low-hanging obstacles, such as low-hanging tree branches and street signs. Even with this training, the guide dogs were not guaranteed to avoid the obstacles due to distractions in the environment. This reactive approach left the user open to collisions and injuries, revealing a use case that the presented system can fill. In post-testing questionnaires, users reported that the device was comfortable to walk with, quick to acclimate to, and provided signals that were easily interpretable.

Haptic devices provide a high frame rate and low-latency feedback, which is desirable for navigation tasks involving avoiding obstacles. Audio feedback was deemed undesirable due to unsuitability in loud environments and possible interference with audio cues that blind people rely upon to navigate. A stereo depth sensor was used

for usability both indoors and outdoors, with better performance in direct sunlight than structured light sensors.

One important lesson learned with this project was the importance of dedicating time to design decisions, and knowing that a design can and should be scrapped if there exists a better solution. An obvious example is that the case design was changed several times to accommodate for several things, including the addition of more components or the change of camera orientation angle. Another example is the choice to have the device held in a single container on the guide dog's harness handle, instead of attaching either the camera or the vibration motor to the user–in future iterations it may be advantageous to move them to be on the user. Future work includes low-power incorporation of the IMU into the algorithm to allow for height estimation of objects, allowing for more accurate path estimation and fewer false positives from obstacles like the ones shown in section 5.2.1. Another area would be to incorporate the acceleration readings from the IMU's accelerometer to calculate the device's speed at any instant. This speed reading could then be used to automatically change the maximum object detection range of the device based on the speed of the individual.

In this thesis, we have discussed the design, construction and evaluation of a wearable object detection system for blind users to avoid collisions. Ultimately, we hope that this device, with some greater refinement, will be able to improve the safety of guide dog users and make blind people more confident in exploring unfamiliar environments.

Appendix A

Parts List

Hardware:

1. Intel RealSense D435 camera (https://click.intel.com/intelr-realsensetm-depth-camera
html)
2. Raspberry Pi 3 Model B+ (https://www.raspberrypi.org/products/raspberry-pi-3-model-
3. Raspberry Pi Breakout Board (https://www.amazon.com/Electronics-Salon-Prototype-Brea
dp/B07BF8Z3HS/ref=sr_1_7?keywords=raspberry+pi+breakout&qid=1558992407&
s=gateway&sr=8-7)
4. Lithium-ion battery pack (https://www.amazon.com/gp/product/B079M9YQP8/
ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1)
5. USB portable charger (https://www.amazon.com/Portable-24000mAh-Capacity-External-Co
dp/B07PDX41LC/ref=sr_1_10?keywords=usb+battery+pack&qid=1558992699&
s=electronics&sr=1-10)
6. Vibration motor (https://www.adafruit.com/product/1201)
7. Motor Driver (https://www.adafruit.com/product/2305)
8. Voltage Regulator (https://www.adafruit.com/product/2164)

Software:

1. Ubuntu Mate (https://ubuntu-mate.org/raspberry-pi/)

2. Librealsense SDK (https://github.com/IntelRealSense/librealsense)

3. WiringPi (http://wiringpi.com/)

4. OpenCV (https://opencv.org/)

5. SolidWorks (https://www.solidworks.com/)

Appendix B

Source Code

```
// Main driver for camera to motor control

// This method will need to do the following:
// 1. Take in the depth data in whatever format it comes in
// 2. Narrows the view to just the relevant pixels
// 3. Check if there is an object within those pixels that is within a threshold
// 4. Activate motor as an inverse relation to the distance that the object is
    away from us

#include <wiringPi.h>
#include <softPwm.h>
#include <iostream>
#include <librealsense2/rs.hpp>
#include <signal.h>
#include <stdio.h>
#include <cstdlib>
#include <unistd.h>
#include <cmath>

#include <map>
```

```
#include <queue>
#include <list>

#include <algorithm>

#include <opencv2/opencv.hpp>

using namespace cv;

float get_depth_scale(rs2::device dev);
float object_within_depth(const rs2::depth_frame& depth_frame, int x_bot,
    int x_top, int y_bot, int y_top, float depth_scale, float clipping_dist);
float cluster_within_depth(const rs2::depth_frame& depth_frame, int x_bot,
    int x_top, int y_bot, int y_top, float depth_scale, float clipping_dist);
std::list<std::pair<int, int>> find_neighbors(const int& x, const int& y);
std::list<std::pair<int, int>> find_neighbors_in_frame(const int& x, const
    int& y, const int& x_bot, const int& x_top, const int& y_bot, const int&
     y_top);
int get_vibration_power(const float& distance_of_object, const float&
    clipping_dist, const int& frame_count);
bool profile_changed(const std::vector<rs2::stream_profile>& current, const std::
    vector<rs2::stream_profile>& prev);



void my_handler(int s) {
     printf("Caught signal %d\n", s);
    softPwmWrite(13, 0);
    delay(100);
     exit(1);
}
```

```
int main(int argc, char * argv[]) try
{
    rs2::log_to_console(RS2_LOG_SEVERITY_ERROR);

    struct sigaction sigIntHandler;

    sigIntHandler.sa_handler = my_handler;
    sigemptyset(&sigIntHandler.sa_mask);
    sigIntHandler.sa_flags = 0;
    sigaction(SIGINT, &sigIntHandler, NULL);

    //window app(1280, 720, "CPP - Align Example"); // Simple window
        handling
    //ImGui_ImplGlfw_Init(app, false); // ImGui library initialization
    //rs2::colorizer c; // Helper to colorize depth images
    //texture renderer; // Helper for rendering images

    // Declare RealSense pipeline, encapsulating the actual device and sensors
    rs2::pipeline pipe;
    // Start streaming with default recommended configuration
    rs2::pipeline_profile profile = pipe.start();
    float depth_scale = get_depth_scale(profile.get_device());

    // Define decimation filter (reduces depth frame density)
    rs2::decimation_filter dec;

    // Define spatial filter (edge-preserving)
    rs2::spatial_filter spat;
```

```
// Configure filter parameters
dec.set_option(RS2_OPTION_FILTER_MAGNITUDE, 3);

// Enable hole-filling
// Hole filling is an aggressive heuristic and it gets the depth wrong many
    times
// (the shortest-path will always prefer to "cut" through the holes since they
     have 0 3D dist

spat.set_option(RS2_OPTION_HOLES_FILL, 2); // 5 = fill all the zero
    pixels
// Initialize wiringPi setup
wiringPiSetupGpio();
// Initialize softPwm on GPIO pin 13 with a range of 0 (off) to 100 (fully on
    )
softPwmCreate(13, 0, 100);

// Define a variable for controlling the distance to clip
float depth_clipping_distance = 2.5f;
float three_depth_clipping_distance = 0.75 * depth_clipping_distance;
float two_depth_clipping_distance = 0.50 * depth_clipping_distance;
float one_depth_clipping_distance = 0.25 * depth_clipping_distance;

//Code vibrations to indicate the start of program
softPwmWrite(13, 100);
usleep(500000);
softPwmWrite(13, 0);
usleep(500000);

softPwmWrite(13, 100);
```

```
usleep(500000);
softPwmWrite(13, 0);
usleep(500000);

int frame_count = 0;
float app_filler = 1.f;
while(app_filler) // Application still alive?
{
    rs2::frameset data = pipe.wait_for_frames(); // Wait for next set of
        frames from the camera
    rs2::depth_frame depth = data.get_depth_frame();

    data = data.apply_filter(dec);
    data = data.apply_filter(spat);

    //Check to see if the device has been changed
    if(profile_changed(pipe.get_active_profile().get_streams(), profile.
        get_streams()))
    {
        //If the profile was changed, get the new device and depth_scale
        profile = pipe.get_active_profile();
        depth_scale = get_depth_scale(profile.get_device());
    }

    // Narrowing of the frame: pick a top left corner and bottom right corner
        to look at
    // Will come into play later when we implement geometry algorithm better
    int top_left_x = 0;
    int top_left_y = 0;
    int bot_right_x = 100;
```

```
int bot_right_y = 100;

// Check to see if there are pixels below a certain threshold, store in a
    boolean
// Get the width and height of the depth frame
// Narrow the frame based on some proportion of the frame
int h = depth.get_height();
int w = depth.get_width();
int y_bot = static_cast<int>(4.5*h/9.0); // left
int y_top = static_cast<int>(8.0*h/9.0); // right
int x_bot = static_cast<int>(4.5*w/9.0); // down
int x_top = static_cast<int>(9.0*w/9.0); // up
float b_1 = object_within_depth(depth, x_bot, x_top, y_bot, y_top,
    depth_scale, depth_clipping_distance);
float b_2 = cluster_within_depth(depth, x_bot, x_top, y_bot, y_top,
    depth_scale, depth_clipping_distance);

bool b = false;

/*
for (int x = top_left_x; x < bot_right_x; x++)
{
    for (int y = top_left_y; y < bot_right_y; y++)
    {
        //std::cout << "Looking at the following points" << std::endl;
        //std::cout << x << std::endl;
        //std::cout << y << std::endl;
        dist = depth.get_distance(x, y) * depth_scale;
        if (dist < depth_clipping_distance)
        {
```

```
            //std::cout << "The distance that is within clipping distance
                is" << std::endl;
            //std::cout << dist << std::endl;
            b = true;
            //break;
        }
        if (dist < min_dist)
        {
            min_dist = dist;
        }
        if (dist > max_dist)
        {
            max_dist = dist;
        }
    }
}
*/

// If there are pixels below a certain threshold, make motor vibrate with
    inverse relation to that distance
int power = 0;
// If the boolean is true, then activate the vibration motor
if (b_2 > 0.0)
{
    // Set power to be a value from 50-100 depending on the distance
        away
    // Set the frequency of the vibration to be different as well
    // Extract this to a helper method
    bool vib_flag = true;
    /*
```

```
    if (( b_1 > three_depth_clipping_distance) && (frame_count%8 ==
        0))
    {
        vib_flag = true;
    } else if ((b_1 > two_depth_clipping_distance) && (frame_count
        %4 == 0))
    {
        vib_flag = true;
    } else if ((b_1 > one_depth_clipping_distance) && (frame_count
        %2 == 0))
    {
        vib_flag = true;
    } else if (b_1 <= one_depth_clipping_distance)
    {
        vib_flag = true;
    }
    */

    // Set power of vibration to be a function of distance of closest
        object
    if (vib_flag) {
        //power = static_cast<int>(30+70.0*(depth_clipping_distance-
            b_1)/depth_clipping_distance);
        power = 100;
    }
    std::cout <<"vib_flag: " << vib_flag << ", power: " << power <<
        ", dist: " << b_1 << ", frame: " << frame_count << std::endl;
    softPwmWrite(13, power);
} else
{
```

```
            softPwmWrite(13, power);
            std::cout << "Not vibrating, no objects detected! " << "frame: " <<
                frame_count << std::endl;
        }

        // Update frame count (used for frequency of vibrations)
        if (frame_count > 89)
        {
            frame_count = 0;
        } else
        {
            frame_count = frame_count + 1;
        }
    }
}
catch (const rs2::error & e)
{
    std::cerr << "RealSense error callin " << e.get_failed_function() << "("
        << e.get_failed_args() << "):\n " << e.what() << std::endl;
    return EXIT_FAILURE;
}
catch (const std::exception& e)
{
    std::cerr << e.what() << std::endl;
    return EXIT_FAILURE;
}

float get_depth_scale(rs2::device dev)
{
    // Go over the device's sensors
```

```
    for (rs2::sensor& sensor : dev.query_sensors())
    {
        // Check if the sensor is a depth sensor
        if (rs2::depth_sensor dpt = sensor.as<rs2::depth_sensor>())
        {
            return dpt.get_depth_scale();
        }
    }
    throw std::runtime_error("Device does not have a depth sensor");
}

float object_within_depth(const rs2::depth_frame& depth_frame, int x_bot,
    int x_top, int y_bot, int y_top, float depth_scale, float clipping_dist)
{
    const uint16_t* p_depth_frame = reinterpret_cast<const uint16_t*>(
        depth_frame.get_data());

    int width = depth_frame.get_width();
    int height = depth_frame.get_height();

    int total_pixels = (x_top-x_bot) * (y_top-y_bot);
    int pixel_count_threshold = static_cast<int>(total_pixels/8.0);
    int b_count = 0;
    float min_dist = clipping_dist;
    #pragma omp parallel for schedule(dynamic) //Using OpenMP to try to
        parallelize the loop
    for (int y = y_bot; y < y_top; y++)
    {
        auto depth_pixel_index = y*width;
        for (int x = x_bot; x<x_top; x++, ++depth_pixel_index)
```

```
        {
            // Get the depth value of the current pixel
            auto pixels_distance = depth_scale * p_depth_frame[
                depth_pixel_index];

            // Check if the depth value is less than the clipping threshold
            if ((pixels_distance < clipping_dist) && (pixels_distance > 0.f))
            {
                b_count = b_count+1;
                if (pixels_distance < min_dist)
                {
                    min_dist = pixels_distance;
                }
            }
        }
    }
    if (b_count > pixel_count_threshold)
    {
        return min_dist;
    }
    return 0.0;
}

bool profile_changed(const std::vector<rs2::stream_profile>& current, const std::
    vector<rs2::stream_profile>& prev)
{
    for (auto&& sp : prev)
    {
        //If previous profile is in current (maybe just added another)
```

```
        auto itr = std::find_if(std::begin(current), std::end(current), [&sp](
            const rs2::stream_profile& current_sp) { return sp.unique_id() ==
            current_sp.unique_id(); });
        if (itr == std::end(current)) // If previous stream wasn't found in
            current stream
        {
            return true;
        }
    }
    return false;
}

typedef std::pair<int, int> pair;

float cluster_within_depth(const rs2::depth_frame& depth_frame, int x_bot,
    int x_top, int y_bot, int y_top, float depth_scale, float clipping_dist) {
    const uint16_t* p_depth_frame = reinterpret_cast<const uint16_t*>(
        depth_frame.get_data());

    int width = depth_frame.get_width();
    int height = depth_frame.get_height();

    //std::cout << "depth_frame width for cluster finding: " << width << std::
        endl;
    //std::cout << "depth_frame height for cluster finding: " << height << std::
        endl;

    int total_pixels = (x_top-x_bot) * (y_top-y_bot);
    int pixel_count_threshold = static_cast<int>(total_pixels/4.0);
```

```
int b_count = 0;
float min_dist = clipping_dist;

// collect all pixels within a clipping distance by their indices
// put em in a map maybe? something with constant time access
// then as we iterate through the collection of pixels within
// a certain distance, we start each iteration assuming the single
// pixel we are looking at is within a "cluster" then we look at each
// of that pixel's neighbors to see if any are also within the distance
// and so on. The iteration continues if we find all the pixels in that
// "cluster" and it doesn't exceed the threshold
// if it does exceed threshold, then keep track of min distance of pixel
// Determine whether we want the algorithm to go through all clusters

#pragma omp parallel for schedule(dynamic) //Using OpenMP to try to
    parallelize the loop
//  Add all pixels within clipping distance to a map

// Map for y_bot => left
//  y_top => right
//  x_bot => down
//  x_top => up

// y is height
// x is width

std::map<pair, int> coordmap;
for (int y = y_bot; y < y_top; y++)
{
    auto depth_pixel_index = y*width;
```

```
    for (int x = x_bot; x<x_top; x++)
    {
        // Get the depth value of the current pixel
        auto pixels_distance = depth_scale * p_depth_frame[(
            depth_pixel_index+x)];



        // Check if the depth value is less than the clipping threshold
        if ((pixels_distance < clipping_dist) && (pixels_distance > 0.f))
        {
            // Hypoteneuse * sine((y-(height/2.0))/(height/2.0))
            float right_offset_estimate = pixels_distance * sin((y-(height
                /2.0))/(height/2.0));
            if (right_offset_estimate < 0.85)
            {
                // Add all pixels here to a map
                b_count = b_count+1;
                if (pixels_distance < min_dist)
                {
                    min_dist = pixels_distance;
                }
                coordmap.insert(std::make_pair(std::make_pair(x, y),1));
                if (b_count > 12000) {
                    return 1.0;
                }
            }
        }
    }
}
```

```
// Loop through them in a BFS way to find clusters, while tracking average
    distance
// For this, keep a map of visited coordinates
// While there are coordinates in the map of pixels within a certain distance
// while
int max_clust = 0;
//std::cout << "max possible coordmap.size() is " << (y_top-y_bot)*(x_top
    -x_bot) << std::endl;
std::cout << "coordmap.size() is " << coordmap.size() << std::endl;

float upper_clust_size_thresh = 0.1f;
int pixels_in_frame = (y_top-y_bot)*(x_top-x_bot); //About 100000
    currently
//std::cout << "here's threshold: " << upper_clust_size_thresh *
    pixels_in_frame << std::endl;
int clust_threshold = 250;
std::map<pair, int> visited;

//std::cout << "here's x_bot, x_top in cluster within "  << x_bot << ", "
    << x_top << std::endl;
//std::cout << "here's y_bot, y_top in cluster within "  << y_bot << ", "
    << y_top << std::endl;

for ( const auto& myPair : coordmap) {
    if (visited.find(myPair.first) == visited.end() ) {
        // not found, check neighbors and add to queue, keeping track of a
            list of these coords. compare size of list to max_clust at end
        auto key = myPair.first;
        // do bfs here on neighbors
```

```
// initialize queue here
std::queue<std::pair<int, int>> q;
// while queue not empty
q.push(key);
int clust_size = 0;
while(q.size() > 0) {
    // find way to obtain the neighbors
    std:: list <std::pair<int, int>> neighbors =
        find_neighbors_in_frame(q.front().first, q.front().second,
        x_bot, x_top, y_bot, y_top);
    // check if neighbors are in coordmap keys
    for (const auto& neighCoords : neighbors) {

        // if so, add them to queue, increment cluster size by 1, add
            to visited
        if((coordmap.find(neighCoords) != coordmap.end()) && (
            visited.find(neighCoords) == visited.end())) {
            q.push(neighCoords);
            clust_size++;
            // Should replace clust_threshold if a new thresh is
                needed
            if (clust_size > clust_threshold) {
                //std::cout << clust_size << " is greater than " <<
                    upper_clust_size_thresh*pixels_in_frame << "
                    with boolean " << (clust_size >
                    upper_clust_size_thresh*pixels_in_frame) <<std
                    ::endl;
                std::cout << "max_clust is reached " << clust_size
                    << std::endl;
                return 1.0;
```

```
                    }
                    visited.insert(std::make_pair(neighCoords, 1));
                }
            }
            // pop off queue
            q.pop();
        }
        // compare cluster size to max_clust
        if (clust_size > max_clust) {
            max_clust = clust_size;
        }
        // update visited
        visited.insert(std::make_pair(key, 1));
    } else {
        // found, continue
        continue;
    }
}

// Check if the largest cluster is above a certain pixel threshold (for now,
    just print the number of pixels in the largest cluster)

std::cout << "max_clust is " << max_clust << std::endl;
//  If it is, return the average distance of pixels in that cluster
if (max_clust > clust_threshold)
{
    return 1.0;
}
return 0.0;
}
```

```
std::list<std::pair<int, int>> find_neighbors(const int& x, const int& y) {
    // Helper method for BFS to find neighbors
    std::list<std::pair<int, int>> l;
    for (int i = x-1; i <= x+1; i++) {
        for (int j = y-1; j <= y+1; j++) {
            if ((i == x) && (j==y)) {
                continue;
            } else {
                l.push_back(std::make_pair(i, j));
            }
        }
    }
    return l;
}

std::list<std::pair<int, int>> find_neighbors_in_frame(const int& x, const
    int& y, const int& x_bot, const int& x_top, const int& y_bot, const int&
     y_top) {
    // Helper method for BFS to find neighbors within a box
    std::list<std::pair<int, int>> l;
    for (int i = x-1; i <= x+1; i++) {
        for (int j = y-1; j <= y+1; j++) {
            if (((i == x) && (j==y)) || (i < x_bot) || (i > x_top) || (j < y_bot
                ) || (j > y_top)) {
                continue;
            } else {
                l.push_back(std::make_pair(i, j));
            }
        }
```

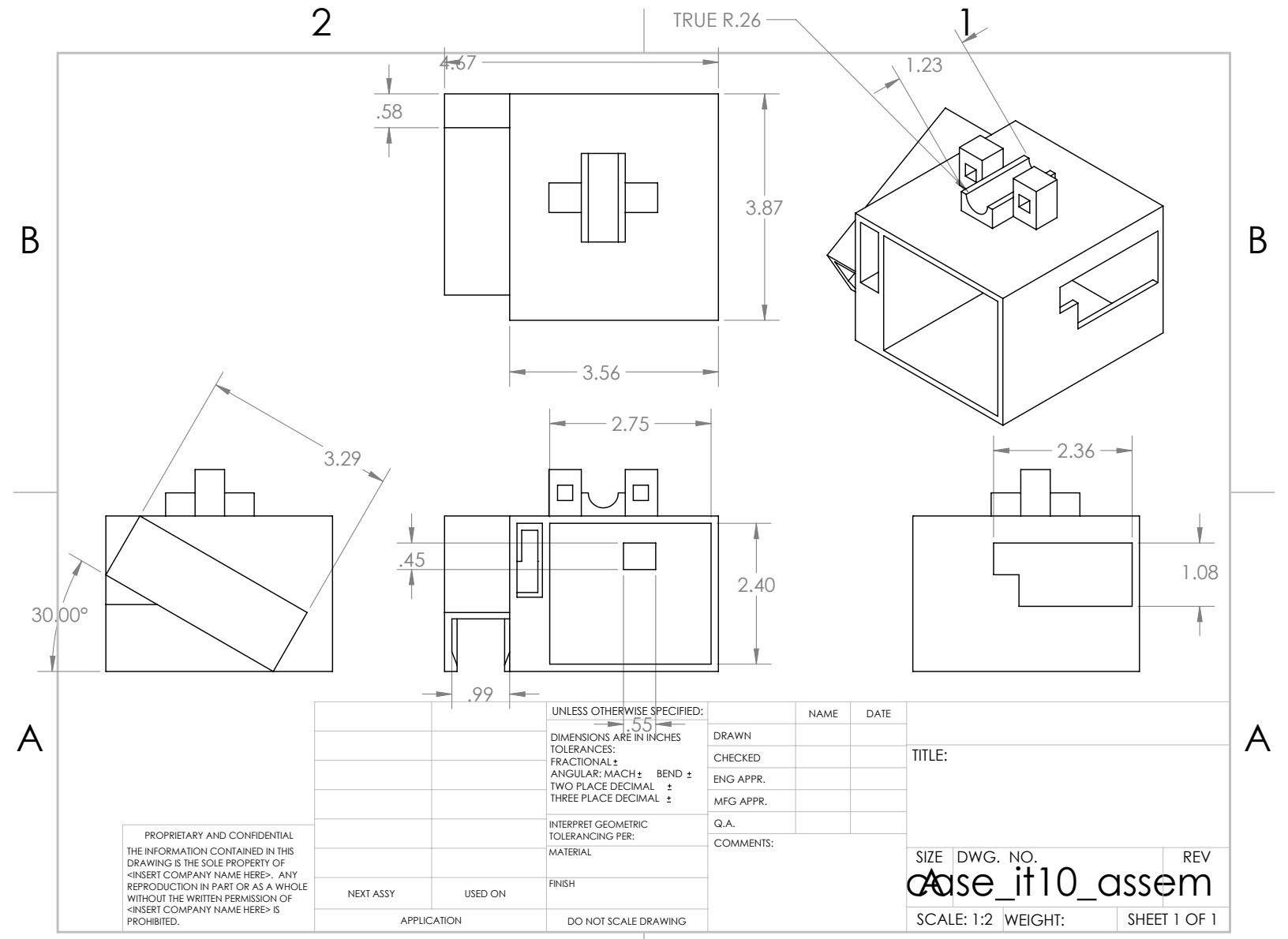
```
    }
    return l;
}

int get_vibration_power(const float& distance_of_object, const float&
    clipping_dist, const int& frame_count) {
    //Method that retrieves the power of vibration
    // Argument distance_of_object will be >0 if there's an object within
        threshold
    // we have defined above. Otherwise it will be 0.0.
    int power = 0;
    bool vib_flag = false;
    // If the boolean is true, then activate the vibration motor
    if (distance_of_object > 0.0)
    {
        // Set the frequency of the vibration to be different as well
        vib_flag = true;
        /*
        if (( b_1 > three_depth_clipping_distance) && (frame_count%8 == 0))
        {
            vib_flag = true;
        } else if ((b_1 > two_depth_clipping_distance) && (frame_count%4
            == 0))
        {
            vib_flag = true;
        } else if ((b_1 > one_depth_clipping_distance) && (frame_count%2
            == 0))
        {
            vib_flag = true;
        } else if (b_1 <= one_depth_clipping_distance)
```

```
        {
            vib_flag = true;
        }
        */
    }
    //std::cout << "distance_of_object: " << distance_of_object << " vib_flag:
        " << vib_flag;
    // Set power to be a value from 50-100 as a function of
    //  distance of closest object
    // This function can be changed to whatever else is deemed to fit
    if (vib_flag) {
        //power = static_cast<int>(30+70.0*(clipping_dist-distance_of_object)/
            clipping_dist);
        power = 100;
    }
    return power;
}
```

Appendix C

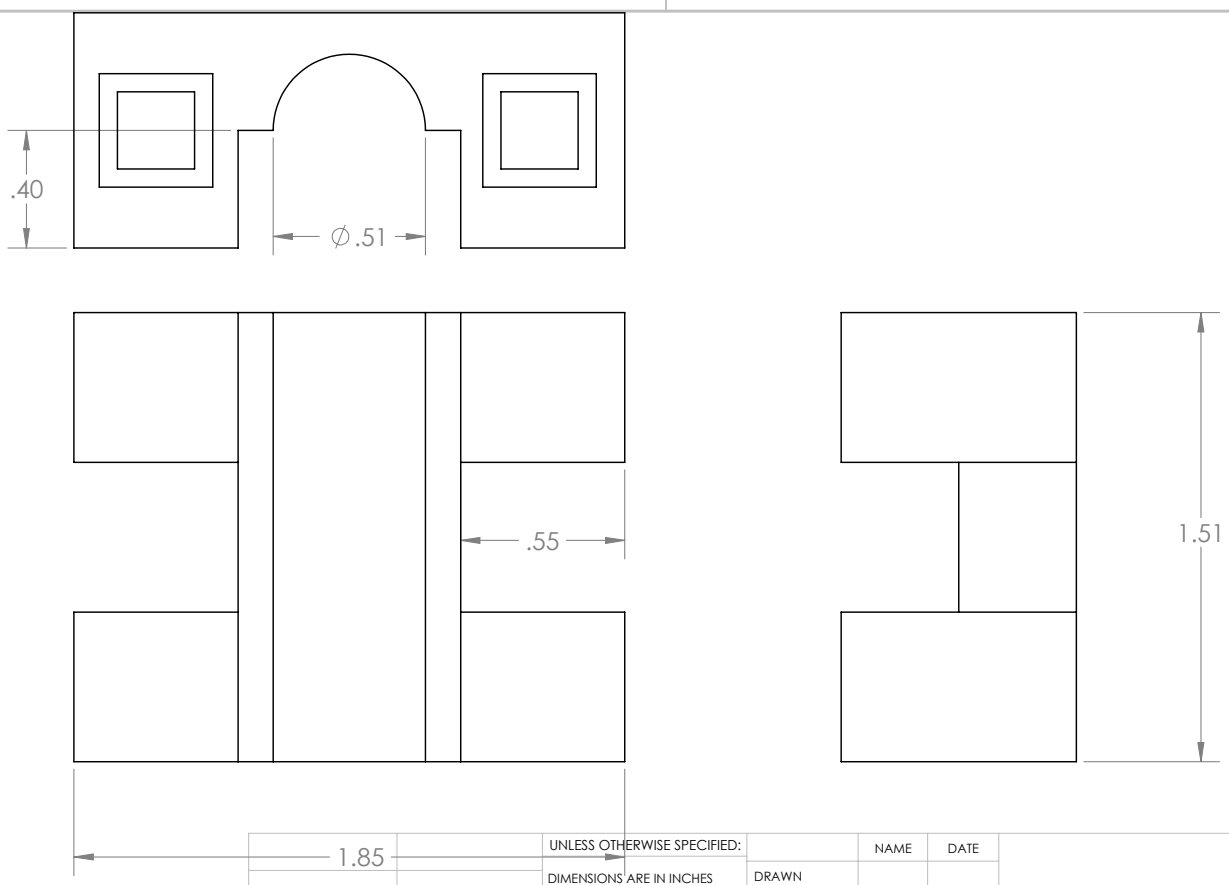
SolidWorks Case Drawings



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

SIZE	DWG. NO.	REV
A	case_it10_assem	
SCALE: 1:2	WEIGHT:	SHEET 1 OF 1



		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE:

SIZE	DWG. NO.	REV
A	clamp_top1_it4	
SCALE: 2:1	WEIGHT:	SHEET 1 OF 1

Bibliography

[1] Dragan Ahmetovic, Cole Gleason, Chengxiong Ruan, Kris M. Kitani, Hironobu Takagi, and Chieko Asakawa. Navcog: A navigational cognitive assistant for the blind. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '16)*, pages 90 – 99, September 2016.

[2] Tomohiro Amemiya, Jun Yamashita, Koichi Hirota, and Michitaka Hirose. Virtual leading blocks for the deaf-blind: A real-time way-finder by verbal-nonverbal hybrid interface and high-density rfid tag space. In *Proceedings of the IEEE Virtual Reality 2004*, VR '04, pages 165–, Washington, DC, USA, 2004. IEEE Computer Society.

[3] T. Ando, R. Tsukahara, M. Seki, and M. G. Fujie. A haptic interface âĂIJforce blinker 2âĂİ for navigation of the visually impaired. *IEEE Transactions on Industrial Electronics*, 59(11):4112–4119, Nov 2012.

[4] M. Asonuma, M. Matsumoto, and C. Wada. Basic study on obstacle avoidance system for the visually impaired. In *SICE 2004 Annual Conference*, volume 3, pages 2411–2415 vol. 3, Aug 2004.

[5] S. Bhatlawande, M. Mahadevappa, J. Mukherjee, M. Biswas, D. Das, and S. Gupta. Design, development, and clinical evaluation of the electronic mobility cane for vision rehabilitation. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 22(6):1148–1159, Nov 2014.

[6] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. *CoRR*, abs/1611.08050, 2016.

[7] S. Choi and K. J. Kuchenbecker. Vibrotactile display: Perception, technology, and applications. *Proceedings of the IEEE*, 101(9):2093–2104, Sep. 2013.

[8] Sakmongkon Chumkamon, Peranitti Tuvaphanthaphiphat, and Phongsak Keeratiwintakorn. A blind navigation system using rfid for indoor environments. pages 765 – 768, 06 2008.

[9] A. Cosgun, E. A. Sisbot, and H. I. Christensen. Guidance for human navigation using a vibro-tactile belt interface and robot-like motion planning. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6350–6355, May 2014.

[10] Karen Duarte, JosÃľ CecÃŋlio, and Pedro Furtado. Easily guiding of blind: Providing information and navigation - smartnav. volume 146, pages 129–134, 11 2014.

[11] Jan B. F. van Erp. Tactile navigation display. In *Proceedings of the First International Workshop on Haptic Human-Computer Interaction*, pages 165–173, Berlin, Heidelberg, 2001. Springer-Verlag.

[12] Navid Fallah, Ilias Apostolopoulos, Kostas Bekris, and Eelke Folmer. The user as a sensor: Navigating users with visual impairments in indoor spaces using tactile landmarks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 425–432, New York, NY, USA, 2012. ACM.

[13] Jolyon J. Faria, Sérgio Lopes, Hugo Fernandes, Pedro Martins, and João Barroso. Electronic white cane for blind people navigation assistance. *2010 World Automation Congress*, pages 1–7, 2010.

[14] VÃŋtor Filipe, Filipe Fernandes, Hugo Fernandes, AntÃşnio Sousa, Hugo Paredes, and Joao Barroso. Blind navigation support system based on microsoft kinect. *Procedia Computer Science*, 14:94 – 101, 12 2012.

[15] T. Gallagher, E. Wise, B. Li, A. G. Dempster, C. Rizos, and E. Ramsey-Stewart. Indoor positioning system based on sensor fusion for the blind and visually impaired. In *2012 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–9, Nov 2012.

[16] R.G. Golledge, James Marston, and C.M. Costanzo. Attitudes of visually impaired persons toward the use of public transportation. *Journal of visual impairment blindness*, 91:446–459, 09 1997.

[17] Luz H Villamizar, Mauricio Gualdron, Fabio Gonzalez, Juan Aceros, and Carlos Rizzo-Sierra. A necklace sonar with adjustable scope range for assisting the visually impaired. *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, 2013:1450–1453, 07 2013.

[18] Joel Hesch and Stergios I. Roumeliotis. An indoor localization aid for the visually impaired. pages 3545 – 3551, 05 2007.

[19] S. Hoefer. Meet the tacit project. it's sonar for the blind, personal communication, 2011.

[20] Brian Hoyle and Dean Waters. Mobility at: The batcane (ultracane). 01 2008.

[21] Hsieh-Chang Huang, Ching-Tang Hsieh, and Cheng-Hsiang Yeh. An indoor obstacle detection system using depth information and region growth. *Sensors (Basel, Switzerland)*, 15:27116–27141, 10 2015.

[22] Andreas Hub, Joachim Diepstraten, and Thomas Ertl. Design and development of an indoor navigation and object identification system for the blind. *SIGACCESS Access. Comput.*, (77-78):147–152, September 2003.

[23] Kiyohide Ito, Makoto Okamoto, Junichi Akita, Tetsuo Ono, Ikuko Gyobu, Tomohito Takagi, Takahiro Hoshi, and Yu Mishima. Cyarm: An alternative aid device for blind persons. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '05, pages 1483–1488, New York, NY, USA, 2005. ACM.

[24] Currie K Clark AF Kanwal N, Bostanci E. A navigation system for the visually impaired: A fusion of vision and depth sensor. Technical report, 2015.

[25] Jee-Eun Kim, Masahiro Bessho, Shinsuke Kobayashi, Noboru Koshizuka, and Ken Sakamura. Navigating visually impaired travelers in a large train station using smartphone and bluetooth low energy. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, pages 604–611, New York, NY, USA, 2016. ACM.

[26] Vladimir Kulyukin, Chaitanya Gharpure, John Nicholson, and Grayson Osborne. Robot-assisted wayfinding for the visually impaired in structured indoor environments. *Autonomous Robots*, 21:29–41, 01 2006.

[27] Amy R. Lam. Vibrotactile pattern recognition on the torso with one and two dimensional displays. 2007.

[28] Bing Li, J. Pablo Muñoz, Xuejian Rong, Jizhong Xiao, Yingli Tian, and Aries Arditi. Isana: Wearable context-aware indoor assistive navigation with obstacle avoidance for the blind. In *ECCV Workshops*, 2016.

[29] Shachar Maidenbaum, Shlomi Hanassy, Sami Abboud, Galit Buchs, Daniel-Robert Chebat, Shelly Levy-Tzedek, and Amir Amedi. The 'eyecane', a new electronic travel aid for the blind: Technology, behavior swift learning. *Restorative neurology and neuroscience*, 32, 09 2014.

[30] Roberto Manduchi. Mobile vision as assistive technology for the blind: An experimental study. In *Proceedings of the 13th International Conference on Computers Helping People with Special Needs - Volume Part II*, ICCHP'12, pages 9–16, Berlin, Heidelberg, 2012. Springer-Verlag.

[31] Roberto Manduchi, Sri Kurniawan, and Homayoun Bagherinia. Blind guidance using mobile computer vision: A usability study. In *Proceedings of the 12th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '10, pages 241–242, New York, NY, USA, 2010. ACM.

[32] M. Nakajima and S. Haruyama. Indoor navigation system for visually impaired people using visible light communication and compensated geomagnetic sensing. In *2012 1st IEEE International Conference on Communications in China (ICCC)*, pages 524–529, Aug 2012.

[33] World Health Organization. Global data on visual impairments.

[34] William Penrod, Michael D. Corbett, and Bruce Blasch. A master trainer class for professionals in teaching the ultracane electronic travel device. *Journal of Visual Impairment Blindness*, 99, 01 2005.

[35] Helen Petrie, Valerie Johnson, Thomas Strothotte, Andreas Raab, Steffi Fritz, and Rainer Michel. Mobic: Designing a travel aid for blind and elderly people. *Journal of Navigation*, 49:45 – 52, 01 1996.

[36] Huy-Hieu Pham, Thi Le, and Nicolas Vuillerme. Real-time obstacle detection system in indoor environment for the visually impaired using microsoft kinect sensor. *Journal of Sensors*, 2016:1–13, 01 2016.

[37] Shrinivas Pundlik, Eli Peli, and Gang Luo. Time to collision and collision risk estimation from local scale and motion. pages 728–737, 09 2011.

[38] Shrinivas Pundlik, Matteo Tomasi, and Gang Luo. Evaluation of a portable collision warning device for patients with peripheral vision loss in an obstacle course. *Investigative ophthalmology visual science*, 56, 03 2015.

[39] Lisa Ran, Sumi Helal, and Steve Moore. Drishti: An integrated indoor/outdoor blind navigation system and service. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, PERCOM '04, pages 23–, Washington, DC, USA, 2004. IEEE Computer Society.

[40] Shraga Shoval, Johann Borenstein, and Yoram Koren. The navbelt-a computerized travel aid for the blind based on mobile robotics technology. *IEEE Transactions on Biomedical Engineering*, 45:1376–1386, 1998.

[41] Vaibhav Singh, Rohan Paul, D O Vishaal Mehra, Anuraag Gupta, Vasu Dev Sharma, Saumya Jain, Chinmay Agarwal, Ankush Garg, Sandeep Singh Gujral, Mahesh Balakrishnan, Kolin Paul, P. V. M. Rao, and Dipendra Manocha. 'smart' cane for the visually impaired: Design and controlled field testing of an affordable obstacle detection system. 2010.

[42] Koji Tsukada and Michiaki Yasumura. Activebelt: Belt-type wearable tactile display for directional navigation. In *UbiComp*, 2004.

[43] Y. Tung and K. G. Shin. Use of phone sensors to enhance distracted pedestriansâĂŹ safety. *IEEE Transactions on Mobile Computing*, 17(6):1469–1482, June 2018.

[44] Hendrik-Jan Veen and Jan B F Van Erp. Providing directional information with tactile torso displays. *Proceedings of EuroHaptics*, 01 2003.

[45] Ramiro Velazquez. Wearable assistive devices for the blind. *CoRR*, abs/1611.09480, 2016.

[46] H. Wang, R. K. Katzschmann, S. Teng, B. Araki, L. GiarrÃľ, and D. Rus. Enabling independent navigation for visually impaired people through a wearable vision-based feedback system. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6533–6540, May 2017.

[47] C. Ye, S. Hong, X. Qian, and W. Wu. Co-robotic cane: A new robotic navigation aid for the visually impaired. *IEEE Systems, Man, and Cybernetics Magazine*, 2(2):33–42, April 2016.