Visually Assistive Hat: A Wearable Device for the Visually Impaired

A Technical Report submitted to the Department of Electrical and Computer Engineering

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Mary DeSimone

Spring, 2022

Technical Project Team Members
LaDawna McEnheimer
Gabriel Morales
Hafsah Shamsie

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Advisor

Harry Powell, Department of Electrical and Computer Engineering

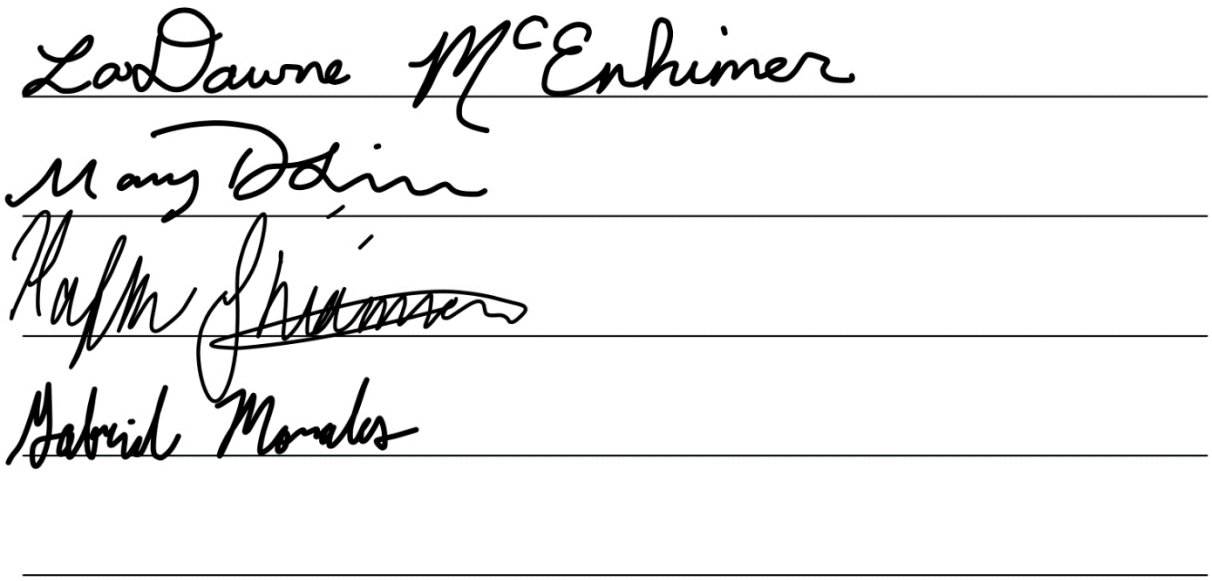
Table In The Back + 1 – Visual Assistant Hat

Mary DeSimone, LaDawna McEnhimer, Gabriel Morales, and Hafsah Shamsie

12/17/2021

Capstone Design ECE 4440 / ECE4991

Signatures



Statement of work:

Mary DeSimone

My primary role was co-handling the assembly of the hat with Hafsah and handling the construction and angling of the sensors to ensure proper readings. This involved sewing components, such as all sensors, motors, and piezo buzzers onto the hat. Additionally, a lot of assembly testing was performed to test the functionality of the device, including full system tests and motor and piezo buzzer tests. I also focused on testing the hat with the user in mind by ensuring the everyday functionality of the hat in terms of user comfort. Hafsah and I also codesigned the selection and calculations of the power system, in which I most specifically focused on the energy consumption of the various components in our design.

Additionally, when the boards arrived, I took them to WWW Electronics Inc. (3W) to have the components soldered on, and handled communication with them for the various problems that arose. After receiving the boards back, I soldered on additional test points and extended the wires from the motors, sensors, buzzers, battery wires, and button. Finally, I assisted LaDawna in the construction of the button task.

LaDawna McEnhimer

My primary role in the project was programming pulse-width modulation (PWM) interrupts for the piezo buzzers and vibration motors. In the first stages of the project, I designed the preliminary expected layout of the hat, with considerations to anticipated risks, the user experience, and space constraints. These specifications informed the possible battery sizes and weights, as well as the type of hat required to have enough space to mount the sensors and house the microcontroller unit (MCU) and battery. I also helped write the hardware test plan and execute the software test plans.

Following the layout design, I programmed the piezo buzzers and vibration motors to trigger using PWM interrupts through the driver library, and later using timer interrupts and toggling the pins to reduce the number of interrupts occurring. I also co-created the button task with Mary. I helped manage the GitHub repository and worked closely with Ricky to integrate the code and later debug issues that arose within the entire codebase, including the PWM interrupts, pin toggling, the finite state machine (FSM) timing, and inter-Integrated Circuit (I^2C) and universal asynchronous receiver-transmitter (UART) protocols. I helped with experimental testing to determine required distance thresholds, sensor positioning, and debug the code.

Gabriel Morales

My main focus for this project was the computer-aided design (CAD) of the circuits, printed circuit board (PCB), and getting communication working between the ultrasonic and light detection and ranging (LiDAR) sensors. In the first part of the project, I researched candidates for our sensors and MCU launchpad selection. After finding the correct LiDAR and ultrasonic sensors, I chose an MSP430 that would have all the correct number of modules (I^2C and UART) to communicate with each sensor. While my team researched for an adequate power

source, I designed the circuitry and PCB that would mount onto our MSP430FR2433 to provide power to our sensors and allow communication between them [1]. Following the design, I also tested the PCB with my teammates to ensure the correct voltages were created by our regulators, metal oxide semiconductor field effect transistors (MOSFET), and that the correct connections were made to our mixed signal processor (MSP) pins.

Following the hardware testing, I took the lead on programming the I^2C and UART communications to send and receive data from our sensors. Once these communication protocols were working, I collaborated with LaDawna on getting our task scheduler operating and timing the communication with the other tasks such as PWM for our piezo buzzers and outputs for our vibrating motors. Additionally, I made modifications to our wiring on the PCB or to our sensors whenever we encountered issues mounting our PCB to the MSP. Lastly, I partnered with LaDawna to modify and debug the code while adjusting the parameters for the entire system to work.

Hafsah Shamsie

My primary role in the project was the physical construction of the hat, which was done jointly with Mary. This involved making modifications to the hat to improve user experience and comfort while wearing the hat, proper placing and mounting of the various components, including all sensors, buzzers, motors, battery, PCB and microcontroller. In addition to this, I worked a great deal on assembly testing in order to ensure that the alignment and angle of all sensors was accurate, and all the components fit comfortably on the user's head. Additionally, the utility of the hat was tested by continuously trying on the hat throughout development. My other main role in the project was the overall power system design, which was also done jointly with Mary. I more closely focused on candidate chips and batteries for our PCB.

As a secondary role, I assisted Ricky with the development of the I^2C communication protocol in order to setup data transfer between the microcontroller and LiDAR sensors through peer programming efforts. This involved searching through existing documentation on I^2C protocol and its specific implementation on our microcontroller as well as assisting with debugging errors with the code as they arose.

Table of Contents

Contents

Table of Figures

Abstract

This project is focused on creating a hat that provides tactile and auditory feedback to blind and visually impaired individuals based on obstruction detection. This was done by embedding LiDAR sensors into the hat and connecting them to a central MSP430FR2433 board to process current surroundings. The device will gather information regarding a user's surroundings from the front, sides, back, and any incline changes. Then, using the information provided by the LiDAR sensors, vibrating direct current (DC) motors can be programmed to vibrate whenever their respective sensor detects an object and increases the vibration frequency as the object gets closer to the individual. Additionally, small piezo buzzers will be mounted to the side of the cap and connected to the MCU to provide auditory feedback for users who may prefer to have auditory feedback. Lastly, the cap will have a sensor that provides feedback whenever the user is approaching steps or some type of change in surface elevation.

Background

Visual impairment, including blindness or simply any form of vision degradation, is an extremely widespread issue. According to the Centers for Disease Control and Prevention (CDC), in 2015 1.02 million people were blind and about 3.22 million people had some sort of vision impairment in the United States. However, by 2050, it is projected for both numbers to double to roughly 2.1 million and 6.95 million respectively as the population of the elderly increases [2]. Additionally, a total of 16.4 million other Americans are anticipated to face issues with seeing due to other degenerative eye diseases by this time [3]. Vision loss or impairment is considered a major disability and it hinders the impacted population's ability to carry out daily life tasks, more notably tasks involving ambulation [4]. This project's goal is to address the issue of assisting the visually impaired with obstacle avoidance and wayfinding when it comes to ambulatory-related tasks in their daily lives.

The design of sensor-based object detection systems for the visually impaired, particularly electronic travel aids (ETA's), devices that gather information about a user's surroundings through sensors, sonar, or lasers, then relay this information back to the user is not a completely novel idea [5]. One notable device is the development of the GuideCane at the University of Michigan. This technology mimics the typical blind cane, but instead is equipped with ultrasonic sensors, servo motors, and wheels. As the cane detects obstructions in the user's path it uses the motors and wheels to guide the user in a path to avoid the obstruction [6]. Similarly, to the GuideCane is the Smart Cane, which also functions like typical blind canes, but it uses ultrasonic sensors to detect obstacles and uses speakers to relay notifications to the user. For user's who may be auditorily impaired, the device makes use of special gloves that provide different types of tactile feedback on each finger, each signifying a different message [7]. Another device is the Eye Substitution, which makes use of an MSP430 microcontroller and ultrasonic sensors to create a hand-held obstacle detecting device. Another similar prototype is the Path Force Feedback Belt, which uses video cameras placed around one's waist to build 3D images of the user's surroundings [5]. Finally, one similar commercially available product is the iGlasses Ultrasonic Mobility Aid, which use ultrasonic sensors and gentle vibrations, with increasing frequencies as objects get nearer [8].

Although many existing devices currently exist, many of them have notable flaws. Both the described cane devices and the iGlasses do not give the user any feedback on obstacles that may be on either of their sides or coming up from behind. The Path Force Feedback Belt attempts to give the user an idea of their surroundings from all sides, but the use of video cameras results in the detection range for this system being too small [5]. The canes and the Eye Substitution devices additionally require the user to always hold them in the correct orientation.

Thus, the Eye Substitution device is not ideal as it is hand-mounted, and this may interfere with other tasks. Regarding the GuideCane specifically, the fact the cane utilizes wheels poses a greater threat to the user's stability, especially when walking in an unfamiliar path. In terms of the Smart Cane, the audio feedback is given too far from the user's ear, as it comes from the cane itself, which leads to the risk of the user potentially not hearing the warnings. Further, the use of relaying a different vibration to each finger to specify different messages is overly complex and may lead to confusion and issues with the user discerning the difference between each vibration. Finally, the iGlasses also face the issue of not being able to detect drop-offs or inclines in paths [9].

Our project ultimately aimed to mitigate all these issues by creating a lightweight device the user can place on their head and not worry about holding. Our design provides the user with a fuller picture and awareness of their surroundings by monitoring obstructions from the user's side or behind. The combined use of LiDAR and ultrasonic sensors help create a more effective and powerful obstacle detection system, as opposed to the use of video cameras or simply ultrasonic sensors alone. Furthermore, the auditory and tactile feedback is straightforward to comprehend and is administered near the ear, rather than right next to it or in it, to relay the messages to the user in a way that will ensure they hear the message, without blocking out sounds of their surroundings [10].

The project made use of a variety of our previous coursework completed throughout our degrees. In order to program all software features, such as the task scheduler, communication protocols, button tasks, motor tasks, and buzzer tasks, which were primarily handled by Ricky and LaDawna, extensive knowledge of embedded programming and general computer architecture was required. All team members had gained this experience through the embedded course series (ECE 3501-3052) and were able to apply these skills in this domain. Furthermore, Ricky, whose primary focus was developing the UART and I^2C communication protocols between the sensors and microcontroller, had also had previous extensive embedded programming experience through work experience. Additionally, Mary and Hafsah completed the advanced embedded course this past semester (ECE 4501), which proved helpful when it came to understanding scheduling conflicts sues in the software.

In terms of power design, Hafsah and Mary made much use of the power concepts learned in the Fundamentals of Electrical and Computer Engineering III. Concepts that were taken from this class involved voltage regulator designs and applications as well as power consumption. For the CAD work, which was primarily done in the form of PCB design, Ricky relied heavily on the Fundamentals of Electrical and Computer Engineering course series (ECE 2630, ECE 2660, and ECE 3750). All team members gained experience with PCB design through all these classes. However, since the purpose of the PCB was to relay the proper power supplies to power all the sensors and associated hardware on the hat, Ricky made the most use of ECE 3750 and ECE 2660, particularly in the application of MOSFET use within an integrated circuit.

Constraints

Design Constraints

Since the Table in the Back +1 consisted entirely of computer engineers, the initial project constraints implemented by the Capstone program included an embedded central

processing unit (CPU), an interface to a device we had not used previously, and a component with industrial quality interconnects such as a PCB.

CPU limitations

The Texas Instruments MSP430FR2433 microcontroller [1] was chosen as the embedded CPU based on the required I^2C and UART modules required to communicate with each sensor. It is also compact enough to fit in the hat, while still having enough pins to run the required devices. A limitation of this CPU includes the inability to change the priority of the interrupts. The sensor protocols have a fixed lower priority relative to the timer interrupts, which introduced complications with the synchronous nature of I^2C communications.

Component Limitations

In choosing components within our budget, there were functionality tradeoffs. A limitation in the processing power of the CPU made it impossible to generate words through speakers, which is why piezo buzzers were chosen instead. This required sound-based warnings to be generated by a series of tones that could be interpreted ambiguously. Furthermore, experimentation showed that the LiDAR sensors being used are incapable of detecting glass obstructions. The ultrasonic sensors cannot detect objects from a distance as far away as the LiDAR sensors can. The ultrasonic sensors detect obstacles within a certain range of distance while the LiDAR sensors rely on pinpoint accuracy and can only detect objects directly in their path. Ultrasonic sensors generate interference with one another when they overlap, and as such it was necessary to position them opposite one another [11].

Software Availability

PCB board design was completed using National Instruments' Multisim [12] and Ultiboard [13], with access facilitated by the University of Virginia's (UVA) licenses. National Instruments' Virtual Bench [14] software was used to test code and generate power to the MSP for testing. Code Composer Studio [15] was freely available and used for writing the embedded code.

Manufacturing Limitations

The critical materials needed to complete our project include LiDAR sensors, ultrasonic sensors, piezo buzzers, vibration motors, batteries, PCB, and waterproof casings. All were available from Digi-Key or Mouser. Based on possible interference between sensors of the same type, the usage and positioning of the ultrasonic and LiDAR sensors were not interchangeable. A hat to mount the components on was purchased from Amazon.

Outside of the critical components of the board, supply chain issues required searching for alternate components suitable to our PCB design. Most notably the chosen P-Channel surface mount MOSFETs had an unanticipated on resistance that prevented the gate voltage from going high enough to trigger the LiDAR sensors to turn on. Suitable MOSFETs were not in stock, so power cycling was not possible. The small sizing of the vibration motor wire leads to complications in crimping and connecting the wires to the housings and required expert

assistance from 3W. Following construction, manually connecting and disconnecting these wires frequently led to the crimps loosening and breaking off within their housings and required repair and replacement. Additionally, the current PCB design is intended for a 40-pin MSP. The MSP430FR2433 only requires a 10-pin connector, and it was necessary to block off half of the available pins on the PCB header pins when connecting to the MSP [1].

Economic and Cost Constraints

The project budget was $500 per team, which mandated compromises in part selection. The largest expense to this project is the combined price of the sensors that need to be acquired because of their specialized purposes. The costs associated with the PCB, including the costs of printing the PCB, part acquisition, and populating the board, were our next highest individual cost. The goal would be to keep the costs as low as possible to reduce sale price and economic inequalities between users as products that improve quality of life should be widely available. Based on the market cost of approximately $100 held by the iGlasses Ultrasonic Mobility Aid [8], our initial price will be higher than other similar products. However, with scale it would be possible to lower our price and remain competitive.

Environmental Impact

When designing this product, it is important to keep in mind the environmental impact created by electrically powered devices. The primary environmental concerns come from the device's battery, and disposing of the electronics built into the sensors, piezo buzzers, and vibration motors. Rechargeable batteries are the primary power source of the device in order to extend its lifespan. Compared to standard batteries, these have a reduced impact on air and water pollution, as well as overall global warming; however, recycling these batteries can still have a negative impact on the battery. In cases where the energy being held is not fully depleted recycling these batteries can have a greater negative impact on the environment, especially when improperly disposed of [16]. Assuming proper disposal, further impact to the environment will be dependent upon the methods used to recycle the batteries, with the greatest reduction in impact caused by using low-temperature processes to maximize the amount of plastic recovered. Other cases such as using hydrometallurgical processes in recycling can lead to accelerated global warming, while simply throwing the batteries in the trash can result in higher levels of toxicity [17]. Overall, there are many tradeoffs that must be made when dealing with batteries, some of which will be offset by the use of rechargeable batteries.

Although it was difficult to find specific information about the individual electronic components, there is information about the general recycling of electronic components. Overall, the components were chosen for their functionality and durability, in the hopes that their small size and lack of moving pieces makes them less prone to being damaged and needing replacement. As a product with several parts, there is the risk of users being tempted to throw the entire device into the trash for convenience. This would be highly dangerous, as the electronic components would eventually begin to leak and release toxins into the environment. If the device is properly recycled, there is the potential for the recovery of some of the valuable and finite raw materials contained in the electronic materials. However, depending on the process being used to recycle these components there is the potential for the release of dangerous chemicals into the

environment, such as the emission of methane, which is known to damage the ozone, increasing the perils of global warming [18]. There is also risk to those charged with recycling the materials in their exposure to these toxins, particularly in less developed countries without stringent safety standards [19].

Sustainability

This device is made primarily from parts purchased from Digi-Key and Mouser. As these suppliers are well-established and there are no current discontinuation warnings on the parts selected, we expect them to be available in the future. However, with current supply chain issues as a result of the pandemic, this is subject to change. Alternative parts are available though, as many of the parts selected can be easily substituted. As wearable technology, there is a risk of rough use and dropping the device. As such, we attained components with fewer delicate parts. Additionally, the most critical parts for continued functionality of the product such as the PCB are going to be located within waterproof casings required by National Electrical Manufacturers Association (NEMA) standards, increasing the safety and longevity of the device [20].

We used a rechargeable power source for our product to extend the lifecycle of our device. For this reason, we want the battery to be easily accessible. The prototype requires the removal of the battery to charge using external alligator clips. As such, the current design allows for easy access to both the battery, MSP, and PCB board as they are all located in a contiguous space in the hat. However, it is most likely that damage would occur to the sensors or motors, which are currently sewn into the hat design. Extended use of the motors running at a high frequency can cause them to grow warm, and risk burning out. Future iterations would ideally have components that would be more easily interchangeable and wires that are significantly more durable, increasing repairability of the device. Based on the battery currently being used and without the implementation of power cycling for the LiDAR sensors, we expect the device to reliably last approximately six hours per use based on an experimental test. It takes approximately four hours to recharge the battery. The next iteration of the device would include MOSFETs capable of performing power cycling, and so the device would last longer and easily reach our predicted battery life of eight hours.

Health and Safety

As the intention of our project is to help someone overcome physical impairment, reliability of information and availability of the product are a priority. Any failure of the device could create a dangerous experience, especially if the failure is not immediately obvious and the user becomes unaware regarding their surroundings. Additionally, it can be bad for your eyes to have a lidar sensor pointed directly into them. As such, it will be necessary to acquire sensors rated using the IEC 602825-1 standard Class 1M, which is safe for the human eye. [21]

Other identified safety risks include a learning curve for identifying the exact location of the motor generating vibrations, as all are located on the same elastic band. As testing continued, differentiation between motors became easier, making this an obstacle more feasible to overcome by the user rather than redesign of layout. Other discovered safety risks include that LiDAR is incapable of detecting clear obstacles such as those made of glass. This is mitigated

slightly by the ultrasonic sensor's capability to detect glass, as it relies on sound-based rather than light-based obstacle detection. Finally, we were unable to acquire a chip that allows incorporation of low battery notifications to our device, which introduces a significant safety risk of the battery dying without warning while in use. As the user is fundamentally dependent on our device for safety, future iterations would have to incorporate this warning and reduce risk of failure. The possibility of the user angling the device correctly was also an initial concern. However, the weight of the battery making specific orientations more convenient and the usage of an elastic band to keep the hat secured on the head mitigates this issue.

External Standards

1. *IPC Standard for Electrical Safety* - These define the standards [22] for PCBs, with the part and track spacings defined as IPC-2221A and the solder mask, plating, material, and board edges of an acceptable PCB listed in IPC-A-600F. For reasons listed in 'Health and Safety,' our PCB must adhere to the class 3 standard where continuous functionality is critical, and these standards were incorporated into the PCB design.
2. *NEMA Standards for Mechanical Casings* – These standards [20] are defined for exposed electrical components. The electrical components of our project are at risk of encountering fabric, skin, or water. This requires the casing to meet a Type 4 weatherproof rating to protect against dirt, dust, and inclement weather.
3. *IEC Laser Standards* - These standards [21] are defined for devices including lasers, such as the LiDAR sensors used in our device. This project adheres to the IEC 602825-1 standard with a Class 1M rating to be safe for the human eye assuming non-prolonged exposure and was considered in selection and placement of LiDAR sensors.
4. *Embedded C Coding Standard* - The Michael Barr Embedded C Coding Standard [23] defines programming standards intended to reduce bugs in code, including guidelines for comments, whitespace, data types, procedures, variables, and statements. These were considered throughout writing the code and used to facilitate shared programming responsibilities and reducing bugs in code.
5. *Inter-Integrated Circuit (I^2C) Communications Protocol* – The I^2C communication protocol is used for synchronous communication between a master driver and slave peripheral device. This is the interface required to communicate with the LiDAR sensors, and best practices were implemented in the configuration of these sensors.
6. *Universal Asynchronous Reciever-Trasmitter (UART)* - The UART protocol is used for asynchronous communication between a transmitting and receiving device and is required to power the ultrasonic sensors, and best practices were implemented in the configuration of these sensors. The standard baud rate used is 9600 bits/second (?) for both sending and receiving data.

Tools Employed

In order to complete this project, a multitude of design, testing and programming tools were used. They are described below in their respective sections.

Hardware

In terms of tools needed to develop the hardware of the project, Multisim [12] and Ultiboard [13] were used. Multisim was used to design the schematics and overall circuitry layout of the project, while Ultiboard was used to place items onto the PCB. While there was previous knowledge on how to use both Multisim and Ultiboard from previous classes, a lot of new tools needed to be learned and improved upon to be able to place the selected parts onto our PCB board.

Firmware

The firmware for this project was produced in C using Texas Instruments' integrated development environment, Code Composer Studio [15]. To test and develop the firmware, the existing library MSP430 *driverlib* were used [24]. To ensure the code was loaded onto our MSP430FR2433 [1] the Texas Instruments' Debugger [15] for MSP430 were used. For initial testing of the sensors, the Arduino integrated development environment was used, including the driver libraries provided by Arduino off of GitHub, from budryerson[30]. With the Arduino IDE, this was a new tool that had not been utilized before, so coding with Arduino was a new skill that needed to be learned. Finally, GitHub was used to share code between the members of the team, as the whole code base was handled through it [25].

Demonstration

To test that the PCB and sensors were reading correctly, the National Instruments Virtual Bench was used [14]. The multimeter functionality of the Virtual Bench was used for continuity checks and voltage tests. Additionally, the ammeter functionality was used, as well as the function generator and digital logic analyzer for signal processing. While this is a skill that had already been learned through prior coursework, some improvement needed to be made to fully test the board and sensors.

Ethical, Social, and Economic Concerns

Ethical Concerns

There are several ethical concerns that arise from our project. The first of which is that this device would be utilized by people with visual impairments, and this could pose a threat to their health and wellbeing if not used correctly. Due to the placement of the sensors, if the hat is not worn at the correct angle (i.e. tilted too far upward or downward), this could affect the accuracy of the sensors and not provide the responses that are needed. If this is the sole device in use for detecting obstructions, if any of the sensors fail to read correctly this could put individuals in danger. Additionally, if the reusable battery dies prematurely, then a user could be left with zero information regarding their surroundings, and they would be unable to detect any obstructions. This is obviously a huge ethical concern, as we would hope that this product would be safe for everyone to use and put their trust into.

Social Concerns

While there are no concerns about privacy or security with this device because it stores no information about the user and could not be hacked into or altered, however there are social concerns prior to it being manufacturable. First, the product was never tested with someone who

was visually impaired and/or auditorily impaired. With that, the hat was also not fully tested for people with varying head sizes or hair types, which limits our ability to be confident that it would work for everyone. Further, the current angle of the sensors introduces the risk that a shorter person would not be detected by the sensors, with the exception of the forward-facing direction that has the added benefit of the incline sensor. Finally, our hat design could not be everyone's style, so there are limitations in the fashion options.

Economic Concerns

Due to the costs of the sensors, MSP430 board, and other economic cost factors that went into the production of this device, it would not be a very cheap product to purchase. Therefore, there may be some economic disadvantages that come with this product. Those that are unable to provide the costs to cover supplies and manual labor that went into the production of this device would not be able to utilize it.

Intellectual Property Issues

The hand-held navigation aid described in [26] consists of "a processing unit, a surrounding data collection unit, a front data collection unit". The patent gets into more specifics about each of the three units described above, breaking down each part. It details how the data collection unit will consist of plurality of sensors will provide data to the microcontroller, where the "metal tip 303 are process and take necessary action, such as blinking led signal light, red led strobe light 107 etc." [26] In addition to this, the battery used to power the microcontroller will be rechargeable via solar power. This product and the project created by our team are similar in that they both aim to allow blind or visually impaired individuals independence and the ability to "walk, jog or even run without help of any second person" [26] however the Visually Assistive Hat differs in that it is a hands-free device rather than having to constantly hold something. Additionally, the Smart Sensor Cane does not have motors to provide feedback and offers no feedback on obstructions from behind. Therefore, this patent would not get in the way of the patentability of our system.

Furthermore, the patent described in [27] builds off the existing white cane but incorporates the use of a time-of-flight (TOF) sensor to the top of the cane for obstacle detection. The use of such "advanced imaging technology" [27] allows for a more precise pixel-based image of the sensor's surroundings. Based on the distance readings of the sensors, tactile feedback is given to the user of the cane via a haptic interface, thus helping visually impaired people better navigate their surroundings. The patent "provides a reliable warning if there is a risk of collision" [27] within a limited distance from the obstruction. The design ideas in patented design are quite similar to those present in our own project. The design of our system also relies mainly on LiDAR time-of-flight sensors as these are the sensors used for obstacle detection from the user's front and back side, as well as incline detection. Furthermore, like the patent, our design makes use of tactile feedback to alert the user of coming obstructions as well. This Visually Assistive Hat varies from this patent as it is a hands-free system and makes use of auditory feedback. Such differences, however, serve as potential improvements in our device that can enhance user experience. Ultimately, due to the similarities in basic designs and goals

between our system and this patent our device would require refinement before it can be patented.

Finally, the Venucane described in [28] is a patented design that builds upon the classic white cane. This device also improves upon the basic cane design through the addition of various sensors as it is "directed [as] an electronic travel aid" [28]. The Venucane uses eight ultrasonic sensors, along with other a variety of other sensors – such as smoke detection sensors, liquid detection sensors, and metal detection sensors. Furthermore, this device uses pre-recorded audio messages to relay obstruction feedback back to the user. The device is advertised to have a "minimal physical interface," which is like our own design as it was designed with the goal of being intuitive [28]. However, the fact the device makes use of so many ultrasonic sensors, and numerous other types of sensors, makes the device more robust in comparison the Visually Assistive Hat. Our design, however, does differ in the fact that it is hands-free and, therefore, has more everyday utility. Our design also makes use of tactile feedback in addition to auditory cues. Based off the design of the Venucane, it may be more difficult for our design to be patented as is since it contains a subset of their more robust sensor system. With the addition of a more powerful sensor system for obstacle detection, however, our design can face patenting. Ultimately, based on the three patents we have observed, with more careful considerations to distinguishing factors the Visually Assistive Hat has the potential to be patented.

Detailed Technical Description of Project

The proposed project is a visually assistive wearable that would allow for the visually impaired to navigate their environment independently. The combination of LiDAR and ultrasonic sensors are used to provide the user tactile feedback in the form of motor vibrations and buzzer tones to notify them if they have objects approaching from a certain direction or if they are approaching a change in incline. Additionally, the project allows the user to toggle the settings of the system to turn off the buzzers, calibrate the incline sensor, switch between outdoor and indoor mode, or to turn off the system entirely.

One of the main goals for this project was to provide a sleek design to conceal the electronic system so that users feel comfortable wearing the device in a public setting. The team decided to use the following hat shown in Figure 1 to provide enough space to store the circuitry while also providing a stylish design for the user. In addition to this, other craft supplies (thread, hot glue, foam, etc.) were used to mount the sensors, buzzers, and motors to the hat in a professional and clean fashion.

Figure 1: Image of Hat with Sensors and Buttons Attached

Two TFmini-S LiDAR sensors are used to determine if any objects are in the front or back of the user and the remaining LiDAR sensor is used to detect changes in incline. Then the US-100 ultrasonic sensors are used to determine if any objects are either on the left or the right of the user. If they do detect an object, each of their respective motors will vibrate with increasing intensity as the object moves closer. As for the incline sensor, there is a motor that is attached to the top of the separator between the user and the component storage section of the hat which vibrates whenever there is a change in incline. In addition to the motors, piezo buzzers are mounted on the left and right side of the hat's brim to allow for auditory feedback as well. Lastly, there is a button on the front right side of the brim which allows for the user to toggle the piezo buzzers on and off, calibrate the incline sensor to account for their height, switch between the indoor and outdoor setting, and to put the system into idle mode.

The piezo buzzer functionality can be toggled by pressing the button once quickly, which will trigger the motors on the left and right to toggle three times to indicate the piezo buzzers have been turned off. The indoor and outdoor setting can be toggled when the button is held for one second, which will cause the motors to all toggle at once. Then, the calibration operation is activated by holding the button for two seconds, which will cause the motors to toggle once in a circle starting with the front and moving counterclockwise. Lastly, the button can be held for three seconds to put the hat into idle mode, meaning that the motors and piezo buzzers are not providing feedback. This is indicated by all the motors toggling three times. The hat can be put back into active mode by pressing the button once after entering the idle mode, causing it to enter back into its normal functionality. Images of the hat can be found in Appendix D.

Hardware

The hardware for the Visually Assistive Hat was placed on a PCB designed as a 20-pin booster-pack for the MSP430FR2433 Launchpad [1]. The PCB contains the voltage regulators to supply power to sensors and MSP, communication module pin connections to the respective sensor connectors, pin connections to our buzzers, and MOSFETs to toggle the voltage supply to both our motors and LiDAR sensors. Additionally, the PCB contains the corresponding pin

headers, connector housing for each of the peripheral devices (sensors, motors, buttons, battery, etc.), and passive components that are needed for the functionality of the other regulators and buttons. Below in Figure 2 the general overview of the schematic is depicted along with some further explanation following the figure.

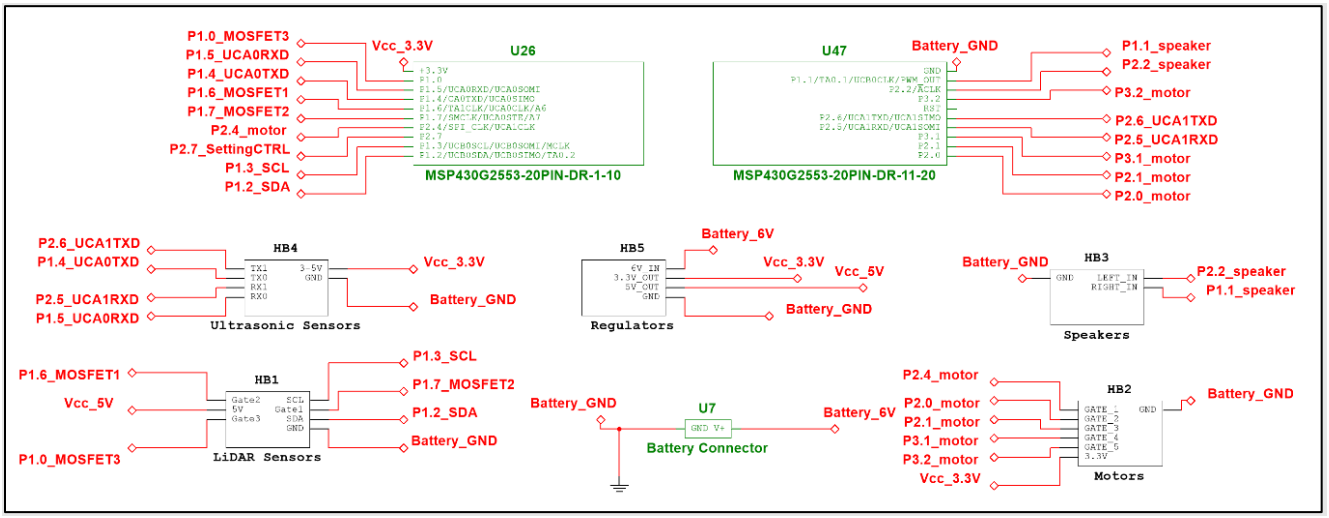


Figure 2: Hierarchical Schematic Overview

To start, some design decisions were necessary to optimize the use of power by the MSP and peripheral devices. The voltage regulators were selected to supply 5V and 3.3V to the respective devices connected to the booster-pack. To reduce the power dissipated in the form of heat by the voltage regulators dropping the voltages to the required levels, we decided to choose a 6V battery and voltage regulators that were rated to handle this input. A two-pin connector was also installed to allow the 6V batter to be connected securely and traced to the regulators. Another power design decision was the inclusion of P-Channel MOSFETs connected to the power sources of each of the LiDAR sensors as shown in Figure 3. This was done to incorporate power-cycling on these sensors since they pulled such a large amount of current when turned on, causing a reduction in battery life.

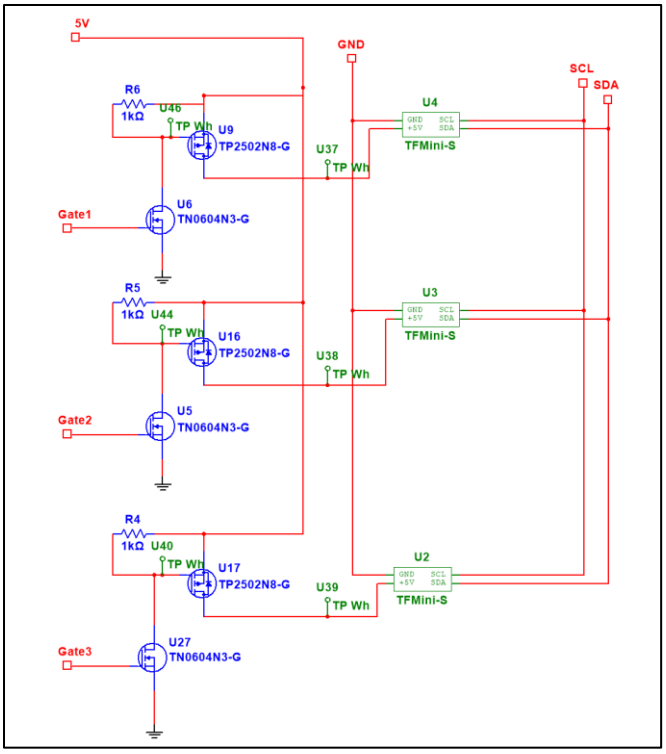


Figure 3: TFmini-S LiDAR Power Cycling Circuit

Another significant part of the PCB design was creating a button with static protection to ensure we could toggle the settings of the hat without potentially damaging the MSP430. A transient voltage suppressor (TVS) diode was used with some additional circuitry shown in Figure 4 to create an active low button.

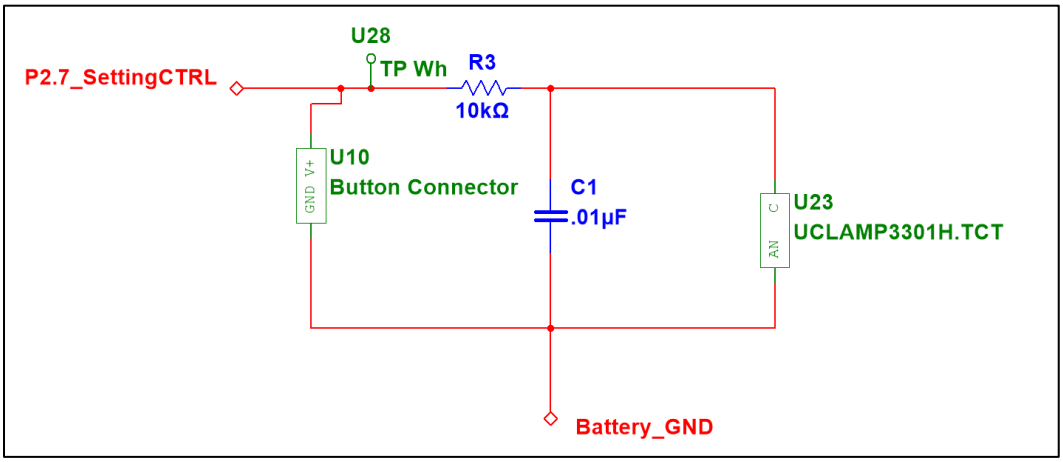


Figure 4: Button Circuit

The last of the circuitry on the PCB was standard with just providing traces to connect the power supplies to each of the sensor connectors and connecting pins to the necessary outputs or inputs. The piezo buzzers operated with 1mA of current and could be directly traced to their respective pins for power. The motors were directly sourced from the 3.3V output of the voltage

regulator since they required a larger amount of current and were toggled using an N-Channel MOSFET. The remaining parts of the schematic can be examined in Appendix A. Additionally, the PCB design is depicted below in Figure 5 with the traces, connectors, passive elements, active elements, and various test points.

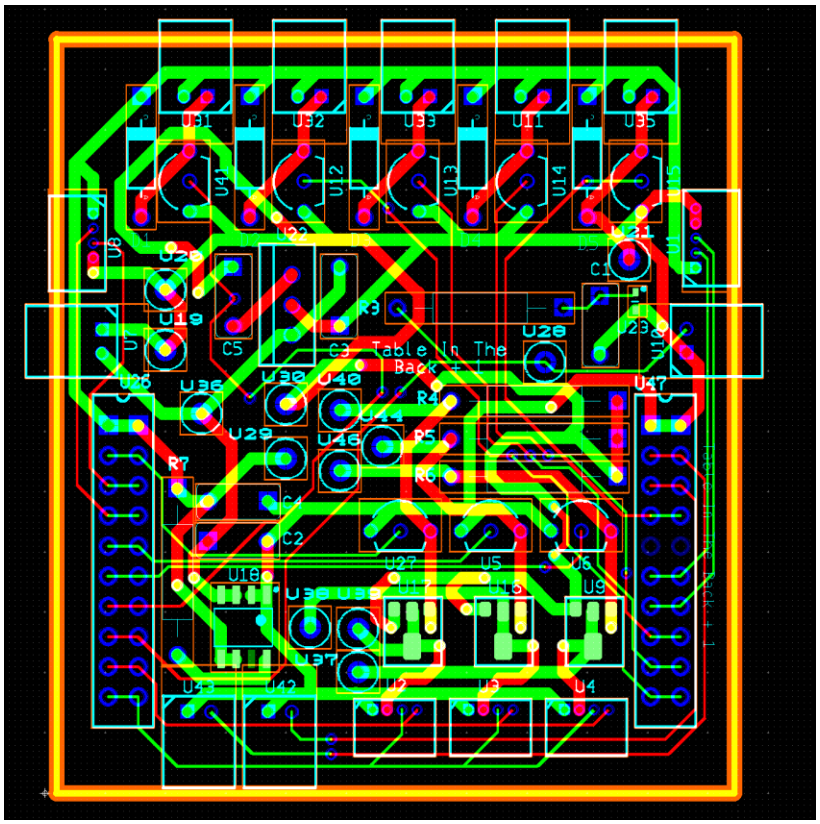


Figure 5: PCB

Software

All this software was developed in Code Composer [15] and loaded onto the MSP430FR2433 using the debugging software. The Barr C Standard [23] was used for all the code we included, which can be found in detail in Appendix B.

Task Scheduler

The task scheduler was designed as a round-robin scheduler to cycle through each of the tasks we must perform for our software. A task structure was defined in the code to hold a pointer to the function to be executed, cycle counter, task execution period, and *fsm_type_t* pointer respectively. The table, which is just an array of these class structures, cycles through each of them and increments the cycle counter variable until it is equal to the execution period. When this point is reached, the *fsm_type_t* pointer is passed into the task function for execution and the counter is then reset to zero. It should be noted that each of the FSM structures are other structure types (*lidar_type_t*, *ultrasonic_type_t*, etc.), but they contain an *fsm_type_t* as their first parameter and can therefore be casted to this *fsm_type_t* pointer to be passed into each task function. This allows the scheduler to run through the table without having to be concerned with

what type should be passed in. Then when inside the task function, each of the *fsm_type_t* pointers can be casted back into the respective structure pointer for the task so that the proper data can be accessed for execution.

There are twelve tasks loaded into the table which are as follows: the button checking task, UART communication with the left and right sensor, I^2C communication for all three sensors, five motor tasks, and a piezo buzzer task. Each of these has their own respective description in the sections to follow. It should also be noted that this scheduler runs in main and does not run on its own independent timer interrupt. The reasoning for this is because our MSP430 was discovered to not allow the priorities in the interrupt vector table to be changed. This made it so that the task scheduler timer would interrupt the I^2C, UART, and PWM functions because they relied on interrupts which were of lower priority. Therefore, we chose to let the scheduler run in main so that it would not only allow for the other interruptions to occur, but to also run as fast as possible to ensure there is no delay in the data collection.

Inter-Integrated Circuit (I^2C) Communication

The I^2C communications software we created follows professional standards [29] required by the technical industry and was used to communicate with our LiDAR sensors. The initialization function sets the clock source, data rate, disables the automatic stop bit, clears the receive and transmit interrupts, enables the interrupts, sets the pins to the correct peripheral mode, and sets the default slave address to the front sensor, and then enables the entire module. This is only called once in the main function since there is only one I^2C module to initialize.

Additionally, a task was designed and added to the task scheduler to collect data from each of the LiDAR sensors. This task was designed to be universally used with the *lidar_type_t* structures to appropriately access the correct slave addresses for each sensor and store the distance data inside of a buffer so that it could be averaged for reference later by the motors and piezo buzzers. A custom function called *get_lidar_data_cm()* is used inside this task to send the command to request the current distance in centimeters and then read the value from the sensor. This *get_lidar_data_cm()* function relies on interrupts that trigger the I^2C interrupt service routine (ISR) to perform the communication process of sending the start bit, the address, and the corresponding data to the sensor. The I^2C module ISR also handles sending the receive command, which expects a 9-byte frame from the sensor which contains the lower and upper bytes for the distance data in bytes 3 and 2 of the frame respectively [29]. The returned value is then stored in the *lidar_type_t* structure's distance buffer and the newest data average is computed. Additionally in this task, the TimerA0 used for PWM by the buzzers is disabled at the beginning of the task so that the I^2C communication is not interrupted. Then at the end of the function, the timer is enabled again to allow for the PWM to resume. This task is executed for each sensor individually, as shown in the task table.

Universal Asynchronous Receiver-Transmitter (UART) Communication

The UART communications between our ultrasonic sensors follow the industry standards for communication. The initialization functions for the UART modules are called independently for the left and right ultrasonic sensor since they are independent of each other. However, both

modules are configured with the same initialization parameters such as 9600 baud rate, no parity bit, least significant bit first, one stop bit, and oversampling baud rate generation. Additionally, the necessary transmit and receive interrupts are enabled in the initialization function as well as configuring the RX and TX pins to the correct peripheral mode.

A task was created for the task scheduler that takes in the *ultrasonic_type_t* structure and requests data from the ultrasonic sensor. This is the only operation performed by this task since the rest of the communication is handled by the UART ISRs. Since each of the UART modules have their own ISR, the data is requested by the task and directly stored in the respective *uart_type_t* structures when collecting the distance data. Similar to the *lidar_type_t* structure, this data is stored in a buffer and the average is computed each time a new data value is received. This buffer serves as an averaging filter to prevent abrupt changes to the motors or piezo buzzers since the ultrasonic sensors are not very precise.

Pulse Width Modulation (PWM)

PWM was initially used to run both the piezo buzzers and the motors. Due to restrictions in the number of specialized pins available, both piezo buzzers operate on TimerA0.1. The built-in PWM functionality is not used as a result, and rather the device is run through two ISR interrupts and toggling the piezo buzzer pins as general-purpose input output (GPIO). TimerA0.1 is initialized in up mode and counts up to the value initialized in the capture compare register 0 (CCR0) register.

In order to generate different tones for the different directions it was necessary to implement a universal *speaker_manager_type_t* structure to manage the tones executed by the piezo buzzers as well as preventing the duty cycle of the CCR registers from changing when the piezo buzzers are disabled. To further differentiate between the different directions, the piezo buzzers operate with a pause in between. If an obstacle is detected, indicated by a value in the *speaker_manager*'s *speaker_frequencies* array, the buzzer pins are set high on overflow. An interrupt occurs when the value in the CCR1 register is reached, which sets the piezo buzzer pins as low. Both CCR registers are set during the motor task, which is explained in the motor section, with CCR1 as half the value of CCR0 in order to generate a 50% duty cycle. Then to change the tone of the piezo buzzers, the period of the PWM signal is varied by increasing or decreasing the CCR0 value. This changes the frequency of the tone generated by the buzzers because the period in which the buzzers are high is varied.

Due to the inability to adjust the priority of interrupts and the synchronous nature of the I^2C communication, it was later decided to toggle the motors independently to reduce the number of necessary interrupts. This is a necessary tradeoff to enable functionality of our LiDAR sensors but comes at the cost of lowered precision and difficulty varying the duty cycle of the motors. Due to the sensitivity of the piezo buzzers relying on timing to generate accurate tones they were required to remain as PWM interrupts. To prevent interruptions of the LiDAR task, the timers are disabled at the start of the LiDAR communications and the button task to prevent accidental interference, since TimerA0.1 runs at a higher priority on the MSP430FR2433 [1]. This tradeoff

results in slightly distorted tones from the piezo buzzers but allows for the successful execution of our LiDAR task.

Motors

The vibration motors are run by conditionally toggling the GPIO pins. This is triggered in the task scheduler for each GPIO pin first through a wrapper function, one for the group of LiDAR sensors and one for each of the ultrasonic sensors. This is facilitated using the global *motor_type_t* structure, which contains a *fsm_type_t* pointer to the sensor. The wrapper function dereferences the different sensors to access the data, then passes it through to an external function to be decoded. This function groups the sensor distances into categories indicating whether the objects are within close, moderate, or far range from the user. The range of these buckets were defined through experimentally assessing how soon the user would require notification of an obstacle depending on whether the hat is in indoor or outdoor mode. The identifier of close, medium and far distances is defined in the *defines.h* file and relates to different tones produced by the piezo buzzers. These identifiers are loaded into the related index of the *speaker_manager*'s *speaker_frequencies* array to reduce the number of accesses to the piezo buzzers and the time required to execute each cycle of the task. Finally, there is a generalized vibration function that toggles the motors at a variable speed based on the decoded data in conjunction with a counter variable. High frequency runs every cycle, medium frequency runs twice every eight cycles, and low frequency runs once every eight cycles.

Button

The button has four settings triggered by the length of time the button is pressed, with user feedback provided through differing patterns in motor vibrations. They were chosen because they were more distinctive than the piezo buzzers, and a full second was left between each pattern to ensure the user notices the distinction and has time to release the button.

The first setting in the cycle turns on or off the piezo buzzers and is indicated by toggling the left and right vibration motors three times. The second setting in the cycle toggles the threshold of object detection between indoor and outdoor mode and is indicated by toggling every vibration motor. The third setting is to calibrate the incline sensor to the user's height and is indicated by the pins being toggled high and then low in a circular pattern around the hat. The fourth setting turns off the feedback of the motors and piezo buzzers entirely and is indicated by toggling every vibration motor three times.

Functionality was built to turn off the piezo buzzers and not the vibration motors because the motors provide less ambiguous and more immediate information about obstacles. The potential safety tradeoff in turning off auditory warnings was deemed necessary as a result of experimentation discovering always having both piezo buzzers and vibration to be overwhelming. Indoor mode has a reduced distance threshold for object detection by the front and back LiDAR based on the values written in *defines.h* which were determined based on experimentation. The left and right sensors remained constant as we anticipated similar needs for detecting obstacles next to the user. The incline sensor also remained constant, as the user's height would not change. The calibration is done based on the information being collected by the

incline LiDAR sensor in the background, storing the user's height to be used as comparisons. Prior to calibration the incline vibration motor is disabled. Turning off the device ceases execution of the task scheduler and disables the PWM interrupts until user next presses the button again. Calibration data is saved after the hat is put into idle mode, as a normal use case for the device would be the same user wearing the hat, and as such the height stored for the incline sensor would not need to change.

Hat Assembly

The specifics for the hat assembly, which involved the placement of the three LiDAR sensors, the two ultrasonic sensors, two piezo buzzers, five vibrating motors, the microcontroller, PCB, and battery in the hat can be seen in the following sections. The directions associated with each component (front, left, right, and back) are in respect to the user's directions when wearing the hat.

General Fit

The first portion of hat assembly was dedicated to creating a more secure fit of the hat on top of a user's head since many hardware components had to be mounted on the inside of the hat. To ensure a secure fit, elastic was sewn into the black lining as well as to the border of the black lining.

Microcontroller and Battery Placement

Both the microcontroller-PCB combination and the battery were placed on the inside of the hat. First, a plastic encasing was placed on top of the PCB, which was mounted to the microcontroller. This configuration was then placed inside the hat, towards the front. The battery was then placed next to the PCB-microcontroller combination and secured to the inside of the hat, towards the back, using Velcro. Foam inserts were then cut and placed around the PCB-microcontroller combination and the edge of the hat, between the PCB-microcontroller and battery, and around the battery against the back of the hat. By doing so, it was ensured that the components inside the hat would not move around while the user was wearing the hat. Finally, after plugging in all the necessary wires into the PCB-microcontroller combination to power all the sensors, motors, and buzzers, a circular foam insert was placed on the inside of the hat, on top of the PCB-microcontroller and battery configuration so that the wearer would not feel the components on the inside of the hat.

Sensor Placement

The first sensors mounted to the hat were the left and right ultrasonic sensors. In order to mount these to the hat, two circular holes were cut on the middle of the left side of the hat and on the middle of the right side of the hat. These holes were made so that the transmitter and receiver on the ultrasonic sensor could be pushed through the holes, from the inside of the hat, so they would be facing the user's surroundings, while the rest of the hardware of the sensor would be protected on the inside of the hat. Next, the placement for the three LiDAR sensors was finalized. The LiDAR sensor for the user's backside was sewn into the back of the hat with a rectangular piece of foam placed behind it to ensure that the sensor had the correct angle. The

same was done for the front sensor and incline sensor. The front sensor was sewn into the brim of the hat at the front center of the hat and the incline sensor was sewn to the left of the front LiDAR sensor. Unlike the front and back LiDAR sensors, the incline sensor's foam insert was much thinner so the sensor would face the ground more directly. This helped to ensure that the incline sensor detected inclines by the user's feet.

Piezo Buzzers/Motor/Button Placement

Since only two buzzers were used in the design, each was sewn into the underside of the hat's brim near the user's ears. Next, in order to mount the motors to the hat, the four motors associated with the front, left, right, and back sides of the hat were sewn into the black lining of the hat. Each motor was sewn into the position of the hat that corresponded with the direction it was associated with. Thus, the front motor was sewn into the lining at the front of the hat, the left motor was sewn into the left side of the lining, the right motor was sewn into the right side of the lining, and finally the back motor was sewn into the back side of the hat's lining. The fifth motor, which was associated with the incline LiDAR sensor, was mounted to the bottom of the circular foam insert placed on top of the PCB-microcontroller combination and battery on the inside of the hat. Thus, the tactile feedback for incline detection is given on top of the user's head. Finally, since the most common dominant hand is the right hand, the button was secured to the underside of the hat's brim on the right side for user ease.

Project Time Line

The original proposed Gantt chart can be seen in Figure 6. The original timeline thought that we'd be able to begin programming and testing immediately upon the start of the semester, however this was infeasible because we hadn't received our parts to be able to set it up until more than a month in. Additionally, the attachment of all the parts into the hat was planned to take place at the end of October, however due to the setbacks and delays in getting our software up and running this was also pushed back to much later than originally planned. We had hoped to be testing and making final adjustments on the entire system for the last month of capstone, but once again this deadline was not met due to unforeseen circumstances regarding issues with I2C, the MSP430, and multiple PCB submissions and send outs to 3W.

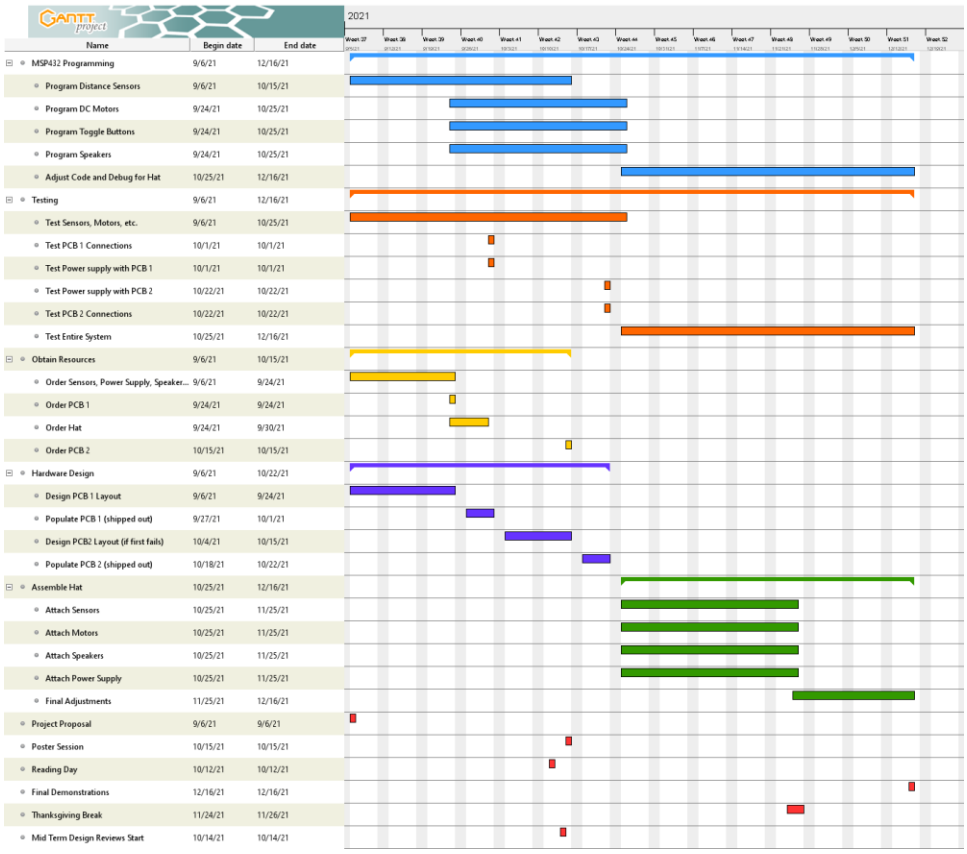


Figure 6: Original Gantt Chart

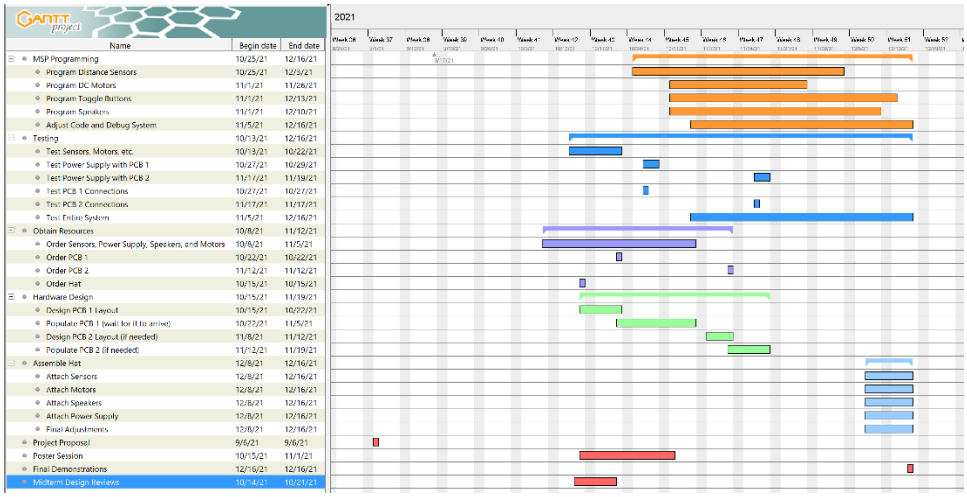


Figure 7: Actual Gantt Chart

To make all these tasks more manageable, the team completed a lot of peer programming, as well as collaborating on tasks to streamline the process. Figure 8 shows how the tasks were distributed and serialized.

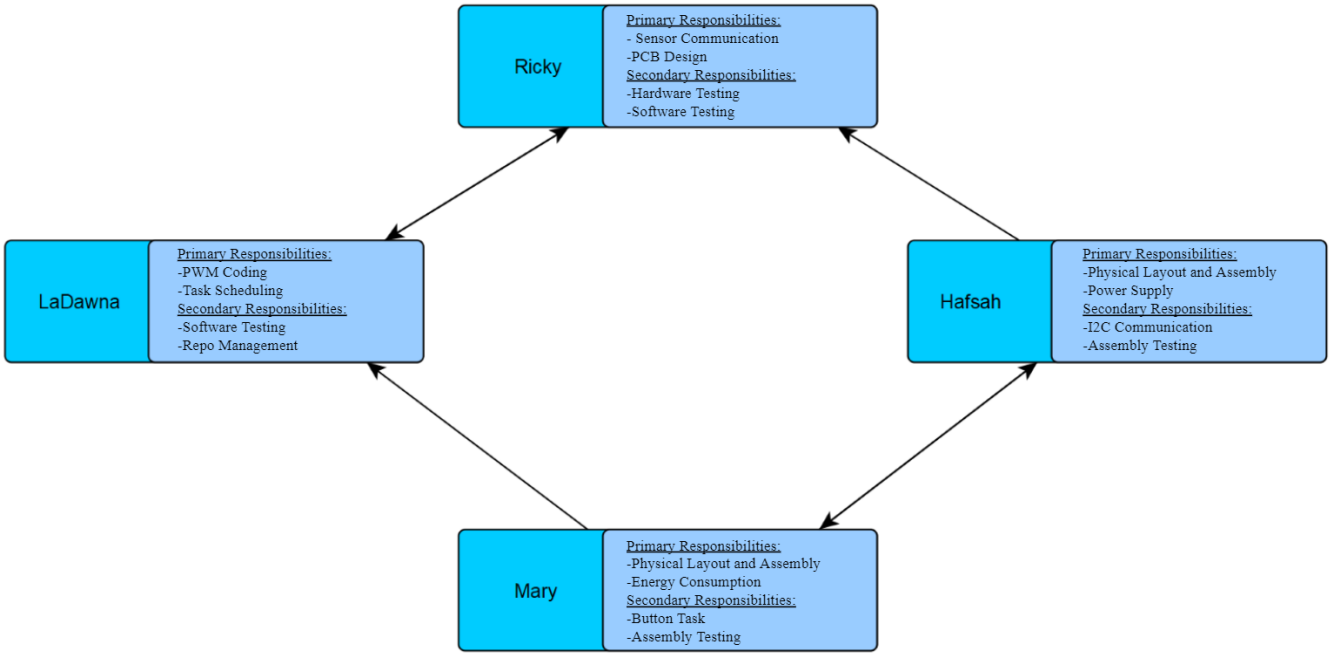


Figure 8: Tasks Divided Amongst Team members

Test Plan

The Visually Assistive Hat was divided into a hardware and software systems for individual testing prior to group integration. Each of the sections below describe the processes involved in testing each of the components of the project.

Hardware

The overall test plan for hardware is laid out in the flow chart for Figure 9 and described in detail following the figure.

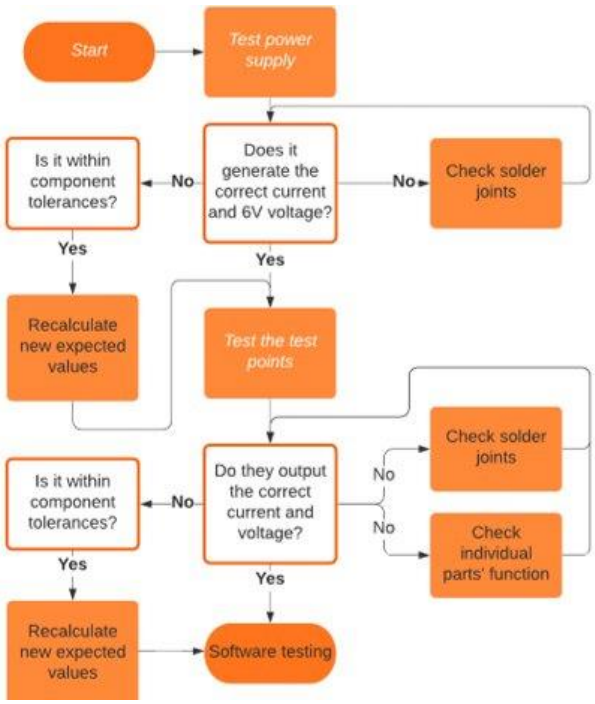


Figure 9: Hardware Test Plan

Referring to Figure 9, the first part was to perform a test on the power supply to ensure the correct voltages were generated by both the 5V and 3.3V regulator. These are shown in Figure 10 and Figure 11 below. Additionally, a continuity test was conducted on all of the connections to ensure the PCB design accurately connected our MSP430 to the correct outputs and inputs.

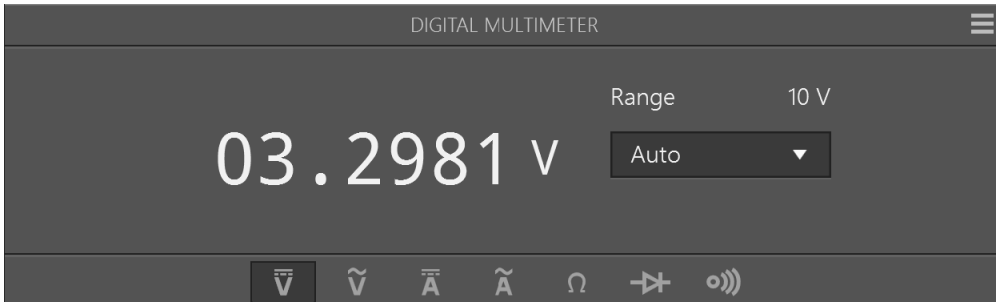


Figure 10: 3.3V Voltage Regulator Output

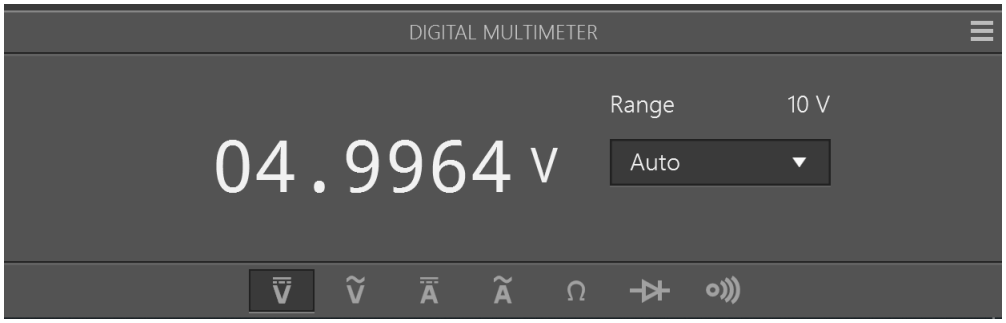


Figure 11: 5V Voltage Regulator Output

The next step was to verify that each of the test points generated the correct voltages for each of the outputs to the peripheral sensors. This was tested for the voltage outputs of the P-Channel MOSFETs for power cycling the LiDAR sensors, outputs for the motors, and power supplies for the UART sensor. This was done and it was concluded that each of the expected voltages were created for all the test points except for the output from the P-Channel MOSFETs. We were getting voltage values ranging from 4.62V to 4.85V when the expected voltage for the LiDAR sensors was 4.9V to 5.1V. This meant that the correct voltage supply was not being generated to power the sensors and was determined to be because the on resistance for the P-Channel MOSFET was too high, causing a voltage drop of about 0.2V-0.4V. To fix this problem, the test points connected to the LiDAR 5V pins were directly connected to the 5V output of the voltage regulator. This bypassed the P-Channel MOSFETs and just provided constant power to the sensors, which meant a shorter battery life for our device but power to the sensors had a higher priority over the power-cycling functionality.

After all these testing and modifications to the final PCB, the design was ready to mount to the MSP430 to test the software and ensure that communication was able to be established with the sensors. One final note is that right before the final submission, it was realized that the PCB did not include pullup resistors on the I^2C lines. To fix this issue, our team sent external pull up resistors to the 3W manufacturer for them to attach these resistors to the I^2C pins and the 3.3V voltage regulator output.

Software

The software was tested based on the flow chart shown in Figure 12.

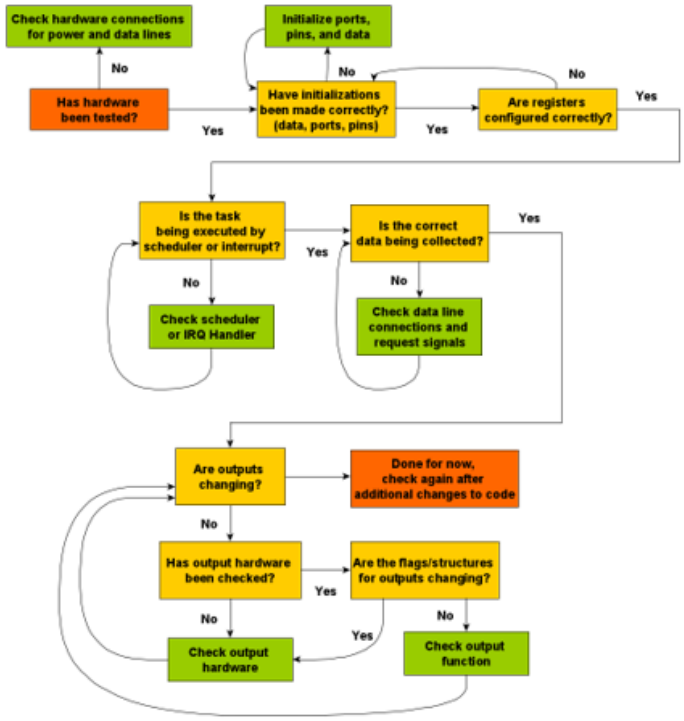


Figure 12: Software Test Plan

The software was programmed in isolation first, with the ultrasonic and LiDAR sensors, PWM interrupts, motor toggling, and button functionality tested independently prior to their integration into the main codebase. Once each of these components was working as expected, then they were integrated into the task scheduler and merged into our main branch on Github and tested with the rest of the system [25]. Testing of the individual software components is outlined below.

LiDAR / I^2C

The LiDAR communication was tested by comparing the response of the sensor to an Arduino using an open-source library [30] to communicate with the TFmini-S sensors via I^2C. Once communication was established, the logic of the start and stop bits along with the bytes of data were replicated on the MSP through the initialization function and I^2C interrupts. Once these modifications were made to the code, then the SDA and SCL pins were monitored when connected to the TFmini-S sensors and the data was capture as shown in Figure 13. Additional communication verification for I^2C is shown in Appendix C.

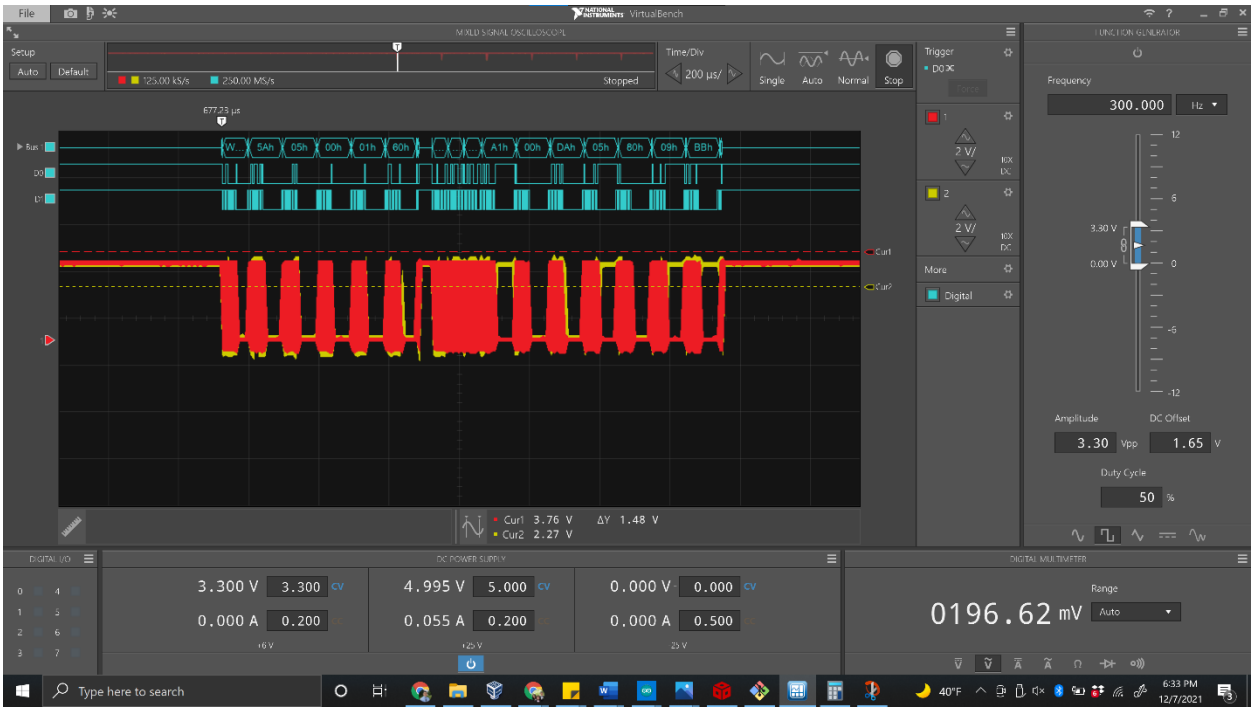


Figure 13: I²C Command and Read from LiDAR

Ultrasonic / UART

To test UART communications, the first step was to write transmission software to send the request to the ultrasonic sensor for the distance data. This was tested using a digital logic analyzer on the Virtual Bench [14] to see if data was being correctly sent, which is shown below in Figure 14 below.

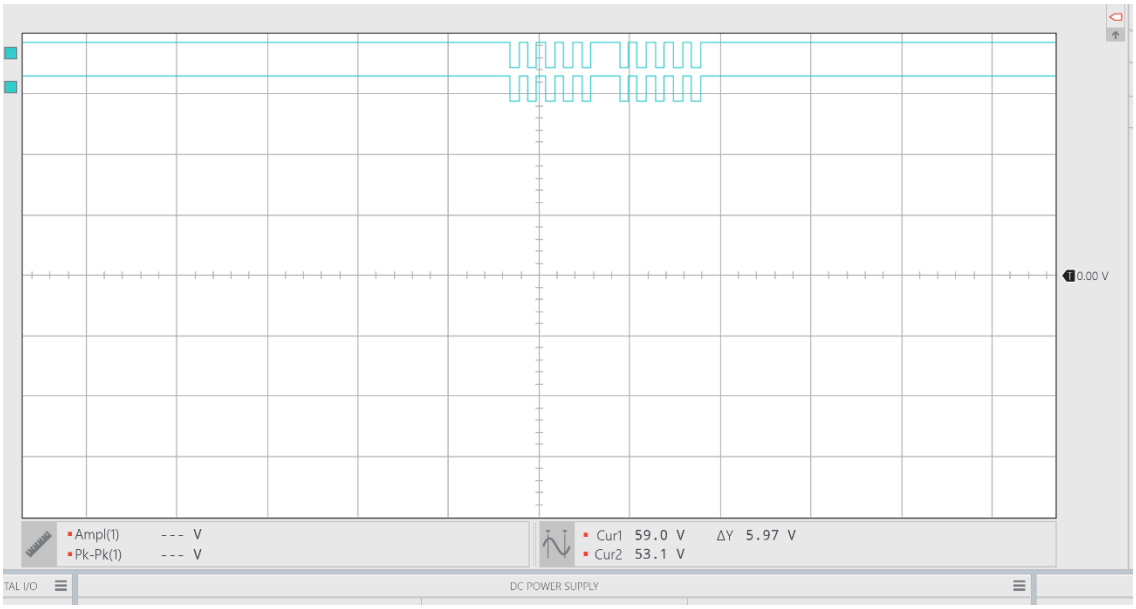


Figure 14: UART Transmission Verification

After getting transmit data sent, the next step was to test that there could be actual data that could be sent to the ultrasonic sensor and then received. The UART ISR in the code was modified to handle data received on the RX pin and was tested by using the register window in Code Composer [15] to see if data was being sent, as shown below in Figure 15.

UCA1RXBUF	0x00B7
UCA1TXBUF	0x0055

Figure 15: UART Receive Verification

Once this data was collected, then final adjustments were made to the ISR to store the low and high bytes of the distance data in the *ultrasonic_type_t* structure so that the actual distance could be determined and stored within the buffer.

One issue we did find in our MSP is that the left module (UART_A0) was not receiving data properly. Upon further inspection, it was seen that the voltage levels on the RX pin were never pulled entirely low when the ultrasonic was connected to the MSP, as shown in Figure 16.

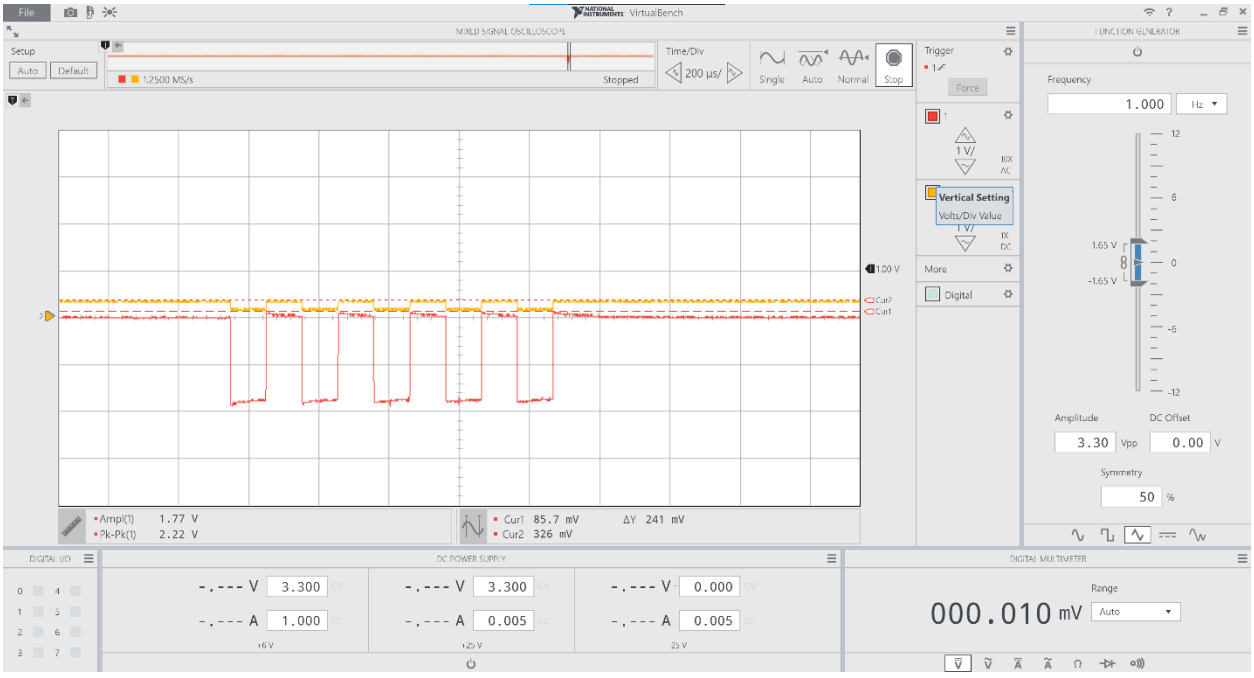


Figure 16: Broken UART_A0 RX Pin (orange) compared to working UART_A1 RX Pin (red)

The first thought was that the ultrasonic sensor was broken, however when testing both sensors on the working right module (UART_A1), both sensors were able to send data back to the MSP. Therefore, we decided to test the voltage of the RX pin when configured as an output just to see if the pin was potentially blown from earlier connection tests with an Arduino. Figure 17 shows the comparison of P1.5 (RX pin; suspected broken pin) output in red to the actual output to of P1.7 in yellow. This figure shows that the voltage difference between these two pins is quite significant and made the team suspect that the pin was damaged. A new MSP was

ordered, but this same issue still appeared even for a new MSP. Even after modifying code and ensuring no other modules were trying to control the pin, the team concluded that this module was broken and would make it so the left sensor could not be used during the demonstration. However, since UART communication was still successful on the left port and the code was identical for both ports, it was assumed that by ordering a new MSP430 in the future with this code should produce the correct results.

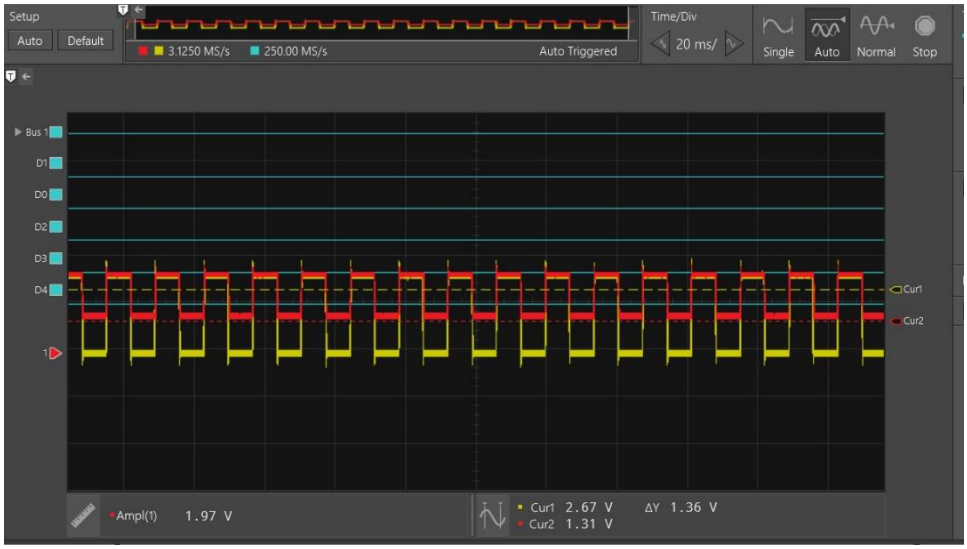


Figure 17: Output Difference Between the Broken P1.5 (RX pin) and a Working Pin

PWM

The PWM interrupts were tested by inserting default alternating values into the *speaker_manager* to test wave generation and the ability to alternate the duty cycle, shown in Figure 18. After testing that the pulses are being generated in the virtual bench, the piezo buzzers were connected in order to test the tones being generated and discover a variety of tones that would be sufficiently differentiable to the user. The motors were tested in a similar fashion, first using PWM interrupts and later using the simple conditional toggling method to correct the issue of invariable interrupt priorities. Both were tested individually with PCB prior to integration, using hardcoded values to ensure the components themselves were functional.

Figure 18: PWM Interrupts Varied Duty Cycle Test

The button code was first tested by flashing a different LED depending on how long the button was pressed to trigger the interrupt. The code was then integrated with the rest of the system and tested experimentally. During these experimental tests, we identified the need to implement stronger feedback to the user and differentiate between the various settings. Further refinements were to the time delay between each setting and the type of feedback produced to make this more intuitive based on experimental tests.

Hat Assembly

The hat assembly required testing for every step along the process, to ensure that the locations of each sensor were correct and would be angled well for use. For the ultrasonic sensors, they were originally taped onto the sides of the hat and then moved up and down the sides of the hat to ensure they are in an appropriate position before being sewn down. They were tested while taped on to make sure that they were not reading information off the bridge of the hat and were genuinely reading from the surroundings. The same was performed for the LiDAR sensors, to ensure everything had an appropriate placement before being sewn down.

To test the motors during assembly, they were taped to the inside of the hat first and had the software run on them, before being sewn into place. They were then routinely tested to ensure that they were vibrating against the user's head in a manner that allowed the user to feel what was happening. The incline sensor motor required the most testing, as it was too weak to feel in the original location (the brim of the hat). Modifications were made and it was tested in numerous locations, including dangling off the hat, secured into the top of the hat, and hidden under felt. The piezo buzzers were tested in the same manner, ensuring that their location on the hat was easy to hear and not too overpowering.

Finally, the button placement was tested by having numerous individuals wear the hat, reach up, and grab the brim in a location that felt most natural to them. During each of these tests, the location of where the individual had grabbed was marked with chalk, and after these experimental tests were completed the location for the button was selected.

Final Results

The final device includes a hat with five mounted sensors, two ultrasonic sensors pointing in the left and right directions, and three lidar pointing towards the front, back, and one angled towards the ground as the incline sensor. All five sensors can return data relaying the distance from the next nearest obstruction; however, due to a blown pin on the microcontroller only four of the sensors are capable of transmitting data to the microcontroller, with the left ultrasonic sensor receiving pin unable to pull down to a low enough voltage to function correctly. This discovery is outlined in the Ultrasonic / UART Testing Section. This raw sensor distance data is converted to different frequencies and the vibration motor associated with the sensor detecting the obstruction vibrates at a more intense frequency as the obstruction gets closer to the user. The buzzers change in tone as the objects get closer to the user, with a higher tone produced the closer the object is. The buzzers are generalized to every sensor, and as such alternate through each direction when generating tones. The button associated with the project contains four different settings, toggled by holding the button down for different lengths of time. These settings are differentiated between different patterns of vibrations from each of the perimeter vibration motors. The first setting turns on and off the buzzers, as experimentation showed having both going at all times can be overwhelming. The second setting toggles between indoor and outdoor mode, which is an adjustment of the distances set for obstacle detection for the front and back sensors, with outdoor mode having a longer distance as we anticipate a greater need to see farther while outside. The third setting triggers the calibration setting for the incline motor to adjust the incline sensors for variations in the user's height. Prior to calibration, the incline sensor is inactive. The final setting places the device in idle mode and stops execution of the task scheduler. If a height value had been retrieved prior to putting the device in idle mode, it remains saved in the system upon resuming the tasks in the scheduler. This is because we assume the same user will be returning to use the hat during normal functions.

Based on the results of the capstone and using the grading rubric, the team is evaluated to have a score of 9 which would result in a grade of a C. However, upon reflection and seeing the error with a blown communication pin to our ultrasonic sensor impacted a lot of the scores for each column, we believe that our team should receive a score of a B+ or low A to be more reflective of the effort our team put forth this semester. The individual section breakdowns are shown below and assertions as to why the team believes they should receive a higher grade than a C.

Table 1: Grading Rubric and Breakdown

Points	***LiDAR and Ultrasonic Sensor Functionality***	***DC Vibrating Motor Functionality***	***Piezo Buzzer Functionality***	***Power Source and PCB***

4	*All 5 sensors are able to collect object distance and incline*	*All 5 motors vibrate with their respective sensors*	*Both piezo buzzers work and indicate objects on left, right, front, back, and incline change*	X
3	*4 perimeter sensors collect correct data, but incline sensor does not*	*4 perimeter motors vibrate with their respective sensors*	*Both piezo buzzers work and indicate left, right, front, back and not incline change*	*Power Source works with PCB and is in a casing and mounted to the hat*
2	*2+ sensors collect data*	*2+ motors vibrate*	*Both piezo buzzers work and indicate objects in vicinity from two directions*	*Power Source works with PCB and is NOT in a casing and mounted to the hat*
1	*Only 1 sensor collects data*	*Only 1 motor vibrates*	*Both piezo buzzers work and indicate when object is around*	*Power Source works with PCB and is NOT in a casing and NOT mounted to the hat*
0	*No sensors work*	*No motors work*	*None of the piezo buzzers work*	*Power Source and PCB do not work*

Grading Breakdown:

A+ 14-15 points
A 13 points
B 12 points
C 8-11 points
D 4-8 points
F less than 4 points

LiDAR and Ultrasonic Sensor Functionality

The LiDAR sensors were proven to work and return data to the MSP for the front, back, and incline sensor. In our rubric, there was a large emphasis on getting just the incline sensor working rather than just the number of sensors that are operational, making the rubric very narrowly focused. However, considering the blown pin discussed above, this made it very difficult to get a value above a 3 in the sensor column since the rubric relied on having the incline sensor failure being one of the main issues to deduct points from. Additionally, since the team was able to successfully establish connection on the right UART module, receive data with both ultrasonic sensors to prove they work, and provide evidence that the pin on the

MPS430FR2433 was faulty, then we feel as though we should not be penalized 2 points for this error and should receive at least 3 points for this column.

DC Vibrating Motor Functionality

The DC vibrating motors were proven to vibrate individually, both based on output wave to the virtual bench and by toggling all pin outputs alternating high and low on the PCB with the vibration motors connected and testing them by hardcoding the categorized frequencies into the program. All vibration motors were capable of being triggered by the individual values, and since identical functions are running on each motor, we expect that if the ultrasonic sensor were working all vibration motors would be functional. Therefore, we feel as though we should not be penalized for this error and should receive 4 points for this column.

Piezo Buzzer Functionality

The buzzers were proven to be successful in indicating the object distances. This was done by attaching each sensor (both LiDAR and ultrasonic) to the MSP and having the task scheduler generate tones as we moved objects closer to each sensor. Three different tones were generated, the lowest being played when the object passes the first tolerance distance, then a medium pitched tone on the next threshold, and the highest pitch tone when inside the closest tolerance distance. Having all these various tones generated for all the sensors in our task scheduler gives us a score of 4 for the buzzer functionality.

Power Source and PCB

For this section, we were able to correctly construct a power source and PCB with a protective casing as shown below in Figure 19. However, this does not provide a casing around the battery, which would have been difficult to obtain since the battery is so unique and did not have any casings provided that would adhere to the NEMA waterproof standard when looking on Digi-Key and other product providers [20]. Despite this, we still attempted to provide protection around the PCB and separation of the battery from direct contact to the user's head, which we believe would give us a score of potentially 2.5 instead of 2 since there were no waterproof casings that we would be able to find and develop on our own with 3D printing.

Figure 19: Inside Hat with PCB Casing

Rubric Summary

Based on these assessments, we hope our grade would be adjusted to 13.5/15, which is a low A on our scale. The LiDAR and Ultrasonic Sensor Functionality section shifted from a score of 2 to 3 because singling out the incline sensor functionality was an overly narrow specification. The DC Vibrating Motor Functionality section shifted from a score of 3 to 4 because all five vibration motors would function correctly without the broken sensor. The Piezo Buzzer Functionality section would have a score of 4 since three different tones were created to indicate the object's distance from the user from all directions. The Power Source and PCB section shifted from a score of 2 to 2.5 because of the attempts to include sufficient casing made impossible by other constraints.

Costs

The cost to produce the Visually Assistive Hat was quite high, but we did manage to stay within the set budget. The overall cost breakdown for the project can be seen in Table 2. In addition to showing the cost breakdown for our project development, the table depicts the predicted costs if the device were to be manufactured in 10,000-unit quantities.

Table 2: Summary of Costs for Visually Assistive Hat

Item	Price for 1 Unit	Qty	Our Cost	Price for 10000 Units
MSP-EXP430FR2433	$11.99	3	$35.97	$119,900.00
PCB	$33	2	$66	$330,000

PCB Components + Assembly	$64.63	2	$129.26	$646,300.00
Sensors/Buzzers/Motors	$159.22	N/A	$159.22	$1,592,200.00
General Wire Assembly	$10	N/A	$10	$100,000
Hat	$20	1	$20	$200,000.00
Total	$298.84	N/A	$420.45	$2,988,400.00

As seen above, it can be noted that the development of the hat stayed within the set budget of $500. Additionally, if errors were not made throughout development, specifically considering the microcontrollers and PCBs, more than one quantity of these items would not have been required, thus further driving down costs.

Furthermore, based on the data in the table above the device was to be manufactured in 10,000-unit quantities, the total cost would be very expensive. However, the case depicted above does not take any discounts into account, so the total depicted is the absolute worst case that can be anticipated. Realistically, since the components will be bulk ordered, discounts for mass production and manufacturing would likely be applied, thus it is realistic to say that the cost of manufacturing in 10,000-unit quantities would be significantly less than the total depicted. A more detailed breakdown of the total costs spent throughout the development of the Visually Assistive Hat can be seen in Appendix D.

Future Work

One unforeseen difficulty of the project included the effectiveness of the buzzers. As mentioned previously, the buzzers can currently detect that there is an obstruction in the user's path (and chirp to let them know), however it cannot provide information on where the obstruction is coming from. This was due to the lack of additional timers that could be used on the current MSP, which does not allow for the separation of the two piezo buzzers to provide different feedback. For future work, the future team should investigate getting buzzers that can speak aloud and verbally communicate if there are obstructions i.e., by saying “Obstruction on your left.”

Another additional feature would include power cycling on all five sensors. The original design included MOSFETs for all three LiDAR sensors that would allow the power to each of them to be cut to conserve battery power, however upon implementation it was discovered that while the five-volt voltage regulator was indeed outputting the correct voltage, after this passed through the MOSFETs there was a voltage drop of about 0.2 volts due to internal resistance. This meant that each sensor would only be supplied with 4.8 volts, which is outside of the safe operation range for them. In future designs, this would be something that should be fixed to ensure that each sensor is receiving the correct voltage and could be turned on and off as needed.

Another place that future teams should improve upon is the battery. While we were happy with the amount of time the battery could run for, it was too heavy and clunky. Additionally, it was difficult to charge and required removal from the device to be recharged. Also, there is currently no indication for when the battery reaches low power, which could pose

as a danger for the user. To fix this, we would recommend future teams add some sort of indication for whether the battery is low on power. For future work, a different battery would be recommended to conserve space, allow the user to charge while it is still inside the hat, notify them of low battery, and make the design lighter and sleeker overall.

It was also noticed that the sensors, at times, would fail to pick up on obstructions that were shorter than the user. For example, if a shorter person were to walk past the user while they are wearing the hat, at times, the sensors would fail to pick up this obstruction and alert the user. This is with the exception of the incline sensor, which can work in tandem to notify the user of shorter obstructions in front of them. Therefore, in future work, the team should reevaluate the sensors layout in a manner that will detect shorter obstructions.

In addition to the other recommendations for future work, our final recommendation is to pay more attention to the professionalism of the design. This includes having a better system for the wires, including threading them under the fabric of the hat to hide them better, adding more buttons for user friendliness, and a darker colored hat in order to hide some of the sensors better.

References

[1]"MSP430G2553 | MSP430G2x/i2x | MSP430 ultra-low-power MCUs | Description & parametrics." [Online]. Available: http://www.ti.com/product/MSP430G2553. [Accessed: 06-Dec-2016].

[2] "Burden of Vision Loss | CDC," Jun. 17, 2020. https://www.cdc.gov/visionhealth/risk/burden.htm (accessed Sep. 09, 2021).

[3] "Visual impairment, blindness cases in U.S. expected to double by 2050," National Institutes of Health (NIH), May 19, 2016. https://www.nih.gov/news-events/news-releases/visual-impairment-blindness-cases-us-expected-double-2050 (accessed Sep. 09, 2021).

[4] E. National Academies of Sciences et al., The Impact of Vision Loss. National Academies Press (US), 2016. Accessed: Sep. 09, 2021. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK402367/

[5] W. Elmannai and K. Elleithy, "Sensor-Based Assistive Devices for Visually-Impaired People: Current Status, Challenges, and Future Directions," Sensors (Basel), vol. 17, no. 3, p. 565, Mar. 2017, doi: 10.3390/s17030565.

[6] I. Ulrich and J. Borenstein, "The GuideCane-applying mobile robot technologies to assist the visually impaired," IEEE Trans. Syst., Man, Cybern. A, vol. 31, no. 2, pp. 131–136, Mar. 2001, doi: 10.1109/3468.911370.

[7] M. H. A. Wahab et al., "Smart Cane: Assistive Cane for Visually-impaired People," arXiv:1110.5156 [cs], Oct. 2011, Accessed: Sep. 09, 2021. [Online]. Available: http://arxiv.org/abs/1110.5156

[8] "iGlassesTM Ultrasonic Mobility Aid," Ambutech. https://ambutech.com/products/iglasses%e2%84%a2-ultrasonic-mobility-aid (accessed Sep. 09, 2021).

[9] J. B. F. van Erp, L. C. M. Kroon, T. Mioch, and K. I. Paul, "Obstacle Detection Display for Visually Impaired: Coding of Direction, Distance, and Height on a Vibrotactile Waist Band," Frontiers in ICT, vol. 4, p. 23, 2017, doi: 10.3389/fict.2017.00023.

[10] "Salus Health - How Do Bone Conduction Headphones Work?" https://www.salusuhealth.com/Pennsylvania-Ear-Institute/Events/News-Stories/How-Do-Bone-Conduction-Headphones-Work.aspx (accessed Sep. 09, 2021).

[11] "How to Use Multiple Ultrasonic Sensors | MaxBotix Inc." https://www.maxbotix.com/tutorials1/031-using-multiple-ultrasonic-sensors.htm (accessed Dec. 16, 2021).

[12] "Multisim," National Instruments. https://www.ni.com/enus/shop/software/products/multisim.html (accessed Sep. 13, 2020).

[13] "Ultiboard," National Instruments. https://www.ni.com/enus/shop/software/products/ultiboard.html (accessed Sep. 13, 2020).

[14] "VirtualBench," National Instruments. https://www.ni.com/enus/shop/hardware/products/virtualbench-all-in-one-instrument.html (accessed Dec. 08, 2020).

[15] "CCSTUDIO IDE, configuration, compiler or debugger | TI.com." https://www.ti.com/tool/CCSTUDIO?utm_source=google&utm_medium=cpc&utm_campaign=epd-der-null-code_composer-cpc-evm-google-wwe&utm_content=code_composer&ds_k=code+composer&DCM=yes&gclid=CjwKCAjw-ZCKBhBkEiwAM4qfF6nQVfq_7j2MkDIRgFLKRG7fooUPcRIqWo1IjQQjlCNAV5AsaQ-slhoCNEsQAvD_BwE&gclsrc=aw.ds (accessed Sep. 17, 2021).

[16] D. Walker, "The Pros and Cons Of Using a Rechargeable Battery," *The Battery Specialists*, Sep. 24, 2020. https://batteryspecialists.com.au/blogs/news/pros-and-cons-of-using-a-rechargeable-battery (accessed Sep. 17, 2021).

[17] A. Boyden, V. K. Soo, and M. Doolan, "The Environmental Impacts of Recycling Portable Lithium-Ion Batteries," presented at the 23rd CIRP Conference on Life Cycle Engineering, 2016, [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827116300701.

[18] "Waste Management on Earth - The Pros and Cons of E Waste Recycling," *Excess Logic*, Jul. 29, 2015. https://excesslogic.com/waste-management-on-earth-pros-and-cons-of-e-waste-recycling (accessed Sep. 17, 2021).

[19] M. Avakian, "E-Waste: An Emerging Health Risk," *National Institute of Environmental Health Sciences*, Feb. 2014. https://www.niehs.nih.gov/research/programs/geh/geh_newsletter/2014/2/spotlight/ewaste_an_emerging_health_risk_.cfm (accessed Sep. 17, 2021).

[20] "NEMA and IEC Enclosure Classifications." https://www.engineeringtoolbox.com/nema-iec-enclosure-standards-d_920.html.

[21] "Laser Standards and Classifications," Rockwell Laser Industries. https://www.rli.com/resources/articles/classification.aspx.

[22] “The History and Basics of IPC Standards: The Official Standards for PCBs,” All About Circuits, Oct. 20, 2017. https://www.allaboutcircuits.com/news/ipc-standards-the-official-standards-for-pcbs/.

[23] M. Barr, “Embedded C Coding Standard.” Barr Group, 2018.

[24] Texas Instruments. “MSP driver library.” ti.com. http://www.ti.com/tool /MSPDRIVERLIB (accessed Dec. 16, 2019).

[25] “GitHub Documentation.” https://docs.github.com/en (accessed Dec. 10, 2020).

[26] C. ROMAN, “SMART SENSOR CANE”

[27] R. Gassert *et al.*, “White cane with integrated electronic travel aid using 3D TOF sensor,” US8922759B2, Dec. 30, 2014 Accessed: Dec. 15, 2021. [Online]. Available: https://patents.google.com/patent/US8922759B2/en?q=electronic+travel+aids&oq=electronic+travel+aids

[28] M. MAHADEVAPPA, J. MUKHOPADHYAY, and B. S. SUBHASHRAO, “Venucane: an electronic travel aid for visually impaired and blind people,” AU2012317177B2, May 04, 2017 Accessed: Dec. 16, 2021. [Online]. Available: https://patents.google.com/patent/AU2012317177B2/en?q=electronic+travel+aid&oq=electronic+travel+aid

[29] “UM10204 I2C-bus specification and user manual,” vol. 2014, p. 64, 2014.

[30] B. Ryerson, *TFMini-Plus-I2C*. 2021. Accessed: Dec. 16, 2021. [Online]. Available: https://github.com/budryerson/TFMini-Plus-I2C

Appendix

Appendix A

Schematics for Hardware Design

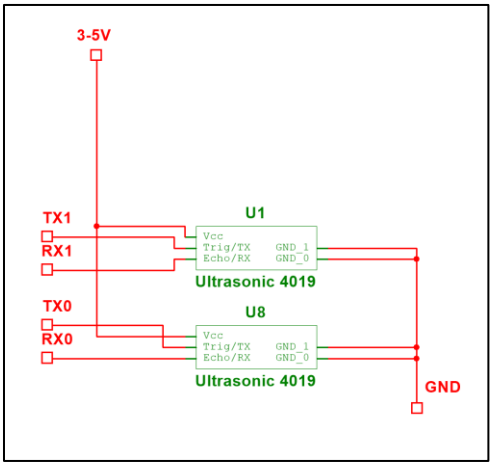


Figure 20: Ultrasonic Schematic

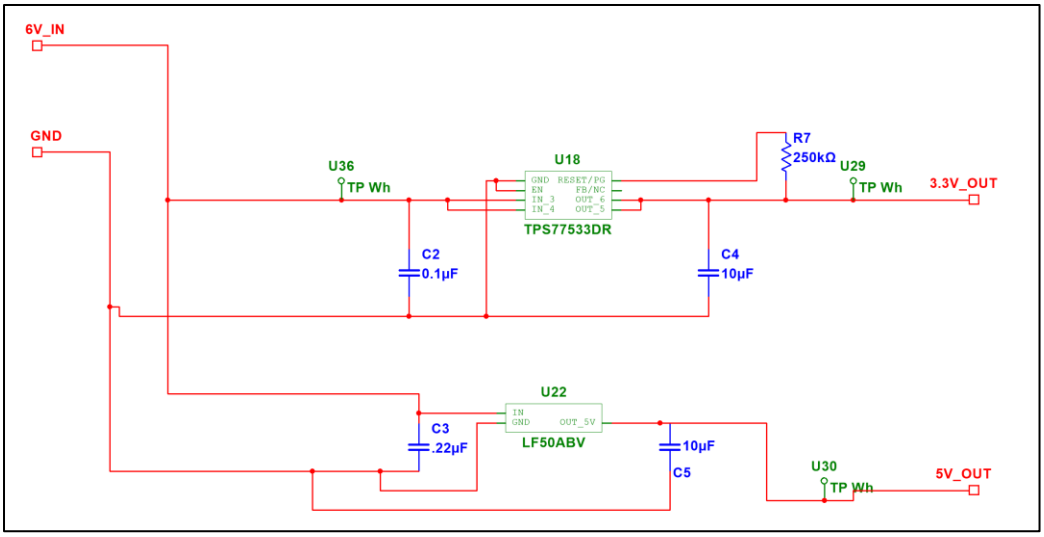


Figure 21: Voltage Regulator Schematic

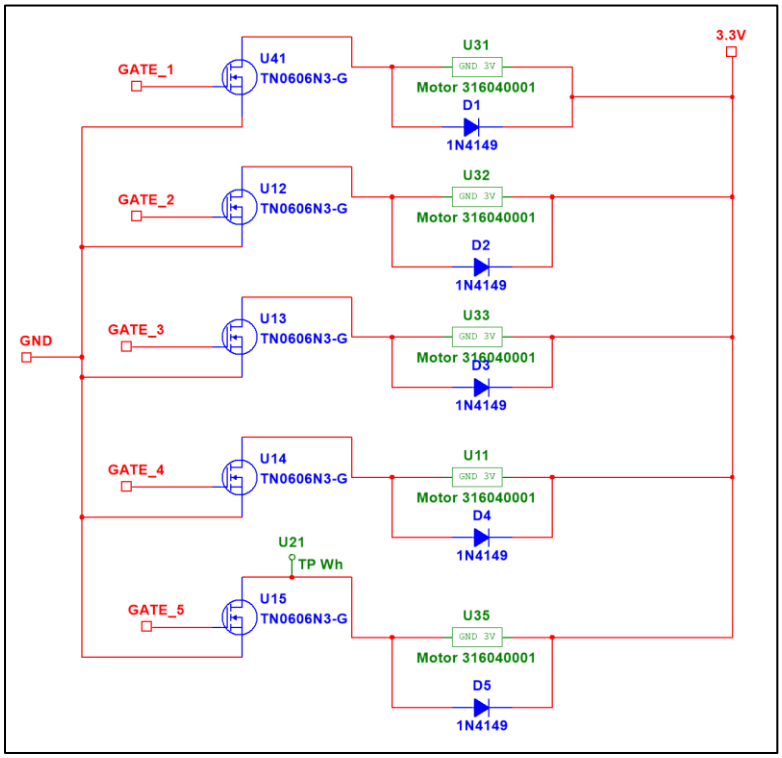


Figure 22: Motor Schematic

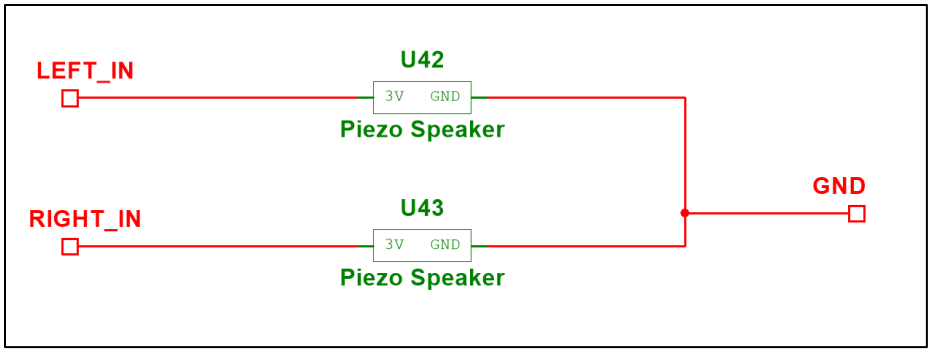


Figure 23: Buzzer Schematic

Appendix B

Software for the Visually Assistive Hat

Task Scheduler

```
// Task table layout:
//    task function            counter   period                FSM object
TaskType tasks[] = {
     {&button_task,                0,       1,     (fsm_type_t *) &button},
     {&uart_task,                  0,       4,     (fsm_type_t * ) &right_sensor},
     {&uart_task,                  0,       4,     (fsm_type_t * ) &left_sensor},
     {&lidar_task,                 0,       2,     (fsm_type_t *) &front_sensor},
     {&lidar_task,                 0,       2,     (fsm_type_t *) &incline_sensor},
     {&lidar_task,                 0,       2,     (fsm_type_t *) &back_sensor},
     {&motor_task_lidar,           0,       3,     (fsm_type_t *) &front_motor},
     {&motor_task_lidar,           0,       3,     (fsm_type_t *) &back_motor},
     {&motor_task_ultrasonic,      0,       3,     (fsm_type_t *) &right_motor},
     {&motor_task_ultrasonic,      0,       3,     (fsm_type_t *) &left_motor},
     {&motor_task_lidar,           0,       3,     (fsm_type_t *) &top_motor},
     {&speaker_task,               0,       7,     (fsm_type_t *) &speaker_manager}
};
```

```
// Task table that is being cycled through (see main.c)
extern TaskType tasks[NUMOFTASKS];

/*
 * Description:
 * This function cycles through each of the tasks defined in the main.c file.
 * Each of these tasks either checks the sensor data, triggers the motors or
 * the speakers. These tasks are executed at various frequencies based on the
 * counter that is placed in the task table (see main.c)
 *
 * Called In:
 * main.c
 *
 * Parameters:
 * None
 *
 * Return Value:
 * None
 */
void task_scheduler_isr(void){

#if DEBUG
    uint8_t counter,period;
#endif

    int i;
    for(i = 0;i < NUMOFTASKS; i++)
    {

#if DEBUG
        // used for testing
        counter = tasks[i].cycle_counter;
        period = tasks[i].task_execution_period;
#endif


        tasks[i].cycle_counter++;
        // Update counter and execute task if applicable
        if(tasks[i].cycle_counter >= tasks[i].task_execution_period)
        {
            (*tasks[i].task) (tasks[i].fsm);
            tasks[i].cycle_counter = 0;
        }


    }
}
```

I^2C

```
/*
 * receive_buffer: Buffer used to receive data in the ISR
 *
 * rx_byte_ctr: Number of bytes left to receive
 *
 * rx_index: The index of the next byte to be received in receive_buffer
 *
 * transmit_buffer: Buffer used to transmit data in the ISR
 *
 * tx_byte_ctr: Number of bytes left to transfer
 *
 * tx_index: The index of the next byte to be transmitted in transmit_buffer
 *
 * cmd_flag: Indicates whether data being sent is a command to the LiDAR sensor
 *     cmd_flag = 1 means writing data to sensor
 *     cmd_flag = 0 means reading data from sensor
 *
 * rx_done: Indicates when the recieving of data is done
 *     rx_done = 1 means all data recieved (9 bytes)
 *     rx_done = 0 means not all the data has been recieved
 *
 *
 */
uint8_t receive_buffer[MAX_BUFFER_SIZE] = {0};
uint8_t rx_byte_ctr = 0;
uint8_t rx_index = 0;
uint8_t transmit_buffer[MAX_BUFFER_SIZE] = {0};
uint8_t tx_byte_ctr = 0;
uint8_t tx_index = 0;
uint8_t cmd_flag = 0;
uint8_t rx_done = 0;


// Command defined in LiDAR datasheet that requests data in cm
uint8_t get_lidar_cm_cmd[] = {0x5A,0x05,0x00,0x01,0x60};
```

```
/*
 * NOTE FOR RETURNED DATA FROM SENSOR:
 *
 * LiDAR 9-byte frame returned when requesting distance
 *
 * byte 0 - 0x59, frame header, same for each frame
 * byte 1 - 0x59, frame header, same for each frame
 * byte 2 - Dist_L distance value low 8 bits
 * byte 3 - Dist_H distance value high 8 bits
 * byte 4 - Strength_L low 8 bits
 * byte 5 - Strength_H high 8 bits
 * byte 6 - Temp_L low 8 bits
 * byte 7 - Temp_H high 8 bits
 * byte 8 - Checksum is the lower 8 bits of the cummulative sum of the numbers of
 * the first 8 bytes
 *
 */

/*
 * Description:
 * Copies over the contents of a uint8_t array to another.
 *
 * Parameters:
 * source - pointer to the array which we want to copy from
 * dest - pointer to the array we want to copy to
 * count - number of bytes to copy over
 *
 * Return:
 * None
 */
void copy_array(uint8_t *source, uint8_t *dest, uint8_t count)
{
    uint8_t copyIndex = 0;
    for (copyIndex = 0; copyIndex < count; copyIndex++)
    {
        dest[copyIndex] = source[copyIndex];
    }
}
```

```
/*
 * Description:
 * Initializes the I2C module on the MSP. This function connects the module to
 * the SMCLK and sets the data rate to 400 kbps. There is no automatic stop
 * sent and there is not byte count threshold set. The I2C operates using the
 * interrupts associated with the module (ISR located at the bottom of this
 * file)
 *
 * Called In:
 * main.c
 *
 * Parameters:
 * None
 *
 * Return:
 * None
 */
void init_i2c()
{

    // disable the I2C to begin
    EUSCI_B_I2C_disable(EUSCI_B0_BASE);


    // Declaring Settings for I2C Using Driver Library structure
    EUSCI_B_I2C_initMasterParam param = {0};

    param.selectClockSource = EUSCI_B_I2C_CLOCKSOURCE_SMCLK;
    param.i2cClk = CS_getSMCLK();
    param.dataRate = EUSCI_B_I2C_SET_DATA_RATE_400KBPS;
    param.byteCounterThreshold = 0;
    param.autoSTOPGeneration = EUSCI_B_I2C_NO_AUTO_STOP;

    // Initialize using the param structure
    EUSCI_B_I2C_initMaster(EUSCI_B0_BASE, &param);

    // setting the initial slave address (default is front sensor for now,
    // changed later)
    EUSCI_B_I2C_setSlaveAddress(EUSCI_B0_BASE, LIDAR_FRONT_ADDR);

    //Set Master in transmit mode (changes throughout code)
    EUSCI_B_I2C_setMode(EUSCI_B0_BASE, EUSCI_B_I2C_TRANSMIT_MODE);
```

```
    //Enable I2C Module to start operations
    EUSCI_B_I2C_enable(EUSCI_B0_BASE);

    // Clear all possible interrupts for I2C
    EUSCI_B_I2C_clearInterrupt(EUSCI_B0_BASE,
                EUSCI_B_I2C_RECEIVE_INTERRUPT0 +
                EUSCI_B_I2C_TRANSMIT_INTERRUPT0 +
                    EUSCI_B_I2C_NAK_INTERRUPT
                );

    //Enable master Transmit, Receive, and NAK interrupt
    EUSCI_B_I2C_enableInterrupt(EUSCI_B0_BASE,
                    EUSCI_B_I2C_RECEIVE_INTERRUPT0 +
                    EUSCI_B_I2C_TRANSMIT_INTERRUPT0 +
                    EUSCI_B_I2C_NAK_INTERRUPT
                  );

    // Set I2C pins to correct peripheral mode
    GPIO_setAsPeripheralModuleFunctionInputPin(
        GPIO_PORT_P1,
        GPIO_PIN2 + GPIO_PIN3,
        GPIO_PRIMARY_MODULE_FUNCTION
    );


}


/*
 * Description:
 * Sends the get_lidar_cm_cmd over the I2C bus to the LiDAR sensor and then
 * requests the data from the sensor. This function includes other functions
 * from the TI Driver Library (MSP430 Ware) for the MSP430FR2433 to set the
 * slave address, set the mode, and then starting the data transmission
 * sequence.
 *
 * Called In:
 * main.c (lidar_task)
 *
 * Parameters:
 * slave_addr - the salve address for the desired sensor to collect data from
 *
```

```
 * Return:
 * unsigned 16-bit value of the distance in centimeters
 */
uint16_t get_lidar_data_cm(uint8_t salve_addr)
{
    // variables that track the number of receive attempts made by the MSP
    // (limited to 5 max send receive attempts before quitting)
    uint8_t rx_attempts = 0;
    uint8_t rx_max_attempts = 5;
    uint16_t dist = 0;

    // Setting the slave address for transmission
    EUSCI_B_I2C_setSlaveAddress(EUSCI_B0_BASE, salve_addr);

    // Set Master in transmit mode
    EUSCI_B_I2C_setMode(EUSCI_B0_BASE, EUSCI_B_I2C_TRANSMIT_MODE);

    // Load TX byte counter (5 bytes sent for command)
    tx_byte_ctr = 5;

    // This is a command so set flag to 1
    cmd_flag = 1;

    // Reset TX buffer index to 0
    tx_index = 0;

    // Load command into the transmit buffer
    copy_array((uint8_t *)&get_lidar_cm_cmd,(uint8_t *) &transmit_buffer,
               sizeof(get_lidar_cm_cmd));

    // Wait for stop bit to be set before sending next data
    while (EUSCI_B_I2C_SENDING_STOP == EUSCI_B_I2C_masterIsStopSent
                   (EUSCI_B0_BASE));

    // Send Start bit for command transmission
    EUSCI_B_I2C_masterSendStart(EUSCI_B0_BASE);

    __delay_cycles(100);

    // Expecting 9 bytes back from get_dist_cm command
    rx_byte_ctr= 9;
```

```
    // This is a read, not a command
    cmd_flag = 0;

    // Reset Index to start of buffer
    rx_index = 0;

    // RX is just starting so it is not done
    rx_done = 0;

    //Set Master in receive mode
    EUSCI_B_I2C_setMode(EUSCI_B0_BASE,
        EUSCI_B_I2C_RECEIVE_MODE
        );

    // wait for stop bit to be set before sending next data
    while (EUSCI_B_I2C_SENDING_STOP == EUSCI_B_I2C_masterIsStopSent
                            (EUSCI_B0_BASE));

    // Send Recieve Start bit
    EUSCI_B_I2C_masterReceiveStart(EUSCI_B0_BASE);

    // wait for the RX to be done
    while((!rx_done) && (rx_attempts < rx_max_attempts)){
        rx_attempts++;
        __delay_cycles(50);
    }

    // If max rx attempts reached, just send 0 as retrieved value
    if(rx_attempts >= rx_max_attempts)
    {
        return 0;
    }else
    {
        dist = ((uint16_t)(receive_buffer[3]<<8) |
                (uint16_t)(receive_buffer[2]));
        return dist;
    }


}
```

```
/*
 * Description:
 * This is the ISR that conducts the I2C transmission for both sending and
 * receiving data. This is normally initialized whenever the
 * EUSCI_B_I2C_masterReceiveStart() or EUSCI_B_I2C_masterSendStart() function
 * is used.
 *
 */
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=USCI_B0_VECTOR
__interrupt
#elif defined(__GNUC__)
__attribute__((interrupt(USCI_B0_VECTOR)))
#endif
void USCIB0_ISR(void)
{
    uint8_t rx_data;
    switch(__even_in_range(UCB0IV, USCI_I2C_UCBIT9IFG))
    {
        case USCI_NONE:             // No interrupts break;
            break;
        case USCI_I2C_UCALIFG:      // Arbitration lost
            break;
        case USCI_I2C_UCNACKIFG:    // NAK received (master only)

            // Resend START if NAK'd
            if(cmd_flag == 1)
            {
                EUSCI_B_I2C_masterSendStart(EUSCI_B0_BASE);

            }
            else if(cmd_flag == 0)
            {
                EUSCI_B_I2C_masterReceiveStart(EUSCI_B0_BASE);

            }


            break;
        case USCI_I2C_UCSTTIFG:     // START condition detected with own
                                    // address (slave mode only)
```

```
    break;
case USCI_I2C_UCSTPIFG:     // STOP condition detected (master &
                            // slave mode)
    break;
case USCI_I2C_UCRXIFG3:     // RXIFG3
    break;
case USCI_I2C_UCTXIFG3:     // TXIFG3
    break;
case USCI_I2C_UCRXIFG2:     // RXIFG2
    break;
case USCI_I2C_UCTXIFG2:     // TXIFG2
    break;
case USCI_I2C_UCRXIFG1:     // RXIFG1
    break;
case USCI_I2C_UCTXIFG1:     // TXIFG1
    break;
case USCI_I2C_UCRXIFG0:     // RXIFG0

    // Get RX Data and load into buffer
    rx_data = EUSCI_B_I2C_masterReceiveSingle(EUSCI_B0_BASE);

    receive_buffer[rx_index] = rx_data;

    // Check to see if byte read limit is reached
    if(rx_index >= rx_byte_ctr)
    {

        rx_index = 0;

        EUSCI_B_I2C_masterReceiveMultiByteStop(EUSCI_B0_BASE);

        rx_done = 1;
    }else
    {
        // Increment index to store next value in empty buffer space
        rx_index++;
    }
    break;

case USCI_I2C_UCTXIFG0:     // TXIFG0
```

```
            // Check TX byte counter
            if (tx_byte_ctr)
            {
                // Send next byte
                EUSCI_B_I2C_masterSendMultiByteNext(EUSCI_B0_BASE,
                    transmit_buffer[tx_index]);

                // Decrement TX byte counter
                tx_byte_ctr--;

                // Increment index to send next byte in buffer next time around
                tx_index++;
            }
            else
            {
                EUSCI_B_I2C_masterSendMultiByteStop(EUSCI_B0_BASE);
                tx_index = 0;
            }
            break;
        case USCI_I2C_UCBCNTIFG:  // Byte count limit reached (UCBxTBCNT)
            break;
        case USCI_I2C_UCCLTOIFG:  // Clock low timeout - clock held low too long
            break;
        case USCI_I2C_UCBIT9IFG:  // Generated on 9th bit of a transmit
                                  // (for debugging)
            break;
        default:
            break;
    }
}
```

UART

```
// Initializes UART modules to run at 9600 baud rate
/*
 * Description:
 * This function initializes the corresponding UART module to a baud rate of
 * 9600 with no parity bit, LSB first, and one stop bit. Total of 10 bits sent (start,
 * 8-bits of data, stop). This function assumes that the SMCLK is running at 1MHz for
 * baud rate calculations and set up.
 *
 * Called In:
 * ultrasonic_init()
 *
 * Parameters:
 * rx_pin - bit mask for the rx pin
 * tx_pin - bit mask for the tx pin
 * port_num - port number where the rx/tx pins are (used for pin initialization)
 * uart_base - module base value to initialize the UART module
 *
 * Return Value:
 * None
 */
void init_uart(uint8_t rx_pin, uint8_t tx_pin, uint8_t port_num,
               uint16_t uart_base)
{

    // Setting TX on UART Module
     GPIO_setAsPeripheralModuleFunctionOutputPin(
         port_num,
         tx_pin,
         GPIO_PRIMARY_MODULE_FUNCTION
     );

     // Setting RX on UART Module
     GPIO_setAsPeripheralModuleFunctionInputPin(
         port_num,
         rx_pin,
         GPIO_PRIMARY_MODULE_FUNCTION
     );
```

```
    //Configure UART
    //SMCLK = 1MHz, Baudrate = 9600
    //UCBRx = 8, UCBRFx = 0, UCBRSx = 0xD6, UCOS16 = 0
    EUSCI_A_UART_initParam param = {0};
    param.selectClockSource = EUSCI_A_UART_CLOCKSOURCE_SMCLK;
    param.clockPrescalar = 6;
    param.firstModReg = 8;
    param.secondModReg = 0x20;
    param.parity = EUSCI_A_UART_NO_PARITY;
    param.msborLsbFirst = EUSCI_A_UART_LSB_FIRST;
    param.numberofStopBits = EUSCI_A_UART_ONE_STOP_BIT;
    param.uartMode = EUSCI_A_UART_MODE;
    param.overSampling = EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION;

    // Initialize UART
    if (STATUS_FAIL == EUSCI_A_UART_init(uart_base, &param))
    {
        return;
    }

    // Enable UART communication
    EUSCI_A_UART_enable(uart_base);

    // Clear the RX interrupt
    EUSCI_A_UART_clearInterrupt(uart_base,
            EUSCI_A_UART_RECEIVE_INTERRUPT);

    // Clear TX interrupts
    EUSCI_A_UART_clearInterrupt(uart_base,
            EUSCI_A_UART_TRANSMIT_INTERRUPT);

    // Enable RX interrupts
    EUSCI_A_UART_enableInterrupt(uart_base,
            EUSCI_A_UART_RECEIVE_INTERRUPT);

    // Enable TX interrupts
    EUSCI_A_UART_enableInterrupt(uart_base,
                EUSCI_A_UART_TRANSMIT_INTERRUPT);

}
```

```
/*
 * Description:
 * This is the IRQ Handler for any RX interrupts triggered on the UART0
 * module (P1.5). This handles the interrupt by updating the structure
 * right_sensor defined in main() by adding in the most recent distance value
 * to it's cumulative sum and distance readigns buffer. Then it computes
 * the new average in addition to this for the output function to handle.
 *
 */
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=USCI_A0_VECTOR
__interrupt
#elif defined(__GNUC__)
__attribute__((interrupt(USCI_A0_VECTOR)))
#endif
void EUSCI_A0_ISR(void)
{
    switch(__even_in_range(UCA0IV,USCI_UART_UCTXCPTIFG))
    {
        uint16_t dist;
        case USCI_NONE: break;
        case USCI_UART_UCRXIFG:

            left_sensor.bytes_received++;
            if(left_sensor.bytes_received == 1)
            {
                left_sensor.dist_high_byte = EUSCI_A_UART_receiveData(
                        EUSCI_A1_BASE);

            }else if(left_sensor.bytes_received == 2)
            {
                // reset the bits recieved
                left_sensor.bytes_received = 0;

                // read the next byte available (lower byte)
                left_sensor.dist_low_byte = EUSCI_A_UART_receiveData(
                        EUSCI_A1_BASE);

                //compute the distance value to add to the buffer
                dist = ((uint16_t) left_sensor.dist_high_byte << 8) |
                        (uint16_t)left_sensor.dist_low_byte;
```

```
                // Update the cumulative sum
                left_sensor.cumulative_sum = left_sensor.cumulative_sum -
                        left_sensor.dist_buffer[left_sensor.buff_idx] + dist;

                // Now update the buffer with the new value
                left_sensor.dist_buffer[left_sensor.buff_idx] = dist;

                // Update the average now
                left_sensor.average = left_sensor.cumulative_sum
                        >> SENSOR_AVERAGING_SHIFT_AMNT;

                // Now update the index
                left_sensor.buff_idx++;

                if(left_sensor.buff_idx >= SENSOR_BUFFER_LENGTH)
                {
                    left_sensor.buff_idx = 0;
                }
            }

             break;
        case USCI_UART_UCTXIFG:
            left_sensor.bytes_received = 0;
            break;
        case USCI_UART_UCSTTIFG: break;
        case USCI_UART_UCTXCPTIFG: break;
    }
}
```

```
/*
 * Description:
 * This is the IRQ Handler for any RX interrupts triggered on the UART1
 * module (P2.5). This handles the interrupt by updating the structure
 * right_sensor defined in main() by adding in the most recent distance value
 * to it's cumulative sum and distance readigns buffer. Then it computes
 * the new average in addition to this for the output function to handle.
 *
 */
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=USCI_A1_VECTOR
__interrupt
#elif defined(__GNUC__)
__attribute__((interrupt(USCI_A0_VECTOR)))
#endif
void EUSCI_A1_ISR(void)
{
    uint16_t dist;
    switch(__even_in_range(UCA1IV,USCI_UART_UCTXCPTIFG))
    {
        case USCI_NONE: break;
        case USCI_UART_UCRXIFG:

            right_sensor.bytes_received++;
            if(right_sensor.bytes_received == 1)
            {
                right_sensor.dist_high_byte = EUSCI_A_UART_receiveData(
                        EUSCI_A1_BASE);

            }else if(right_sensor.bytes_received == 2)
            {
                // reset the bits recieved
                right_sensor.bytes_received = 0;

                // read the next byte available (lower byte)
                right_sensor.dist_low_byte = EUSCI_A_UART_receiveData(
                        EUSCI_A1_BASE);

                //compute the distance value to add to the buffer
                dist = ((uint16_t) right_sensor.dist_high_byte << 8) |
                        (uint16_t)right_sensor.dist_low_byte;
```

```
                // Update the cumulative sum
                right_sensor.cumulative_sum = right_sensor.cumulative_sum -
                        right_sensor.dist_buffer[right_sensor.buff_idx] + dist;

                // Now update the buffer with the new value
                right_sensor.dist_buffer[right_sensor.buff_idx] = dist;

                // Update the average now
                right_sensor.average = right_sensor.cumulative_sum
                        >> SENSOR_AVERAGING_SHIFT_AMNT;

                // Now update the index
                right_sensor.buff_idx++;

                if(right_sensor.buff_idx >= SENSOR_BUFFER_LENGTH)
                {
                    right_sensor.buff_idx = 0;
                }
            }
            break;
        case USCI_UART_UCTXIFG:
            right_sensor.bytes_received = 0;
            break;
        case USCI_UART_UCSTTIFG: break;
        case USCI_UART_UCTXCPTIFG: break;
    }
}
```

Motors

```
/* Description:
 * Toggles the motor at intervals, with faster pulses happening at closer distances.
 * Turns off the motor if no data is detected.
 *
 * Called In:
 * motor_task_ultrasonic(fsm_type_t* fsm), main.c
 * motor_task_lidar(fsm_type_t* fsm), main.c
 *
 * Parameters:
 * motor - the motor being vibrated
 * data - the data received from the sensors.
 *
 * Return Value:
 * none
 */
void vibrate_task(motor_type_t* motor, uint16_t data)
{
    if (data == NO_FREQ)
    {
        // Turns off the pin if nothing is detected
        GPIO_setOutputLowOnPin(motor->port, motor->pin);
        motor->counter = 0;
        motor->is_off = 1;
    }
    else if ((data == SLOW_FREQ && motor->counter == 0) ||
            (data == MED_FREQ && motor->counter%4 == 0) ||
            (data == FAST_FREQ))
    {
        // An obstacle has been detected & the pins need to be toggled
        if (motor->is_off)
        {
            // Turns the pin on & sets is_off to false
            GPIO_setOutputHighOnPin(motor->port, motor->pin);
            motor->is_off = 0;
        }
        else
        {
            // Turns the pin off & sets is_off to true. Increments the counter
            GPIO_setOutputLowOnPin(motor->port, motor->pin);
            motor->is_off = 1;
            motor->counter = (motor->counter+ 1)%8;
        }
    }
    else
    {
        // An obstacle has been detected & the pins do not need to be toggled
        motor->counter = (motor->counter + 1)%8;
    }

}
```

```c
/*
 * Description:
 * Converts the given fsm value to the subclass of motor, and retrieves the
 * data from the related lidar sensor.
 *
 * Called In:
 * task_scheduler_isr(), main.c
 *
 * Parameters:
 * fsm - contains a motor variable, passed through by the fsm.
 *
 * Return Value:
 * None
 */
void motor_task_lidar(fsm_type_t* fsm){
    // WARNING: The fsm variable passed through must be of type motor_type_t
    motor_type_t *motor = (motor_type_t*) fsm;

    // WARNING: The sensor_ref variable associated with the motor must be of
    // type ultrasonic_type_t
    uint16_t data = ((lidar_type_t*)motor->sensor_ref)->average;
    uint16_t frequency;

    // WARNING: The sensor_ref variable associated with the motor must be of
    // type ultrasonic_type_t
    if(((lidar_type_t*)motor->sensor_ref)->address == LIDAR_INCLINE_ADDR){
        // Decodes the data using adjusted values for the incline sensor
        frequency = data_decoder_incline(motor->index, data);
    } else {
        // Performs the standard data decoding
        frequency = data_decoder(motor->index, data, 1);
    }

    vibrate_task(motor, frequency);

}
```

```c
/*
 * Description:
 * Simplifies the data into buckets based on detected obstruction distance.
 * Updates the speaker manager's proximity array.
 *
 * Called In:
 *  motor_task_ultrasonic(fsm_type_t* fsm), main.c
 *  motor_task_lidar(fsm_type_t* fsm), main.c
 *
 * Parameters:
 * index - the sensor's index in the speaker_manager array
 * data - the data received from the sensors.
 * is_lidar - 1 if the data comes from a lidar sensor, 0 if not
 *
 * Return Value:
 * uint16_t frequency - the decoded frequency relative to the data
 */
uint16_t data_decoder(uint8_t index, uint16_t data, uint8_t is_lidar)
{
    uint16_t frequency = NO_FREQ;

    if(setting == Inside)
    {
        // Compares the distances for close range obstacles for lidar or uart
        if (is_lidar)
        {
            // Compares the returned sensor distance with pre-defined distances and
            // returns a frequency indicative of nothing, close, medium, or far
            if (data < INSIDE_LIDAR_IGNORE_LOW) {  }
            else if (data < INSIDE_LIDAR_LOW) { frequency = FAST_FREQ; }
            else if (data < INSIDE_LIDAR_MID) { frequency = MED_FREQ; }
            else if (data < INSIDE_LIDAR_HIGH) { frequency = SLOW_FREQ; }
        }
        else
        {
            // Compares the returned sensor distance with pre-defined distances and
            // returns a frequency indicative of nothing, close, medium, or far
            if (data < INSIDE_UART_IGNORE_LOW) {  }
            else if (data < INSIDE_UART_LOW) { frequency = FAST_FREQ; }
            else if (data < INSIDE_UART_MID) { frequency = MED_FREQ; }
            else if (data < INSIDE_UART_HIGH) { frequency = SLOW_FREQ; }
        }
    }
    else
    {
        // Compares the distances for long range obstacles for lidar or uart
        if(is_lidar)
        {
            // Compares the returned sensor distance with pre-defined distances and
            // returns a frequency indicative of nothing, close, medium, or far
           if (data < OUTSIDE_LIDAR_IGNORE_LOW) {  }
           else if (data < OUTSIDE_LIDAR_LOW) { frequency = FAST_FREQ; }
           else if (data < OUTSIDE_LIDAR_MID) { frequency = MED_FREQ; }
           else if (data < OUTSIDE_LIDAR_HIGH) { frequency = SLOW_FREQ; }
       }
       else
       {
           // Compares the returned sensor distance with pre-defined distances and
           // returns a frequency indicative of nothing, close, medium, or far
           if (data < OUTSIDE_UART_IGNORE_LOW) {  }
           else if (data < OUTSIDE_UART_LOW) { frequency = FAST_FREQ; }
           else if (data < OUTSIDE_UART_MID) { frequency = MED_FREQ; }
           else if (data < OUTSIDE_UART_HIGH) { frequency = SLOW_FREQ; }
       }
    }

    //updates the speaker management array
    speaker_manager.speaker_frequencies[index] = frequency;
    return frequency;
}
```

```
/*
 * Description:
 * Simplifies the data into two buckets based on detected obstruction distance.
 * Updates the speaker manager's proximity array for the lidar task.
 *
 * Called In:
 * motor_task_lidar(fsm_type_t* fsm), main.c
 *
 * Parameters:
 * index - the sensor's index in the speaker_manager array
 * data - the data received from the sensors.
 *
 * Return Value:
 * uint16_t frequency - the decoded frequency relative to the data
 */
uint16_t data_decoder_incline(uint8_t index, uint16_t data){
    uint16_t frequency = NO_FREQ;

    if(is_calibrated)
    {
        // execute frequencies only once user height is ascertained
        if (user_height + HIGH_TOLERANCE <= data) { frequency = MED_FREQ; }
        else if (user_height - HIGH_TOLERANCE >= data) { frequency = MED_FREQ; }
        else if (user_height + LOW_TOLERANCE <= data) { frequency = SLOW_FREQ; }
        else if (user_height - LOW_TOLERANCE >= data) { frequency = SLOW_FREQ; }
    }

    //updates the speaker management array
    speaker_manager.speaker_frequencies[index] = frequency;
    return frequency;
}
```

Speakers

```
/*
 * Description:
 * Cycles through the frequency output for each sensor, as stored in the
 * speaker_manager struct. The CCR register for the PWM wave is updated.
 *
 * Called In:
 * fsm, main.c
 *
 * Parameters:
 * fsm - the speaker_manager struct passed through by the fsm
 *
 * Return Value:
 * None
 */
void speaker_task(fsm_type_t* fsm){
    speaker_manager_type_t *speaker_manager = (speaker_manager_type_t*) fsm;

    if(speaker_manager->speaker_is_on)
    {
        // update the index for the next speaker to be evaluated
        speaker_manager->speaker_frequencies_index =
                (((speaker_manager->speaker_frequencies_index)+1)%SPEAKER_BUFFER_LENGTH);

        // update the duty cycle to the tone of the direction being evaluated, with CCR1
        // as half the duty cycle of CCR0
        change_duty_cycle((speaker_manager->speaker_frequencies[speaker_manager->
                                                                speaker_frequencies_index]),
                          speaker_manager->ccr1_index, speaker_manager->ccr0_index);
    } else {
        change_duty_cycle(0, speaker_manager->ccr1_index, speaker_manager->ccr0_index);

    }
}
```

Appendix C
I^{2}C Verification

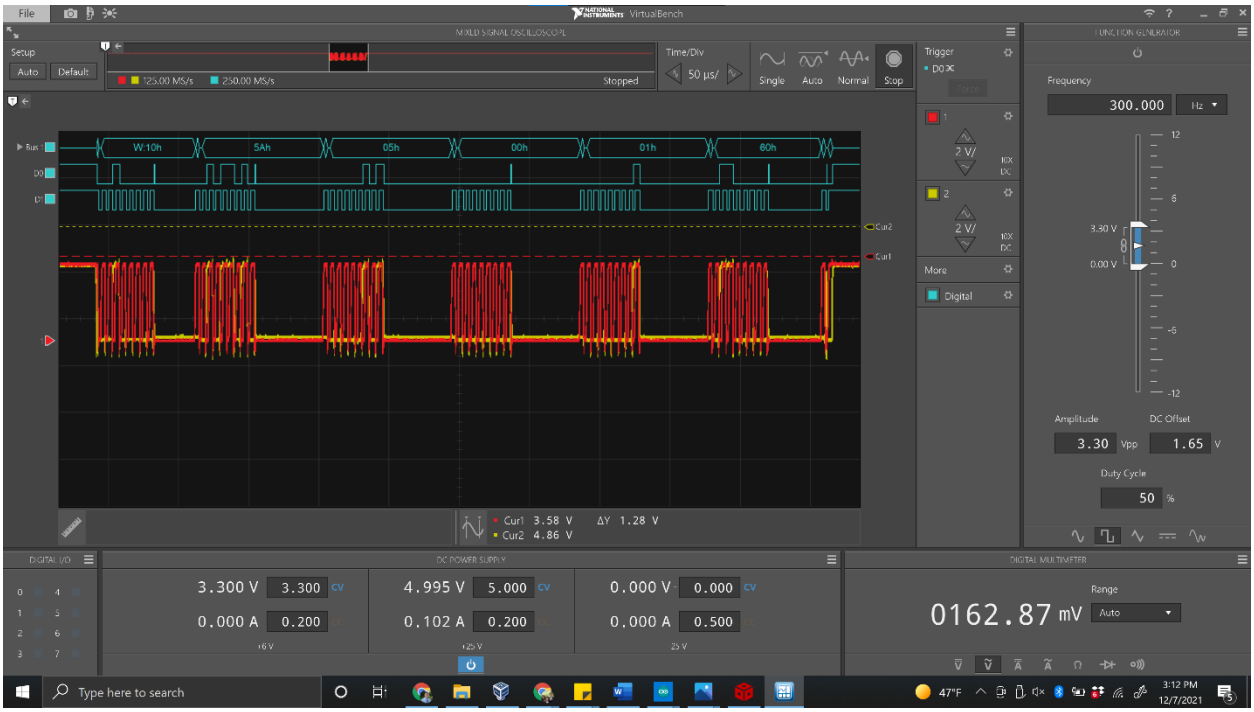


Figure 24: I^{2}C Command to LiDAR

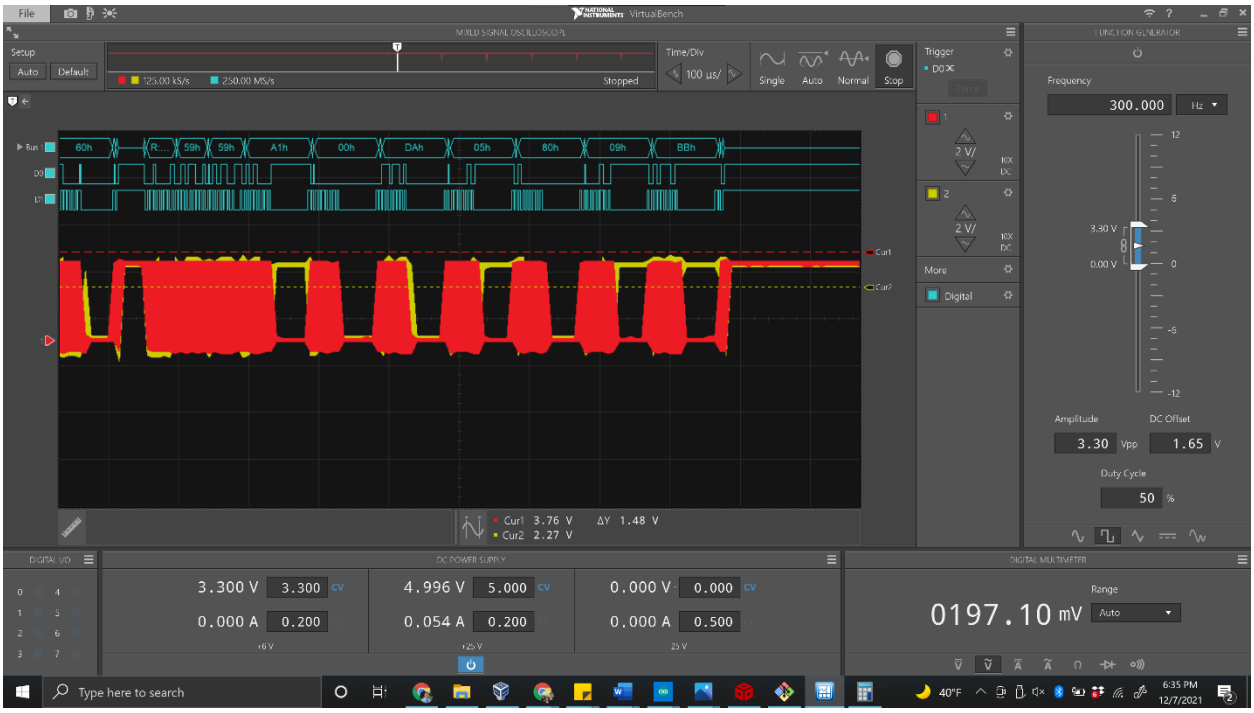


Figure 25: I^{2}C Read from LiDAR

Appendix D

Cost Breakdown

Team:	**Table In The Back + 1**
Budget	$500.00
Current Spent	$427.71
Remaining Balance	$72.29

Item	Price	Count	Total
Speaker - TP304003-2	$1.56	2	$3.12
LiDAR Distance Sensor - TFmini-S	$43.90	3	$131.70
Vibration Motors - 316040001	$1.20	5	$6.00
Ultrasonic Sensor - US-100 Ultrasonic	$6.95	2	$13.90
LiDAR Distance Sensor Cable	$1.50	3	$4.50
MSP-EXP430FR2433	$11.99	2	$23.98
5V Regulator	$1.49	1	$1.49
MOSFET transistor (N-Channel)	$1.00	5	$5.00
MOSFET transistor (P-Channel)	$0.57	3	$1.71
Diodes (for motors)	$0.13	5	$0.65
button 2 (adjustable hieght and cheaper)	$1.79	1	$1.79
3.3V REGULATOR	$3.09	1	$3.09
Booster Pack Connectors	$1.26	2	$2.52
5V NMOS for P-channel MOSFET	$1.32	3	$3.96
UClamp for static Protection	$0.27	1	$0.27
I2C LiDAR Connectors	$0.75	3	$2.25
Ultrasonic Connectors	$0.70	2	$1.40
test points	$0.34	13	$4.42
10k Ohm Resistors	$0.10	2	$0.20
1k Ohm Resistors	$0.10	3	$0.30
250k Resistor	$0.75	1	$0.75
.1uF Capacitor (Ceramic)	$0.23	1	$0.23
10uF Capacitor (Ceramic)	$0.50	1	$0.50
1uF Capacitor (Ceramic)	$0.32	2	$0.64
.01uF (10000pF - Ceramic)	$0.21	1	$0.21
Battery and 2-Wire Connectors	$0.19	10	$1.90
2 Wire Housing	$0.10	10	$1.00
22 Gauge Wire Contacts	$0.10	42	$4.20
i2c - lidar - Housing	$0.19	3	$0.57
22 awg wire 25'	$2.95	2	$5.90
uart - ultrasonic - Housing	$0.20	2	$0.40
6V Rechargeble Battery	$8.72	1	$8.72
6V Battery Recharger	$13.36	1	$13.36
10uF Capacitor	$0.50	3	$1.50
IC REG 5V TO220AB	$1.19	3	$3.57
.33uF Capacitor	$0.33	3	$0.99
.10uF Capacitor	$0.50	2	$1.00
IC REG 5V TO220AB	$1.19	1	$1.19

.22uF Capacitor	$0.33	1	$0.33
MOSTFET P-Ch TO243AA	$1.61	3	$4.83
1k Resistor	$0.10	3	$0.30
MOSTFET N-Ch TN0604N3-G (motors	$1.32	3	$3.96
Slotted Test points	$0.26	13	$3.38
4POS Connector (LiDAR)	$0.75	3	$2.25
IC REG 3.3V 8SOIC	$3.09	1	$3.09
.1uF Capacitor	$0.23	1	$0.23
250k Resistor	$0.75	1	$0.75
5POS Connector (Ultrasonic)	$0.77	2	$1.54
POS Connector (Battery, Motors, Buttor	$0.15	10	$1.48
MOSFET N-CH TN0606N3-G (LiDAR	$1.00	5	$5.00
Diode 1N4149	$0.13	5	$0.65
10,000 pF (.01uF) Capacitor	$0.21	1	$0.21
10k Resistor	$0.10	1	$0.10
UCLAMP	$0.70	1	$0.70
CONN HDR 20POS PCB	$1.26	2	$2.52
UART and LiDAR Clamps	$0.10	22	$2.20
MSP-EXP430FR2433	$11.99	1	$11.99
Black Wire	$6.98	1	$6.98
Red Wire	$5.27	1	$5.27
White Wire	$5.27	1	$5.27
Green Wire	$5.27	1	$5.27
Board Submission Fees	$33.00	2	$66.00
Board Population Fees	$6.00	2	$12.00
Hat	$20.00	1	$20.00
2 pin crimp connectors (part 1735801-1)	$0.10	50	$0.79
Extra crimps for 4 and 5 POS connectors	$0.10	25	1.74
Wire Assembly	$10.00	N/A	N/A

Appendix D: Hat Pictures