Ultrasonic Embedded System for Visually Impaired Persons to Enhance Situational Awareness

for
Dr. N. Xiong
CS-4233 Professional Development
Northeastern State University
Broken Arrow, Oklahoma

by
G. C. Holden

November 27, 2018

Table of Contents

Introduction 1
Background 1
Project Goals 1
Report Structure 2
Hardware Components Section 3
Embedded System Platforms 3
Raspberry Pi Platform 3
Arduino Platform 4
Platform Selection 4
Arduino 5
Prototype Arduino Mega Specifications 5
Embedded Arduino Pro Mini Specifications 6
Arduino IDE 7
Ultrasonic Sensors 8
Theory of Ultrasonic Ranging 8
HC-SR04 Ultrasonic Module Specifications 8
Micro Vibration Motors 9
Micro Vibration Motor Specifications 9
Micro Vibration Motor Operation 9
Motor Current and Arduino Protection. 10
Software Development Section 12
User Experience 12
Acceptable Detection Rate 12
Intuitive Haptic Feedback 12
Coding of Ultrasonic Sensors 13
Basics of Communication 13
Software Improvement of Detection Accuracy 13
Simultaneous Operation 14
Portability 17
Coding of Micro Vibration Motor 17
Software Control of Motor 17
Non-Linear Intensity 17
Code Walk-Through Section 19
Conclusion 21
Contributions 21
Limitations 21
Potential Enhancements to System 21
Biography 22
References 23
Appendix A: Code of Ultrasonic Embedded System A-1

Figures and Tables

Figure 1 Raspberry Pi board 3
Figure 2 Arduino Mega style board 5
Figure 3 Arduino Pro Mini board 6
Figure 4 Arduino IDE Web Interface 7
Figure 5 Ultrasonic Ranging Technique 8
Figure 6 HC-SR04 Dimensions and Detection Angle 9
Figure 7 Haptic Motor Diagram 10
Figure 8 Protection Circuit Schematic 11
Figure 9 Chart of Cyclical Problem Reading Distances with NewPing library 12
Figure 10 HC-SR04 Communication Sequence 13
Figure 11 Chart of Readings Using NewPing Library 14
Figure 12 Chart of Readings Using Custom Code 14
Figure 13 Pin Mapping of Atmega168 15
Figure 14 Bitmasks and Logic for Simultaneous Setting of Trigger Pins 16
Figure 15 Bitmasks and Logic for Simultaneous Reading of Echo Pins 16

Introduction

Background. Embedded systems are becoming a part of everyday life, from front door locks that can be opened with a cell phone to refrigerators that can notify owners when the milk is low. While the Internet-of-Things (IoT) offers conveniences to the masses and is therefore the most commonly thought of application of embedded systems, similar embedded systems also have the potential to be used for much more serious purposes such as human sensory replacement.

Replacement human sensory input research and development is ongoing at a very technical level. For example, cochlear implants are already used to directly stimulate nerves related to audio signaling to the brain and allow deaf users to be more aware of sound-producing events. Unfortunately, the cost for a cochlear implant can run close to $100,000.00, making it difficult for persons to access this technology. Even worse, there is currently no standard medical technology to assist persons with severe vision loss, leaving no help for people who need visual assistance.

This all-or-nothing situation does not have to be the only solution, though. Commonly available electronics can be used to fill these sensory gaps very inexpensively. In addition, once sensory enhancement projects are created and published to public repositories, the devices can be re-created by novice users. In this way, the community can join forces to provide assistance to persons around the globe.

Project Goals:

- Develop a portable embedded system for use by persons with extremely limited vision to augment the use of a tapping cane.
- Use ultrasonic sensors and a haptic feedback motor to alert users to the presence of objects and guide users away from obstacles which are not yet in the range of a cane and would otherwise be undetectable without vision.
- Use multiple ultrasonic sensors to improve performance and range.
- Determine an acceptable means of translating ranging information into intuitive feedback using only a haptic vibration motor.
- Use inexpensive and widely available components to make system components easy to source.
- Distribute instructions and software for free on the internet enabling anyone to re-create the system or improve upon it.

Report Structure. This report aims to give a basic understanding of the major hardware components used in this project as well as the design process of the software developed.

The first section is the hardware components section. It discusses choice of development platform, differences of prototyping and embedding boards, and how to program the boards. It goes on to explain how ultrasonic sensors function, specifications of the sensors used, specifications of the motor used, operation of motors, and what protections are needed when a motor is used in an electronic circuit.

The second section is the software development section. It explains the role software plays in providing acceptable detection rate from the sensors and creating intuitive feedback using only the motor. The explanation of ultrasonic sensor communication is given, and use of the built-in pulseIn function is discussed, as well as the need for custom drivers in order to use two sensors at the same time. Use of hard coded registers and bitmasks to control the sensor is described, but the need for portability requires a slightly different method which is described here. Finally, the software control of the motor and means for providing information from the motor is covered.

The third section is the code walk-through section which provides a high-level pseudo code description of the logic used in operating the sensors and the motor. The actual code is provided in Appendix A.

The fourth section is the conclusion which discusses the contributions of the project. The limitations of the system and potential enhancements are also included in this section.

Hardware Components Section

Embedded System Platforms

The first step of designing an embedded system is choosing an appropriate platform on which to develop and implement the system. There are two main choices for the platform of a project meant to be widely distributable and easily replicated: Raspberry Pi, and Arduino. They are equally good but have considerable differences in their abilities and uses. This section gives an explanation of both, and the selection factors used in this project.

Raspberry Pi Platform. A Raspberry Pi 3 B+ is technically a fully functioning computer, but has the footprint of a credit card. It uses a standard keyboard / mouse for input and has a quad core CPU, 1GB of memory, Bluetooth, Wireless LAN, 4 USB ports, HDMI out, microSD card slot, 40 GPIO pins, and a full operating system such as Raspian Linux.

The typical cost of the current Raspberry Pi model is close to $40, and there have been times when they are hard to come by. Projects with several programs running, heavy computations, a large amount of I/O, or a need for Ethernet communication would be well served by a Raspberry Pi.

Figure 1 – Raspberry Pi board. *Source:* https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus

Arduino Platform. Where the Raspberry Pi is a miniaturized computer system, Arduino is an enhanced microcontroller chip. It has no built in methods of I/O except writing to the serial monitor while connected to a computer. Even though the chip must be programmed with machine code, the Arduino IDE makes this easy by allowing high level C++ based code to be compiled and converted to the commands the microprocessor uses. Arduino is meant to do one set of tasks repeatedly without necessarily needing any action on the part of the user.

An Arduino brand prototyping / development board can cost between $18 and $38 depending on the microcontroller and number of I/O pins, but generic versions which are fully compatible are much cheaper. In addition, once a design is finalized, small boards such as the Arduino Pro Mini or compatibles can be purchased for less than $5 and left in the embedded system.

One of the biggest advantages of Arduino is its support community. The ‘about’ page of Arduino.cc states “Arduino is the first widespread Open Source Hardware project and was set up to build a community that could help spread the use of the tool and benefit from contributions from hundreds of people who helped debug the code, write examples, create tutorials, supports other users on the forums and build thousands of groups around the globe.” It goes on to say, “Arduino has become the number one choice for electronics makers, especially for developing solutions for the IoT marketplace, which has been predicted to become a $6 trillion market by 2021.”

Platform Selection. An Arduino was selected as the platform for development of this project due to many factors. Top considerations were that it is inexpensive, extensively documented, readily available, and open source.

In addition, libraries have been developed to support auxiliary sensors and peripheral devices in order to make communication between them and the Arduino quick and easy. These drivers cover a vast assortment of sensors and peripherals, making nearly any project possible with very limited, or possibly even no knowledge of programming. When needed, though, custom drivers can be written because the Arduino architecture does allow low level access to registers and specific bits.

More importantly, Arduino can easily be set up to ensure the end-user does not require any experience or special knowledge to use the system. This is possible because programs which are written to the microprocessor controller on an Arduino board are run automatically on power-up, removing any need for the extra resource overhead and possible complications of an operating system. This end-user ease of use was the single most important factor in choosing Arduino for this project.

Based on the selection of Ardino for the platform, the hardware used in this project consists of four main components: an Arduino mega for prototyping, an Arduino Pro Mini for embedding into system, an HC-SR04 ultrasonic module for distance ranging to obstacles, and a haptic motor to provide feedback on obstacle location and distance. This section discusses details about these hardware components.

Arduino

Prototype Arduino Mega Specifications.

The Seeedstudio Mega used for prototyping this project consists of:

Microcontroller	ATmega2560
Clock Speed	16 MHz
Flash Memory	256 KB
SRAM	8 KB
EEPROM	4 KB
Recommended Input Voltage	7-12V
Min-Max Input Voltage	6-20V
Operating Voltage	5V
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Total Number of Digital I/O Pins	54
Number of PWM Capable Digital I/O Pins	14
Analog Input Pins	16

Use of a 5V Arduino board is required to safely receive the signal from the HC-SR04 ultrasonic device's echo pin without damaging the microcontroller, and also to drive the ultrasonic sensor's transmitters. Although this particular prototyping board has capabilities to interface with Android devices, that feature was not utilized for this project.

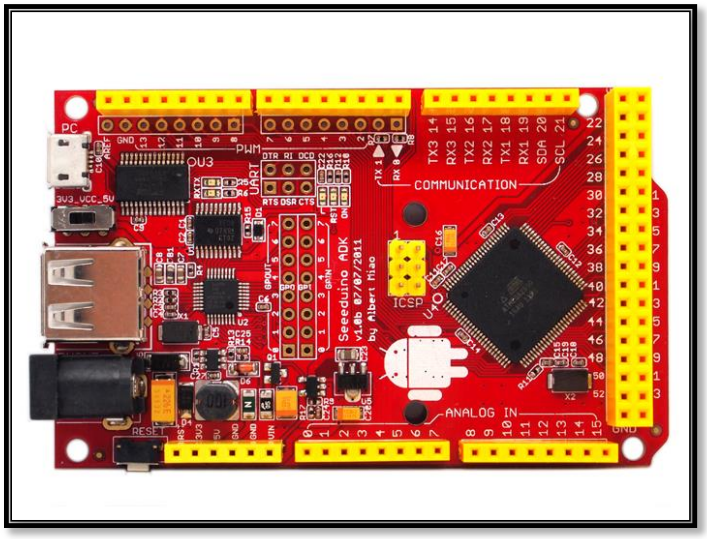

Figure 2 – Arduino Mega style board. *Source:* https://www.seeedstudio.com/Seeeduino-ADK-Main-Board-p-846.html

Embedded Arduino Pro Mini Specifications.

The Pro Mini embedded into the final system consists of:

Microcontroller Atmega328p
Clock Speed ..... 16 MHz
Flash Memory ..... 32 KB
SRAM ..... 2 KB
EEPROM ..... 1 KB
Raw Voltage Input 5-12V
Operating Voltage 5V
DC Current per I/O Pin ..... 40 mA
Maximum total current drawn from chip ..... 200 mA
Total Number of Digital I/O Pins 14
Number of PWM Capable Digital I/O Pins 6
Analog Input Pins 8

With a very small 18mm x 33mm footprint, the Pro Mini is an excellent choice for embedding in projects. Although it has less memory than the Mega, the 32k is usually sufficient memory for all but very large projects. It has 14 digital I/O pins, 6 of which are capable of PWM, and runs at the same 16MHz clock speed as the Mega.

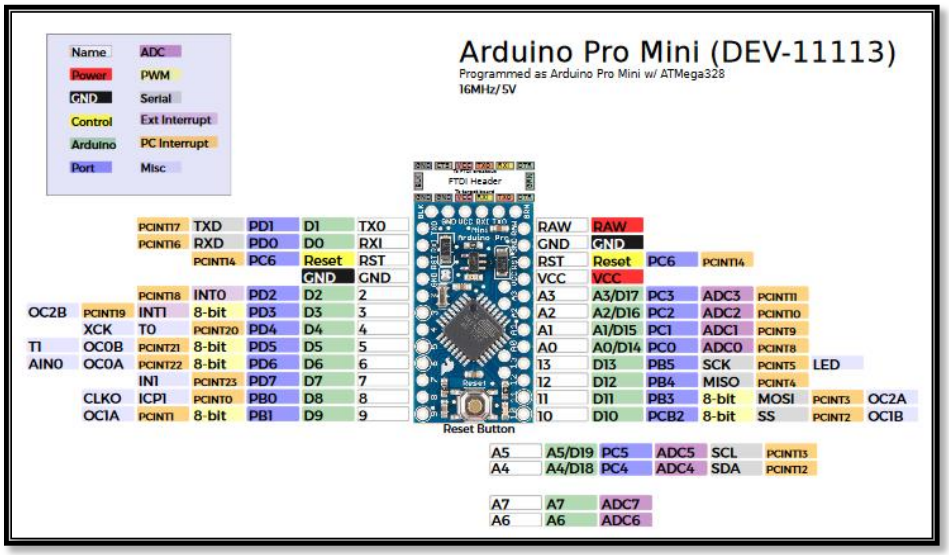

Figure 3 – Arduino Pro Mini board. *Source:* https://cdn.sparkfun.com/datasheets/Dev/Arduino/Boards/ProMini16MHzv1.pdf

Arduino IDE. Programming the microcontroller is one of the most important parts of creating any custom embedded system. Every microcontroller has a specific machine language in which all instructions for it must be written. Machine language is a sequence of 1s and 0s, and is usually arrived at by writing the instructions in a more human-friendly assembly language which is then converted to the required machine language.

Although assembly is more readable than machine language, it is still a very low level language and not at all intuitive to use. For this reason, most all microcontroller programs start out as high-level language such as C++, and are then compiled into assembly code. The assembly code is then converted into the machine language required by the specific microcontroller being programmed.

There are numerous ways to program microprocessors, but the Arduino IDE is the simplest and most straight-forward method for programming any of the Arduino style of boards. After selecting the appropriate board, the IDE will automatically compile code into the correct machine language and upload it onto the microprocessor. The Arduino IDE has a desktop version which can be installed directly onto the user's computer by following the detailed instruction guide provided at the download site.

The programming method recommended by Arduino.cc is the web interface Create Editor IDE. By using the web editor, users always have the most recent version of software, and the most complete list of available hardware and microprocessors. In addition, users can access and work on their projects from any computer able to log into the web site. New users especially can benefit from the web software because there is no complicated installation process involving downloading and installing separate compilers.

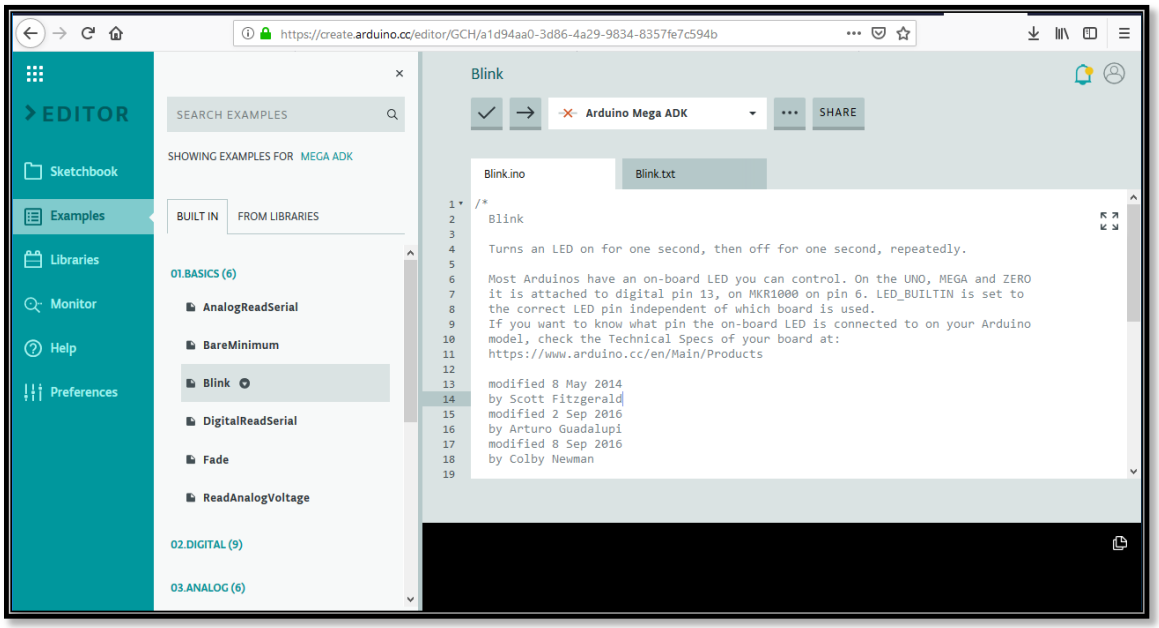

Figure 4 – Arduino IDE Web Interface. *Source:* https://create.arduino.cc/editor

Ultrasonic Sensors

Theory of Ultrasonic Ranging. Ultrasonic ranging is the technique of using ultra high frequency sound waves bounced off of an object to determine the distance of the object from the detector. An ultrasonic module sends out sound wave pulses from a transmitter, and then listens with a receiver for sound in the same frequency. If such a sound is detected, it means the sound pulses have bounced off of an object back towards the ultrasonic device.

The total elapsed time from when the pulses are transmitted until they are received back is then conveyed to the driving device. Other than very slight variations for temperature and humidity, the speed of sound in air is constant. Because the sound travels to an object, bounces off the object, and then returns to the ultrasonic module, multiplying the time elapsed by the speed of sound gives the roundtrip distance to the object and back. The roundtrip distance divided by two is the one way distance to the object ("Connecting an Ultrasonic Sensor").

This method of sending a signal and calculating a distance based on how long the signal takes to return is called time-of-flight (ToF) ranging.

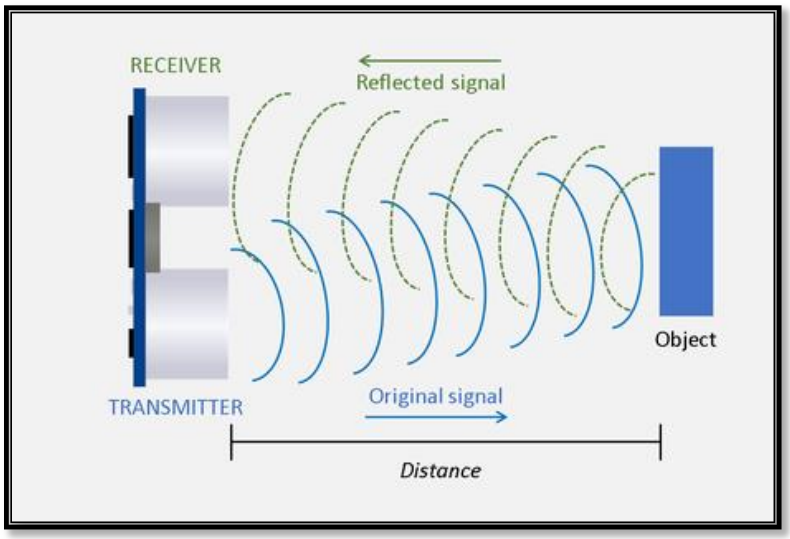

Figure 5 – Ultrasonic Ranging Technique. *Source:* http://www.teachwithict.com/hcsr045v.html

HC-SR04 Ultrasonic Module Specifications.

Power ..... 5VDC nom 4.5Vmin 5.5Vmax
Current ..... <20mA Operation
Current ..... <2mA Quiescent
Frequency 40KHz
Theoretical Range ..... 2cm to 4m (1in. to 13ft.)

The ultrasonic sound waves of an HC-SR04 travel outwards from the transmitter in an approximately 30 degree cone shape. The sound waves become more dispersed in all directions as they

travel away from the transmitter. Because of this, the farther away an object is from the sensor, the larger the object must be in order to return enough of the sound wave back to be detected by the receiver. Based on this information and preliminary tests, it was determined the best detection range of the sensor for this project was 45cm – 275cm.

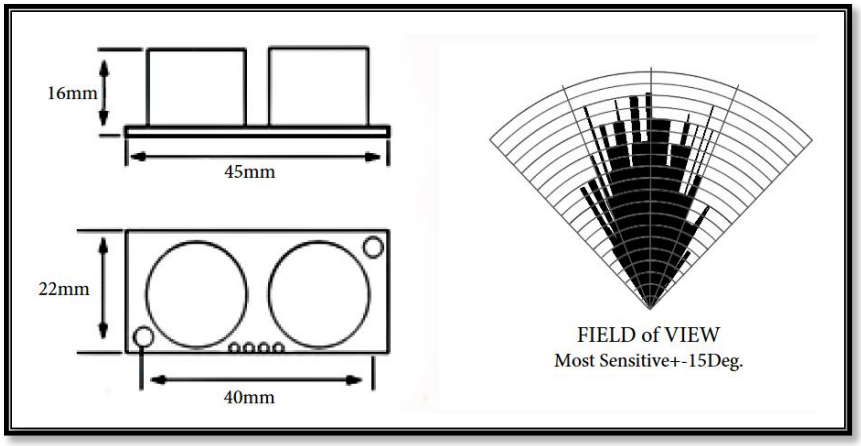

Figure 6 – HC-SR04 Dimensions and Detection Angle. *Source:* http://www.mpja.com/download/HC-SR04.pdf

Micro Vibration Motors

Micro Vibration Motor Specifications.

Diameter 10mm
Height 2.7mm
Voltage ..... 2V - 5V
5V Current Draw 100mA
4V Current Draw 80mA
3V Current Draw 60mA
Revolutions per Minute at 5V 11000
Weight ..... 0.9 gram

Micro Vibration Motor Operation. In order to provide haptic feedback in a very small space such as a cell phone, a tiny motor with off-center weight disk is used. These motors are completely self-contained and have no external moving parts, making them perfect for use in exposed locations. Because the eccentric mass counter weights are off-center, when the disk spins, it sets up a vibration ("Coin Vibration Motors"). The intensity and frequency of the vibrations can be controlled by adjusting the input voltage and using pulse width modulation of the input voltage. The figure below is an exploded diagram of the typical pancake style haptic motor showing the magnetic coils and weight disk.

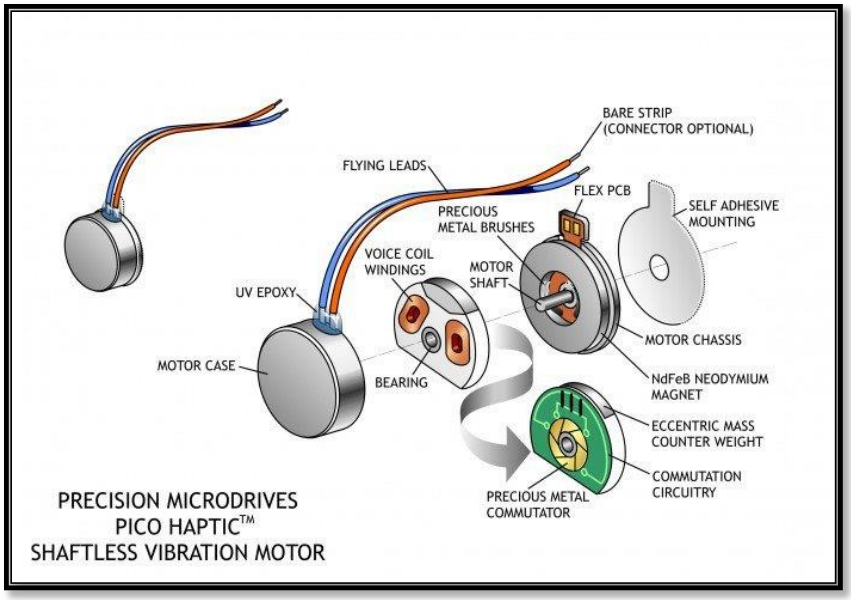

Figure 7 – Haptic Motor Diagram. *Source:* https://www.precisionmicrodrives.com/vibration-motors/coin-vibration-motors

Motor Current and Arduino Protection. Using motors in an electronics project requires special circuits for protection due to the amount of current motors draw as well as the amount of current they generate. Both of these issues are capable of destroying microcontrollers.

The signal pins of microcontrollers are not able to supply operating current to a device; they are only meant to indicate a logic level of High or Low. In an Arduino, the output pins will be damaged if currents over 40mA are drawn from any one pin, or a total of 200mA is drawn from any combination of pins (“Arduino Pin Current Limitations”). Because of this, it is not possible to power a motor of any kind with the signal pins of an Arduino.

Instead, the motor should be connected directly to an appropriate power supply, and an NPN transistor can be used as a switch to protect the microcontroller signal pin. The collector of the transistor is attached to the negative wire of the motor, and the emitter of the transistor is attached to ground. The Pulse Width Modulation signal pin of the Arduino is connected to the base pin of the transistor, and whenever logic high is sent, the transistor completes the circuit, allowing current to flow through the motor, through the transistor, and on to ground. A resistor is placed between the microcontroller signal pin and the transistor base pin in order to further protect the signal pin from accidental current overdraw (Monk).

The other main issue which must be dealt with when using motors is the amount of current they generate. In order to spin a motor, electricity is run through copper coils, creating a changing magnetic field which forces the motor to spin. The moment electricity to the motor stops, the still-spinning motor creates a changing magnetic field in the copper coils, which generates electricity. This newly created electricity must have somewhere to go, or there is a high risk of arcs which will damage electronics.

Because the motor speed of this project is being controlled by PWM which turns the power to the motor on and off at a rapid rate, it is critical to implement a circuit to allow the electricity generated by the spinning disk a path to be consumed. In order to do this, a diode is placed between the leads of the motor with the blocking side connected to the same lead as the input voltage. While the motor is being powered, the diode does not allow any additional connection. But as soon as the power to the motor stops and the motor becomes a generator, the diode allows the newly generated electricity a path to flow back into the motor and be quickly and safely consumed (Recktenwald).

One additional issue when using motors is the large amount of current they draw when first supplied power. In order to help smooth this additional current draw and also help absorb some of the electricity generated by the motor when electricity is stopped, a small ceramic capacitor can be added between the leads of the motor.

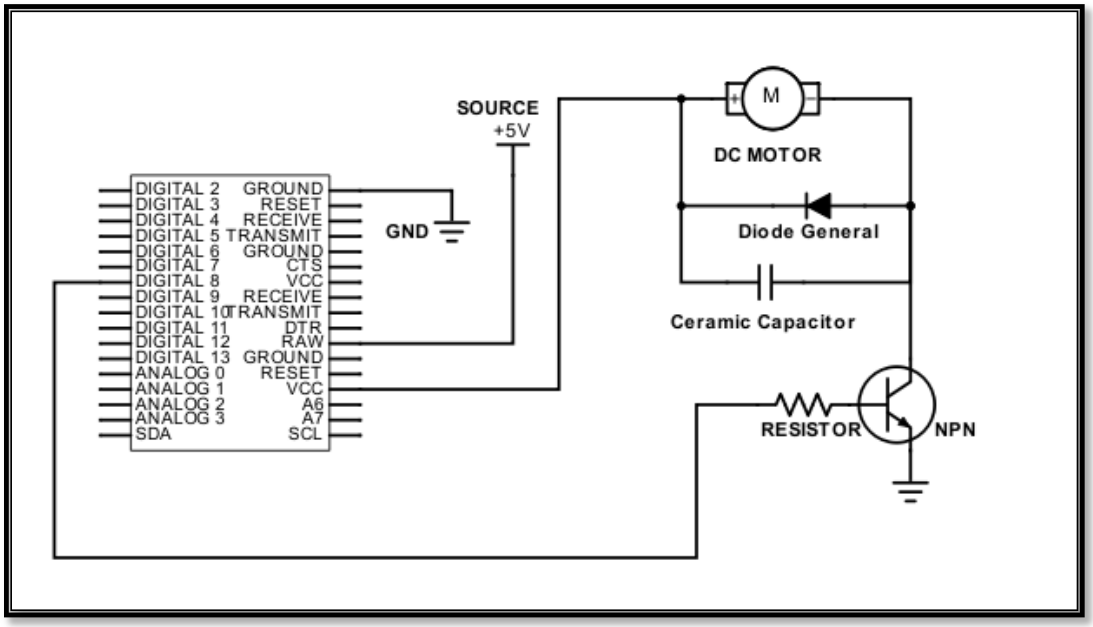

Figure 8 – Protection Circuit Schematic.

Software Development Section

User Experience

Acceptable Detection Rate. The HC-SR04 ultrasonic sensor was chosen for this project based on its extremely low cost and wide availability, which make it easy for anyone who wishes to build this project to acquire. It also has drivers written for it for use on Arduino boards. Unfortunately, two problems arose during development which made it impossible to use the pre-written drivers.

The NewPing library, which was written to allow an Arduino to drive an HC-SR04 module, did not work correctly in this system. It returned a very large number of zero distance readings for every one non-zero reading. Although the non-zero readings were accurate, the time between valid readings was too long due to all the invalid reading attempts. When the time and distance values were sent to the serial monitor and saved as a csv file, a scatter plot of the data showed a cyclical problem and an unacceptable successful reading rate.

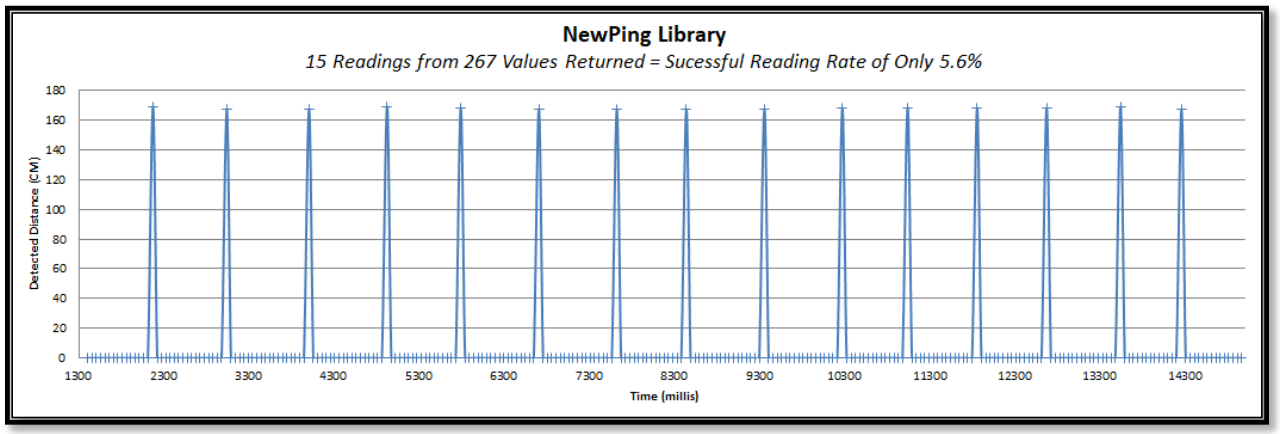

Figure 9 – Chart of Cyclical Problem Reading Distances with NewPing library.

By writing a custom program which utilized the Arduino PulseIn function to test the status of the echo pin on the ultrasonic module, the zero readings were eliminated and steady readings were achieved.

Intuitive Haptic Feedback. One of the main goals of this project is to provide intuitive information about the presence of obstacles and also indicate the distance to the obstacle. The only output of this system is the vibration of the micro motor, making it imperative to find a way to convey information in an intuitive manner needing no interpretation.

The first goal is to indicate the presence an object by powering the micro motor vibration when an object is detected. This alert also needs to convey information about the distance to the object. By varying the intensity to be approximately inverse to the distance, less sense of urgency should be given to farther away objects, and more urgency given to immediate obstacles.

Another important design consideration is the timeliness of the vibration feedback. Because the sensors are mounted on a cane which is being swept back and forth, it is critical the alert vibrations

happen at nearly the same instant the sweep passes the object. Any delay will cause the feedback to indicate the wrong location for the object.

Coding of Ultrasonic Sensors

Basics of communication. Communication with the HC-SR04 sensor is accomplished through the use of the trigger pin and the echo pin. In order to initiate ranging, a 10us (microsecond) pulse is sent to the trigger pin of the module. The module then emits eight 40KHz pulses of ultrasonic sound. The receiver is activated and the echo pin is set high. When the return pulses are detected, the echo pin goes back low, and the time taken to detect the return of the pulses is the time the echo pin was high.

By monitoring the timestamps of when the echo pin goes high and when it returns to low, the total time can be calculated by finding the difference. The time multiplied by the speed of sound gives the roundtrip distance, and when divided by two gives the one way distance.

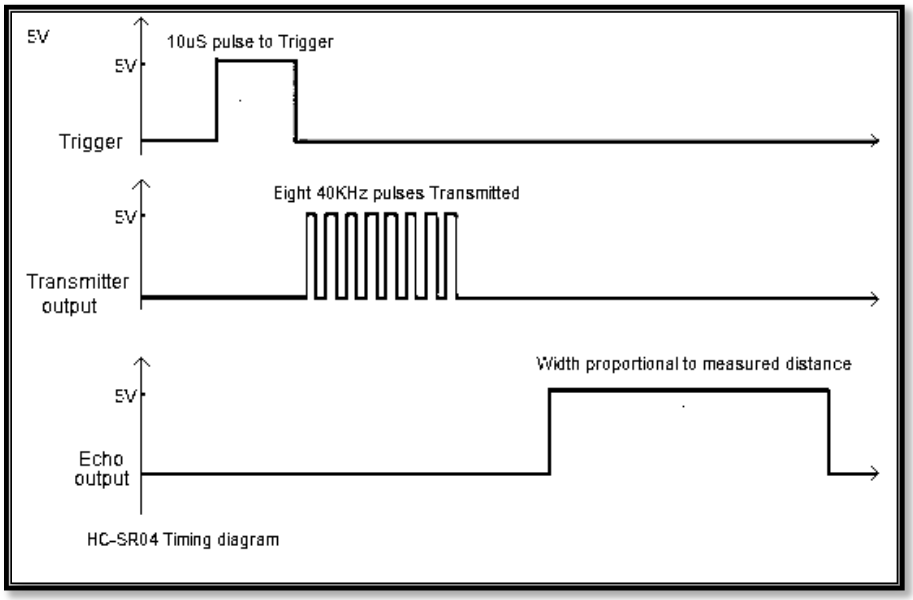

Figure 10 – HC-SR04 Communication Sequence. *Source:* https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

Software Improvement of Detection. The built in Arduino function pulseIn(pin, value, timeout) reads the given pin number for up to the timeout waiting for it to go from the opposite of the desired value, to the value, and back to the opposite of the value. A value of HIGH will return the duration of a LOW, HIGH, LOW pulse, and a value of LOW will return the duration of a HIGH, LOW, HIGH pulse. The duration returned is in microseconds (“Pulse In”).

By using the pulseIn() function to interface with the ultrasonic module instead of the NewPing library, the incident rate of erroneous zero values being returned was nearly eliminated.

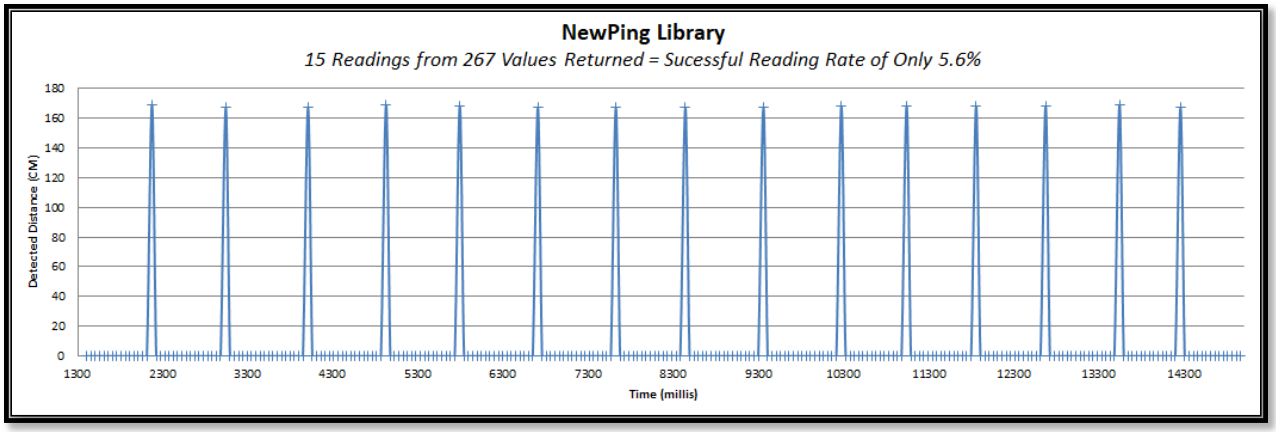

Figure 11 – Chart of Readings Using NewPing Library.

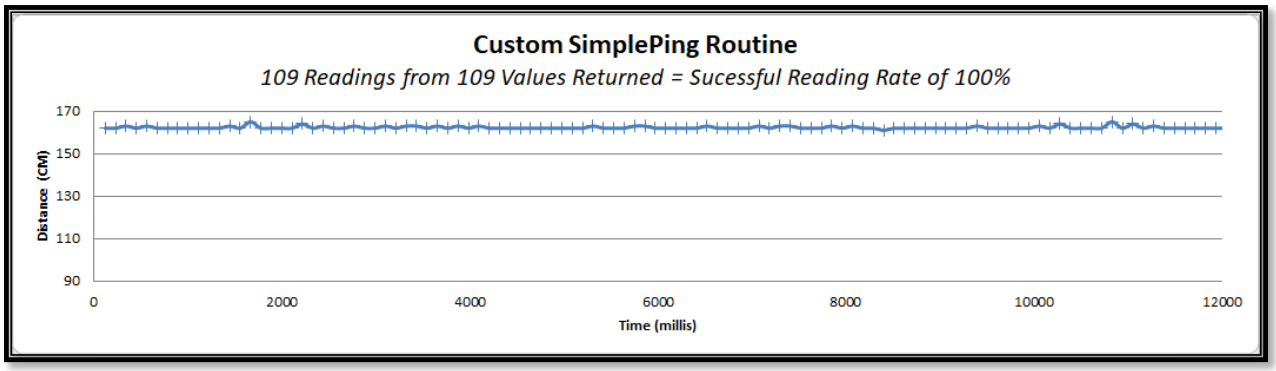

Figure 12 – Chart of Readings Using Custom Code.

Simultaneous Operation. The HC-SR04 module shows a high loss of accuracy in detecting objects approximately six to nine feet from the sensor. Although implementing a more expensive ultrasonic module might solve this problem, the more than ten-fold increase in cost needed to be avoided due to the accessibility goal of the project.

In order to improve the accuracy in this range of distances, it was decided to use two modules pointed in the same direction at the same time. Doing so doubles the ultrasonic energy output and also uses two receivers to improve the chances of detecting reflected pulses.

The pulseIn function can easily be used to operate a single HC-SR04 sensor, but only one call to pulseIn can be used at a time, so it is impossible to use the pulseIn() function to operate two sensors simultaneously. Developing a custom driver to operate two sensors simultaneously was a major component of this project.

Driving two sensors truly simultaneously requires choosing two output pins which are located in one port register to trigger the modules, and two input pins which are located within one port register to read the echo pins. Referring to the specifications of the architecture type being used can give the information of exactly which ports and bit locations particular pins are assigned to. For example, specifications for Arduinos using the Atmega169 show Arduino digital pin 8 is mapped to the first position of port B. Arduino digital pins 9, 10, and 11 are also assigned to port B in the second, third, and fourth positions respectively.

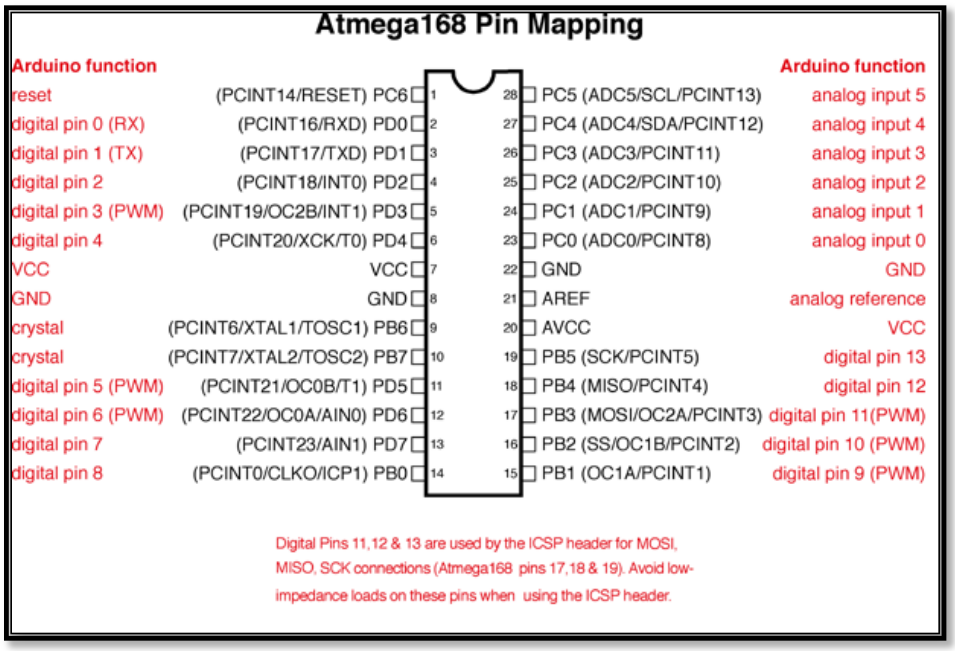

Figure 13 – Pin Mapping of Atmega168. *Source:* https://www.arduino.cc/en/Hacking/PinMapping168.

In this scenario, if Arduino digital pins 8 and 10 were used for triggering the ultrasonic modules, they would be assigned to the first and third bits of the B port. Assuming they had previously been assigned as output pins, setting these pins high would be accomplished by assigning PORTB the current contents of the PORTB register OR 00000101. The syntax is: PORTB = PORTB | B000000101. None of the other pin values are altered. Setting the first and third pins LOW without changing the state of any other pin requires assigning the PORTB register the current register contents AND 111111010. The syntax for this is: PORTB = PORTB & B11111010 (“Port Registers.”).

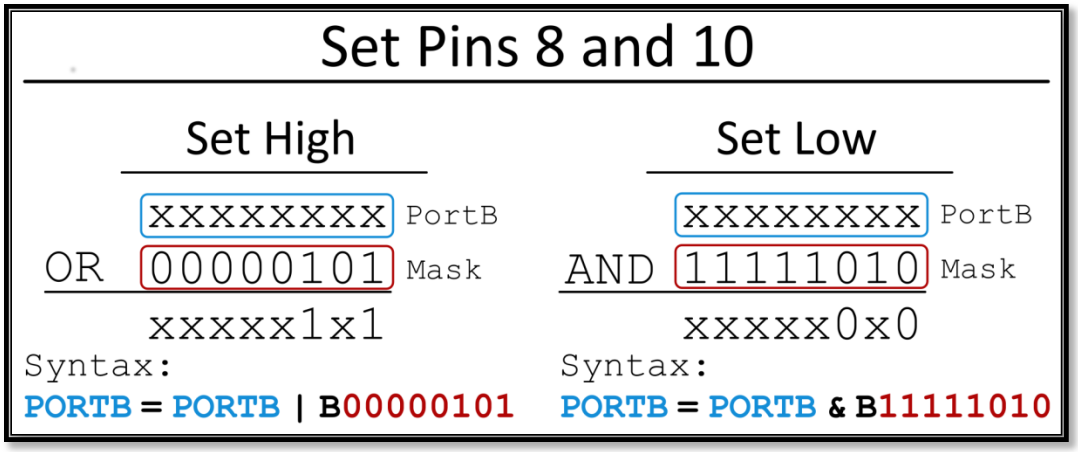

Figure 14 – Bitmasks and Logic for Simultaneous Setting of Trigger Pins.

Reading the echo pins simultaneously requires a similar process. If Arduino digital pins 9 and 11 were used for receiving the Echo signal from the ultrasonic modules, they would be assigned to the second and fourth bits of the B port. Assuming they had previously been assigned as input pins, checking the status of these pins would be accomplished by reading the contents of the PINB register and doing a bitwise AND with 00001010. The result would indicate one of four situations: no bits high – neither module sending an echo signal; second bit high – module 1 sending an echo signal; fourth bit high – module 2 sending an echo signal; second and fourth bit high – modules 1 and 2 both sending an echo signal.

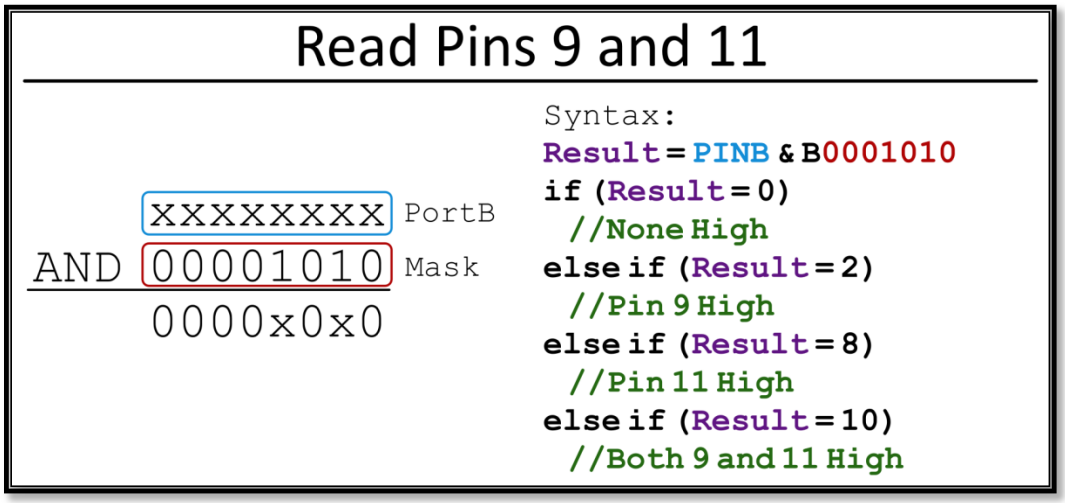

Figure 15 – Bitmasks and Logic for Simultaneous Reading of Echo Pins.

Portability. In order to trigger multiple pins simultaneously and read pin states simultaneously, the pins need to be mapped to the same register. Because of this, the selection of pins must be based on their register assignment, and then a bitmask must be hard-coded based on the location of the pins within the register.

For systems programmed onto a microcontroller directly and distributed as a complete system, control over pin assignment would be guaranteed and hard-coding bitmasks would not be an issue. This project, however, is designed for reproducibility by hobbyists, who may be working with a different Arduino architecture, or may need to assign different digital pins to be used with the ultrasonic modules. The simultaneous use of the ultrasonic sensors to boost range and accuracy is mutually exclusive to the portability needed to allow hobbyists to recreate the system. Unfortunately, in this project portability is a higher priority than precision of distance, meaning the simultaneous writing or reading of registers is impossible.

The compromise to this situation was found by using sequential-parallel operation of the sensors which introduces slight errors in elapsed time values, but not enough to have a detrimental effect on the output. The main Arduino.h library includes functions which can be passed a pin number, returning the port, bitmask, or register based upon the exact Arduino architecture being compiled for.

These functions allow the portability to be maintained. Then, separating the writes or reads and applying them to one pin at a time only introduces an additional four-millionths of a second error in timing for the additional instruction processing. Because the errors in time are so small they result in only one or two centimeter errors in distance. Detecting objects does not require precision distance values, so the difference between true simultaneous ultrasonic sensor operation and this sequential-parallel operation is imperceptible in the system, but allows for full portability between Android platforms.

Coding of Micro Vibration Motor

Software Control of Motor. Once the necessary electronic components are in place to protect the electronics of the Arduino from over-draw of current, control of the micro vibration motor is accomplished by using an Arduino pin capable of PWM and using digitalWrite() to set the intensity of the motor. The smaller the value written, the smaller the duty cycle of the pulse sent to the motor, causing the motor to be supplied with less power and slowing the rate the motor spins. Every time a reliable distance value is calculated, the power level of the motor should be written in order for changes in distance to be detected in real-time.

Non-Linear Intensity. The farther away an object is detected by this system, the less imperative it is for the user to navigate away from it. As closer objects are detected, though, the feedback needs to alert the user to the increased need of avoidance. Devising the means of translating the calculated distance into intuitive haptic feedback required a considerable amount of trial and error.

Originally, an exponential decay equation was used to deal with the issue of the distance being inversely proportional to the urgency needing to be conveyed as well as the increased importance of

navigating away from closer objects. Unfortunately, even though the values were correct and operated the intensity of the motor as expected, the haptic feedback was not intuitive in helping locate an unobstructed path.

The technique which provided the most intuitive feedback was a simple system of sectioning the distances into zones and applying an intensity level based on the zone detected. The intensity levels used change more between zones as objects get closer. The abrupt changes in intensity level provide a more intuitive feedback of the location and distance of obstacles and help users select a path which is clear of obstacles.

Code Walk-Through Section

Overall Logic. The coding logic for the system relies on timestamps and flags. This walk-through explains the overall logic, but does not include the register checking and bitmasking which was involved. Use of the bit level logic is described in the comments within the actual code shown in Appendix A.

While the unit is powered on, the program continuously loops through the following:

- Set all variables to 0 and flags to false
- If either Echo input pin reads high (system is not ready)
    - Wait ¼ millisecond and check again.

Signal both modules to initiate ranging sequence:

    - Set Trigger 1 low, set Trigger 2 low, wait 4 microseconds
    - Set Trigger 1 high, set Trigger 2 high, wait 16 microseconds
    - Set Trigger 1 low, set Trigger 2 low
- Save the current time in TimePulseSent
- Begin loop which continues while both modules have not had their Echo HighSet flag set to true and elapsed time has not passed the maximum wait
    - If Echo pin 1 reads high and HighSet1 flag is still false
        - Set StartTime1 to current clock microseconds
        - Set HighSet1 to true (module 1 started to listen for reflected sound)
    - If Echo pin 2 reads high and HighSet2 flag is still false
        - Set StartTime1 to current clock microseconds
        - Set HighSet1 to true (module 2 started to listen for reflected sound)
- If either Echo HighSet flag is true (at least one module began listening)
    - Begin loop which continues as long as both modules have not had their Echo LowSet flag set to true and elapsed time has not passed the maximum wait
        - If Echo pin 1 reads low and LowSet1 flag is still false
            - Set EndTime1 to current clock microseconds
            - Set LowSet1 to true
        - If Echo pin 2 reads low and LowSet2 flag is still false
            - Set StartTime1 to current clock microseconds
            - Set LowSet1 to true

Requests have been sent to both modules to initiate a ranging sequence. Both, one, or neither may have been successful. Setting elapsed time depends on the state of the HighSet and LowSet flags for each module.

- If Echo HighSet1 flag is true (module began listening) and Echo LowSet1 is true (module heard reflected sound)
    - Set RoundtripTime1 to EndTime1 – StartTime1
    - Set Distance1 to RoundtripTime1 / 58 (Calculation for centimeters)

- If Echo HighSet flag is true (module began listening) and Echo LowSet is FALSE (module never heard any reflected sound)
    - Set Distance1 to MaximumDistance

Repeat for module two:

- If Echo HighSet2 flag is true (module began listening) and Echo LowSet2 is true (module heard reflected sound)
    - Set RoundtripTime2 to EndTime2 – StartTime2
    - Set Distance2 to RoundtripTime2 / 58 (Calculation for centimeters)
- If Echo HighSet flag is true (module began listening) and Echo LowSet is FALSE (module never heard any reflected sound)
    - Set Distance2 to MaximumDistance

How to use distances depends on whether or not both modules were successful and whether or not the distances detected agree.

- If Distance1 and Distance 2 are not 0 (Distance calculations were set for both modules)
    - If the difference between the two distances is less than 50cm (Modules agree)
        - Use both distances in average
    - If the difference between the two distances is more than 50cm (Modules agree)
        - Use the farthest distance in average (look for clear path)

Motor intensity is set based on the average distance.

- If Average is less than 1 or greater than 273 (Module anomaly or clear path)
    - Set Intensity to 50 of 255
- Else, if Average is less than 80 (Extremely close obstacle)
    - Set Intensity to 255 of 255
- Else, if Average is less than 120 (Very close obstacle)
    - Set Intensity to 200 of 255
- Else, if Average is less than 160 (Somewhat close obstacle)
    - Set Intensity to 150 of 255
- Else, if Average is less than 200 (Somewhat far obstacle)
    - Set Intensity to 110 of 255
- Else, if Average is less than 240 (Somewhat farther obstacle)
    - Set Intensity to 75 of 255
- Else, if Average is less than 80 (Very far obstacle)
    - Set Intensity to 65 of 255

Begin the cycle again.

Conclusion

Contributions. Overall, the components of this system may not be novel, but currently there does not seem to be an Arduino driver for ultrasonic modules available which allows for enhancing range and responsiveness by using two modules nearly simultaneously. This project's creation and free distribution of such software could be a significant contribution to the Arduino community.

In addition, detailing the components and steps to create the system and freely distributing the information may empower novice programmers and even non-programmers to build this system to help a friend, loved-one, or community member who has impaired vision. Improving another person's quality of life by enhancing their ability to navigate unfamiliar areas would certainly be a contribution, as would sparking creativity in someone who is inspired to make this project better.

Limitations. This system is not intended to replace a tapping cane for visually impaired persons. Tapping canes give a wide variety of critical information about terrain which cannot be easily replaced. This system can provide auxiliary information about objects which are beyond the range of the tapping cane, though, in order to help users select a clear path before running into obstacles.

Because this project is a prototype, none of the components have been made waterproof, which would be necessary for use outdoors if there is any possibility of rain. Ultrasonic technology inherently has difficulties detecting sound-absorptive materials including items which are furry, generally fibrous, or fabric. Items which have small surface areas such as chair legs are also difficult for ultrasonic sensors to pick up.

Possible Enhancements

- **Mute button**. In very busy or crowded surroundings, the constant haptic feedback may become distracting to a user. Placing a button in the system which would temporarily stop any detection and motor feedback for a pre-determined amount of time would be advisable.

- **Haptic motor module**. Haptic vibration motor modules are available with built-in protection circuitry making the additional soldering of individual electronic components unnecessary. Some also have libraries which trigger the motor to create distinctly different vibration patterns, which could be used to provide additional information.

- **Triple axis magnetometer module integration**. This would give important compass heading information to the user even if it were only used to allow the user to determine the direction of North. Having access to this information could help a person orient themselves and navigate both indoors and out.

- **LIDAR module integration**. Garmin makes a LIDAR module which can be operated by an Arduino for extremely accurate indoor obstacle detection. Harsh lighting conditions, such as outdoors on a sunny day, can diminish the range these modules can detect, though, which might make an ultrasonic sensor a better choice for those conditions. Implementing a photocell could automatically switch priority detection between LIDAR and ultrasonic based on the intensity of surrounding light.

Biography

G.C. Holden is an undergraduate student at Northeastern State University in Broken Arrow, Oklahoma. She is currently working on three degrees: B.S. Computer Science with Mathematics minor, B.B.A. Business Administration: Business Analytics, and M.B.A. Business Administration: Business Analytics. At the age of seven, she fell in love with computers and started teaching herself to program. Now she is interested in applying technology to create accessible systems which can improve people's way of life. By freely sharing knowledge and ideas, she hopes to spark new ideas and creativity in others.

References

"Arduino Pin Current Limitations." *Arduino.cc*, playground.arduino.cc/Main/ArduinoPinCurrentLimitations. Accessed 14 Oct. 2018.

"Coin Vibration Motors." *Precision Microdrives*, www.precisionmicrodrives.com/vibration-motors/coin-vibration-motors. Accessed 2 Oct. 2018.

"Connecting an Ultrasonic Sensor." *Teach with ICT*, www.teachwithict.com/hcsr045v.html. Accessed 2 Oct. 2018.

Monk, Simon. "Arduino Lesson 13." *Adafruit*, learn.adafruit.com/adafruit-arduino-lesson-13-dc-motors/transistors. Accessed 14 Oct. 2018.

"Port Registers." *Arduino.cc*, www.arduino.cc/en/Reference/PortManipulation. Accessed 1 Oct. 2018.

"Pulse In." *Arduino.cc*, www.arduino.cc/en/Reference.PulseIn. Accessed 26 Sept. 2018.

Recktenwald, Gerald. "Basic DC Motor Circuits." *Sparkfun*, cdn.sparkfun.com/assets/resources/4/4/DC_motor_circuits_slides.pdf, pp. 4-8. Accessed 14 Oct. 2018.

Appendix A

Code of
Ultrasonic Embedded System

```
// Simultaneous Ultrasonic Readings
// by G. C. Holden
// 11/22/2018
// Allows for improved consistency in ultrasonic reading by utilizing
// two inexpensive HC-SR04 modules pointed in the same direction and
// manually pulsing them nearly simultaneously, using timestamps (no interrupt timers)
// to calculate distances.


#include <Arduino.h>    // Must include the Arduino Library
//#include <LiquidCrystal.h>    // Uncomment if using 16 x 2 LCD display for testing

#define TRIG_PIN1        8    // Pin # connected to Ultrasonic Module 1 Trigger
#define ECHO_PIN1        9    // Pin # connected to Ultrasonic Module 1  Echo

#define TRIG_PIN2        10   // Pin # connected to Ultrasonic Module 2 Trigger
#define ECHO_PIN2        11   // Pin # connected to Ultrasonic Module 2 Echo

#define MOTOR_PWM_PIN    6    // PWM Pin # connected to Motor control

#define ELEMENTS         4    // Number of elements to average together for distance reading

#define MAX_ECHO_WAIT    15892   // Maximum time to wait for echo pins to return low (distance of 274 cm * 58 microseconds seconds per cm)
#define MAX_READY_WAIT   550     // Maximum number of microseconds to wait for echo pins to be high (ready)

// LiquidCrystal lcd(8, 9, 4, 5, 6, 7); // Uncomment to Setup LCD display object

uint8_t port1;      // Register corresponding to echo pin of ultrasonic module 1
uint8_t bitMask1;   // Bitmask corresponding to echo pin of ultrasonic module 1
uint8_t port2;      // Register corresponding to echo pin of ultrasonic module 2
uint8_t bitMask2;   // Bitmask corresponding to echo pin of ultrasonic module 2

bool echoHigh_1;    // Flag indicating that Module 1 Echo pin goes high and timestamp set
bool echoLow_1;     // Flag indicating that Module 1 Echo pin goes back low and timestamp set
bool echoHigh_2;    // Flag indicating that Module 2 Echo pin goes high and timestamp set
bool echoLow_2;     // Flag indicating that Module 2 Echo pin goes back low and timestamp set

uint32_t Tm_outOfSetLoop; // Timestamp of exiting the loop to wait for echo pins to be ready
uint32_t Tm_outOfRtnLoop; // Timestamp of exiting the loop to wait for return of ping
uint32_t Tm_pulseSent;    // Timestamp of pulse initialization routine beginning

uint32_t Tm_start1;  // Timestamp of Module 1 Echo pin going high
uint32_t Tm_end1;    // Timestamp of Module 1 Echo pin going low

uint32_t Tm_start2; // Timestamp of Module 2 Echo pin going high
uint32_t Tm_end2;   // Timestamp of Module 2 Echo pin going low

uint32_t Tm_1 = millis();      // Timestamp of beginning of one averaging cycle
uint32_t Tm_2 = millis();      // Timestamp of end of one averaging cycle

uint32_t Tm_roundtrip1;    // Microseconds until ping returns
uint32_t Tm_roundtrip2;    // Microseconds until ping returns

volatile uint8_t* addrInputRegPort1;  // Address of the register holding the input state for port 1
volatile uint8_t* addrInputRegPort2;  // Address of the register holding the input state for port 2
// Declared volatile in case ultrasonic module changes the state while temporarily stored

int dist1 = 0;                 // Temporary storage of calculated distance 1
int dist2 = 0;                 // Temporary storage of calculated distance 2
int distances[ELEMENTS + 2] = {0};// Array holding validated distance calculations
int index = 0;                 // Array indexing
int sum = 0;                   // Accumulator for averaging distances
int average = 0;               // Average of validated distances
int intensity = 0;             // Motor intensity


void setup() {         // Initialization routine only runs once
  // Define pin modes
  pinMode(TRIG_PIN1, OUTPUT);       // Pin used to trigger module 1
  pinMode(ECHO_PIN1, INPUT);        // Pin used to read time of flight for module 1
```

```
  pinMode(TRIG_PIN2, OUTPUT);       // Pin used to trigger module 2
  pinMode(ECHO_PIN2, INPUT);        // Pin used to read time of flight for module 2

  pinMode(MOTOR_PWM_PIN, OUTPUT);   // PWM pin used to control intensity of haptic motor

  // Store the register corresponding to Echo pin 1
  port1 = digitalPinToPort(ECHO_PIN1);
  // Store value of the current numbered pins register bitmask in pinToBitMask
  bitMask1 = digitalPinToBitMask(ECHO_PIN1);
  // Store the address of the input register of port 1
  addrInputRegPort1 = portInputRegister(port1);

  // Store the register corresponding to Echo pin 2
  port2 = digitalPinToPort(ECHO_PIN2);
  // Store value of the current numbered pins register bitmask in pinToBitMask
  bitMask2 = digitalPinToBitMask(ECHO_PIN2);
  // Store the address of the input register of port 2
  addrInputRegPort2 = portInputRegister(port2);

  //   // Uncomment block to ready LCD display
  //   // USE LCD DISPLAY FOR TESTING OF VALUES
  //   // Initialize LCD display
  //   lcd.begin(16, 2);              // Define 16 character x 2 line
  //   lcd.clear();
  //   lcd.setCursor(0,0);
  //   lcd.print("Booting up...."); // print a simple message
  //   delay(1000);
  //   lcd.clear();
  //   lcd.setCursor(0,0);
  //   lcd.print("Dist 1:     cm");
  //   lcd.setCursor(0,1);
  //   lcd.print("Elapse:     millis");

  //   //**** Uncomment block in order to output CSV Header to serial monitor ****
  //   // Start Serial communication
  //   Serial.begin(115200);
  //   // Wait 1 second to enable activation of Serial Monitor
  //   delay(1000);
  //   Serial.println( "Intensity,   port1 , bitMask1,  port2,  bitMask2, *addrInputRegPort1, *addrInputRegPort2 , Tm_outOfSetLoop,  Tm_outOfRtnLoop,
Tm_pulseSent, echoHigh_1,  echoLow_1, Tm_start1, Tm_end1, echoHigh_2,  echoLow_2,  Tm_start2, Tm_end2, Tm_roundtrip1, Tm_roundtrip2, Dist1,
Dist2");

}    // END of initialization


// Main loop runs continuously after setup
void loop() {
  // Small delay to allow ultrasonic waves to clear
  delay(20);

  // Clear values for this ping cycle
  Tm_start1 = 0;
  Tm_start2 = 0;
  Tm_end1 = 0;
  Tm_end2 = 0;
  Tm_roundtrip1 = 0;
  Tm_roundtrip2 = 0;
  dist1 = 0;
  dist2 = 0;
  echoHigh_1 = false;
  echoHigh_2 = false;
  echoLow_1 = false;
  echoLow_2 = false;

  // If either Echo input pin is still high (true/anything but 0), wait
  while ( ((bitMask1 & *addrInputRegPort1)) || ((bitMask2 & *addrInputRegPort2))) {
    delayMicroseconds(250);   // Wait 1/4000 second before checking again
  }

  // Basic sequence to signal both HC-SR04s to Transmit
```

```
digitalWrite(TRIG_PIN1, LOW);   // Set trigger pin of both modules LOW
digitalWrite(TRIG_PIN2, LOW);
delayMicroseconds(4);          // Hold for 4 millionths of a second
digitalWrite(TRIG_PIN1, HIGH);  // Set trigger pin of both modules HIGH
digitalWrite(TRIG_PIN2, HIGH);
delayMicroseconds(16);         // Hold for 16 millionths of a second
digitalWrite(TRIG_PIN1, LOW);   // Set trigger pin of both modules LOW
digitalWrite(TRIG_PIN2, LOW);

// Timestamp of when the pulse was requested - for timeout calculation
Tm_pulseSent = micros();

// Transition-to-high timestamp setting loop
// Keep checking the echo pins while they are not both high and time is within reason
while ( !(echoHigh_1 && echoHigh_2) && ( (micros() - Tm_pulseSent) < ( MAX_READY_WAIT ) )) {

   // Check if input register port BITWISE-ANDed with bitmask
   // is true (Echo pin 1 has gone high) AND the echo pin 1 high flag is not yet set
   if ( (!echoHigh_1) && (bitMask1 & *addrInputRegPort1)) {
      // Set module 1's beginning timestamp and set the started flag to true
      Tm_start1 = micros();
      echoHigh_1 = true;
   }
   // Check if (input register port BITWISE-ANDed with bitmask)
   // is true (Echo pin 2 has gone high) AND the echo pin 2 high flag is not yet set
   if ( (!echoHigh_2) && (bitMask2 & *addrInputRegPort2)) {
      // Set module 2's beginning timestamp and set the started flag to true
      Tm_start2 = micros();
      echoHigh_2 = true;
   }
}   // End of Transition-to-high timestamp loop

// Timestamp when the Echo going high loop was exited
Tm_outOfSetLoop = micros();

// Transition-to-low timestamp setting loop
// If either of the Echo pins went high - meaning they sucessfully started listening
if ( echoHigh_1 || echoHigh_2) {

   // Keep checking the echo pins while they are not both low and time is within reason
   while ( !(echoLow_1 && echoLow_2) && ((micros() - Tm_pulseSent) < ( MAX_ECHO_WAIT ) )) {

      // If the Echo pin 1 sucessfully went high AND
      // input register port BITWISE-ANDed with bitmask is false (pin has gone low)
      // AND the echo pin 1 low flag is not yet set
      if ( ( (echoHigh_1) && (!echoLow_1) && !(bitMask1 & *addrInputRegPort1)) ) {
         // Set the first module ending timestamp and set the ended flag to true
         Tm_end1 = micros();
         echoLow_1 = true;
      }

      // If the Echo pin 2 sucessfully went high AND
      // Input register port BITWISE-ANDed with bitmask is false (pin has gone low)
      // AND the echo pin 2 low flag is not yet set
      if ( (echoHigh_2) && (!echoLow_2) && !(bitMask2 & *addrInputRegPort2) ) {
         // Set the second module ending timestamp and set the ended flag to true
         Tm_end2 = micros();
         echoLow_2 = true;
      }

   }   // End of Transition-to-low timestamp loop

   // If module 1 started listening
   if (echoHigh_1) {
      // If module 1 heard an echo, calculate the roundtrip time
      if (echoLow_1) {
         Tm_roundtrip1 = Tm_end1 - Tm_start1;
      }
         // If module 1 did not hear an echo, set roundtrip time to maximum
      else {
```

```
            Tm_roundtrip1 = MAX_ECHO_WAIT;
        }
        // One way distance is approx equal to roundtrip time / 58
        dist1 = Tm_roundtrip1 / 58;
    }

    // Same sequence for module 2
    if (echoHigh_2) {
        if (echoLow_2) {
            Tm_roundtrip2 = Tm_end2 - Tm_start2;
            dist2 = Tm_roundtrip2 / 58;
        } else {
            Tm_roundtrip2 = MAX_ECHO_WAIT;
        }
        dist2 = Tm_roundtrip2 / 58;
    }
}

// Timestamp when the Echo going low loop was exited - for debugging
Tm_outOfRtnLoop = micros();


// WORK WITH DISTANCE VALUES TO CLEAN UP BAD DATA AND DISCREPANCIES
// If there are distance values for both of the modules
if (dist1 && dist2) {
    // If the two distances agree (are within 50cm of each other)
    if ( abs(dist1 - dist2) < 50 ) {
        // Add the distances to the distances array and increment index
        distances[index] = dist1;
        index++;
        distances[index] = dist2;
        index++;
    } // If there are two values, but they differ more than 50cm
    else {
        // Place the larger value into the distance array and increment the counter
        if (dist1 >= dist2) {
            distances[index] = dist1;
        }
        else {
            distances[index] = dist2;
        }
        index++;
    }
} // If two values were not set then place either one or none into the array
else {
    if (dist1) {
        distances[index] = dist1;
        index++;
    }
    else if (dist2) {
        distances[index] = dist2;
        index++;
    }
}

// If there are at least the required number of values in the array, average them
if (index >= ELEMENTS) {
    for (int i = 0; i < ELEMENTS; i++) {
        sum += distances[i];
    }
    // Calculate average and then reset variables
    average = sum / ELEMENTS;
    index = 0;
    sum = 0;
    Tm_2 = millis();


    // If the distance indicates the path is clear of obstacles, set motor intensity low
    if ((average <= 0) || average >= 274) {
        intensity = 50;
```

```
    }
    else if (average < 80) {
      intensity = 255;
    }
    else if (average < 120) {
      intensity = 200;
    }
    else if (average < 160) {
      intensity = 150;
    }
    else if (average < 200) {
      intensity = 110;
    }
    else if (average < 240) {
      intensity = 75;
    }
    else if (average < 274) {
      intensity = 65;
    }

    analogWrite(MOTOR_PWM_PIN, intensity);

    //      // Output distance in CM to LCD for testing purposes
    //      lcd.setCursor(8,0);
    //      lcd.print("   ");
    //      lcd.setCursor(8,0);
    //      lcd.print(average);
    //
    //      lcd.setCursor(8,1);
    //      lcd.print("    ");
    //      lcd.setCursor(8,1);
    //      lcd.print(Tm_2-Tm_1);

    Tm_1 = Tm_2;    // Move end of this averaging cycle to beginning of next
  }

  //   //**** Uncomment block in order to output CSV data to serial ****
  //
  //   Serial.print( intensity);    Serial.print(" ,   ");
  //   Serial.print( port1);        Serial.print(" , ");
  //   Serial.print( bitMask1);     Serial.print(" , ");
  //
  //   Serial.print( port2);        Serial.print(" , ");
  //   Serial.print( bitMask2);     Serial.print(" , ");
  //
  //   Serial.print(* addrInputRegPort1);     Serial.print(" , ");
  //   Serial.print(* addrInputRegPort2);     Serial.print(" , ");
  //   Serial.print( Tm_outOfSetLoop);        Serial.print(" , ");
  //   Serial.print( Tm_outOfRtnLoop);        Serial.print(" , ");
  //   Serial.print( Tm_pulseSent);           Serial.print(" , ");
  //
  //   Serial.print(echoHigh_1);      Serial.print(" , ");
  //   Serial.print(echoLow_1);       Serial.print(" , ");
  //   Serial.print( Tm_start1);      Serial.print(" , ");
  //   Serial.print( Tm_end1);        Serial.print(" , ");
  //   Serial.print(echoHigh_2);      Serial.print(" , ");
  //   Serial.print(echoLow_2);       Serial.print(" , ");
  //   Serial.print( Tm_start2);      Serial.print(" , ");
  //   Serial.print( Tm_end2);        Serial.print(" , ");
  //
  //   Serial.print( Tm_roundtrip1);  Serial.print(" , ");
  //   Serial.print( Tm_roundtrip2);  Serial.print(" , ");
  //
  //   Serial.print( dist1);        Serial.print(" , ");
  //   Serial.print( dist2);        Serial.println("");


} // END of main loop
```