EE 478 – Final Project

Ultrasonic Wayfinder

Alex Chiem, Vien Ly, Michael Molina

June 10, 2014

# Contents

Contents 2

Abstract 4

Introduction 4

Requirement Specification 4

System Description 4

Specification of External Environment 5

System Input and Output Specification 5

System Inputs 5

System Outputs 5

User Interface 5

Use Case Diagram 6

Textual Description 6

System Functional Specification 7

Operating Specifications 7

Reliability and Safety Specification 7

Design Specification 8

System Description 8

Specification of External Environment 8

System Input and Output Specification 8

System Inputs 8

System Outputs 9

User Interface 9

System Functional Specification 10

Operating Specifications 12

Reliability and Safety Specification 13

Design Procedure 13

System Description 13

Software Implementation 14

Head-mounted PIC18F25K22 Microcontroller 14

Setup 14

Distance Measurement 15

Output 16

Communication 17

Hardware Implementation 18

Test Plan 19

Test Specification 20

Test Cases 20

Results 21

Head Module/ Hand-mounted Module 21

Error Analysis 30

Analysis of Why Project May Not Have Worked 30

Summary 30

Conclusion 31

ABET 31

Appendix 32

Head Module 34

Setup.h 34

ISR.h 34

Wayfinder.h 35

ISR.c 35

Setup.c 44

Wayfinder.c 57

Hand Module 70

Setup.h 70

ISR.h 70

Setup.c 71

ISR.c 77

Hand\_mounted\_unit.c 80

# Abstract

The ultrasonic pathfinding prototype is a system designed and implemented to provide sight- impaired individuals with the ability to recognize their immediate surrounding using ultrasonic echolocation capabilities of two different transducer pairs. The first transducer pair is located on top of a hat, which the user can wear. This module continuously calculates the distance between the user and the immediate obstacles that might lie ahead, then relay this information to the user by vibrating a motor at an intensity proportional to the distance. When encountered an obstacle blocking the path of the user, the system scans the left and the right areas of the user then suggests the direction to take. Another transducer module is mounted onto the hand of the user. Using haptic feedback in the form of another vibration motor, this module gives the user the ability to recognize how far away an obstacle is by pointing the transducer in its direction. The system also supports a remote monitor system. The monitor provides basic knowledge about the user's activities as well as the ability to assist the user when a problem is encountered.

The main driver of the system includes two PIC18F25k22 MCUs. The software driver of the system is written in C and implemented in the MPLAB X embedded environment. Debugging is done using the PICkit 3 in-circuit debugger and an Agilent 4000 X-Series Logic Analyzer. During implementation, the system is fully tested to ensure that the individual modules and communication channels work separately. Then by careful integration of each module starting from the head-mounted module, the system is successfully tested and installed.

This documentation provides thorough description of the system, design process, testing procedures, error analysis, and the final result of the implementation.

# Introduction

The ultrasonic pathfinder system is a system that will give haptic feedback based on the distance of objects in front, left and right of the user. This system was designed to assist individuals with vision impaired disabilities. The system includes ultrasonic sensors that captures the distance of objects by transmitting a signal in a certain direction and receiving it back. The duration it takes for this to happen is used to calculate the distance of the object. If the object is far away, the vibration motor will vibrate with lower intensity. If the object is closer, the vibration motor will vibrate with higher intensity. This system also includes a hand mounted module where the user can scan freely in a direction they like and also receive haptic feedback. The system is implemented by using the PIC18F25K22 microcontrollers. The head and hand modules communicate through I2C communication.

# Requirement Specification

## System Description

The design specification and requirements are describes as an implementation of an ultrasonic-range-finding device that utilizes haptic feedback to alert its user about obstacles in the user’s path. The device will consists of at least two ultrasonic transmitter/receiver sensors that can emit short pulses of ultrasonic frequency sound. The receiver picks up the echoed pulses. The time delay between the emission of the initial pule and when the pulse is received gives the system the necessary data to determine how far away the obstacle is from the sensor, which can be described to the user by vibrating a small motor at the intensity proportional to the distance. This gives the user a sense of spatial recognition in almost all kind of environment. The sensor will be on a servo motor so it can scan for objects in front, left and right of the user and control different motor’s intensity. The motor will only turn the sensor when an object is too close in front of the user. Also a hand-mounted module will be implemented so the user can directly point at an area they want to scan. The head module will also send data wirelessly to a remote monitoring system for assistance when the user is in trouble. The project is purposed as a proof-of-concept for a product that can be used by sight-impaired individuals to detect obstructions.

## Specification of External Environment

The system is designed to operate in any dry environment. But its main purpose is to help the user navigate spaces with obstacles in their paths. Since the system utilizes ultrasonic echolocation to provide the user the ability to navigate close spaces, it works best when the ultrasonic waveforms are not be interrupted by other potential interferences in the general vicinity. Therefore, the amount of such way finding devices that can operate within the same area is limited. This is not a major concern in designing the system since it’s fairly rare for multiple people who require the assistance of such a system to be in the same space at once.

The device is powered by a 9V battery; therefore, it does not depend on an immobile power system. However, in order to ensure an uninterrupted working condition, the user is advised to carry a backup battery.

## System Input and Output Specification

### System Inputs

The system supports the following basic inputs from the user:

* Power on [User controlled] – On condition for the system
* Power off [User controlled] – Off condition for the system
* Acknowledge button
* Panic button

The internal system input, which does not rely on user specification:

* Echoed Signal [Internal system] – The input needed to determine the distance from the obstacles
* PWM signal for servo motor.

### System Outputs

The system main outputs are the haptic feedback provided to the user:

* Vibration left – Left motor vibration to indicate obstacle on the left side.
* Vibration right – Right motor vibration to indicate obstacle on the right side.
* Vibration center – Center motor vibration to indicate obstacle on the right side.
* Vibration hand – Hand-attached motor vibration to indicate obstacle in the direction pointed to by the user
* Bluetooth module – Send data to remote monitor via terminal

The internal system output, which does not rely on user specification:

* Transmitted Signal [Internal system] – The output from the system to echolocate the closest obstacles

### User Interface

The user has control of these inputs:

* On button – Allows the user to turn the system on
* Off button – Allows the user to turn the system off
* Reset – Allows the user to reset the system
* Acknowledge button – Allow user to prompt system to continue scanning in front
* Panic button – Allow user to prompt monitoring system for help

All outputs of the system will be in the form of haptic feedback or audio responses

* Vibration left
* Vibration right
* Vibration front
* Vibration hand

## Use Case Diagram



Figure 1. Use case diagram

### Textual Description

System On/Off:

Power the system on and off by removing a power supply using a switch

Exception: No power supply, battery dead

Reset:

Resets the system, mainly for debugging or if a bug occurs

Exception: Initial start-up of device, not battery, system off

Acknowledge button:

User took suggested route and directs sensor to point in front of the user and keep scanning

Exception: No battery. System off

Panic button:

Notifies the monitoring system that assistance is needed.

Exception: No battery. System off

Vibration Feedback:

Provide haptic feedback to one of the three motors on the device using a PWM output to control the intensity of the motor which will change according to the distance away from the object

Exception: System off. Not enough power

Ultrasonic Pulse Transmission:

Emit an outgoing pulse to determine the distance by measuring the time it takes to reach back to the receiver.

Exception: System off. Not enough power

## System Functional Specification

The ultrasonic wayfinder is intended to allow the user the ability to navigate close spaces without the dependency on the heavier equipment such as the white cane. The system also gives the user a sense of spatial awareness that the traditional cane was not able to. The control of the system is relatively simple as the only task the user is required to do is equipping the necessary items, then turn on the system.

The output of the system includes haptic feedbacks to the four motors; the conditions and intensities of these feedbacks correspond to the environment in which the system is used. Using a servo motor to turn the ultrasonic sensor, the space right and left of the user can be scanned for obstacles and suggest the right path the user should take. The motor only scans the left and right when an obstacle gets too close to the front of the user. An acknowledge button is pressed after the user decides on which path to take to direct the sensor to point forward and keep scanning for more obstacles. A panic button will be used to notify a remote monitoring system to call for help.

The hand-mounted motor vibrates whenever the user points their hand-mounted sensor at a nearby object. This hand-mounted device represents the traditional cane, but it is less wieldy and it offers the user more spatial awareness for any direction the user may want.

## Operating Specifications

The system will be expected operate in a standard commercial environment. In the cases of harsh weather such as rain, snow, or other extremities, the system is not expected to operate due to the nature of it being an electronic device. A weatherproof prototype may be developed in the future if the product performs exceedingly well in the initial development phase.

Temperature range

* 20 – 80°C

Humidity

* 20 – 60% non condensing

Pressure

* 1 atmosphere

Power

* 9 VDC

## Reliability and Safety Specification

Because this system might ultimately be used by a person with disability, safety is one of the main design concerns. The following standards are followed during the design and implementation of the system:

* ANSI/ISA S82.02.02:1996 – Safety standard for electrical and electronic testing
* ECMA-287 – Standard of electronic equipment
* EN 50065-1:1991 – Signaling on low-voltage

During the current phase of the design, MTBF can’t be precisely estimated. The operation is expected to be working as intended for 1 month per demo purposes.

# Design Specification

## System **Description**

The design specification describes an implantation of an ultrasonic-range-finding device that utilizes haptic feedback to alert its user about obstacles in the user’s path. The device will consists of at least two ultrasonic transmitter/receiver sensors that can emit short pulses of ultrasonic frequency sound. The receiver picks up the echoed pulses. The time delay between the emission of the initial pule and when the pulse is received gives the system the necessary data to determine how far away the obstacle is from the sensor, which can be described to the user by vibrating a small motor at the intensity proportional to the distance. This gives the user a sense of spatial recognition in almost all kind of environment. The sensor will be on a servo motor so it can scan for objects in front, left and right of the user and control different motor’s intensity. The motor will only turn the sensor when an object is too close in front of the user. Also a hand-mounted module will be implemented so the user can directly point at an area they want to scan. The head module will also send data wirelessly to a remote monitoring system for assistance when the user is in trouble. The project is purposed as a proof-of-concept for a product that can be used by sight-impaired individuals to detect obstructions.

## Specification of External Environment

The system is designed to operate in any dry environment. But its main purpose is to help the user navigate spaces with obstacles in their paths. Since the system utilizes ultrasonic echolocation to provide the user the ability to navigate close spaces, it works best when the ultrasonic waveforms are not be interrupted by other potential interferences in the general vicinity. Therefore, the amount of such way finding devices that can operate within the same area is limited. This is not a major concern in designing the system since it’s fairly rare for multiple people who require the assistance of such a system to be in the same space at once.

The device is powered by a 9V battery; therefore, it does not depend on an immobile power system. However, in order to ensure an uninterrupted working condition, the user is advised to carry a backup battery.

## System Input and Output Specification

### System Inputs

The system supports the following basic inputs from the user:

* Power on [User controlled] – On condition for the system
  + 9 V - Voltage threshold + or – 0.5 V
* Power off [User controlled] – Off condition for the system
  + 0 V - + or – 0.01 V
* Acknowledge button
  + Switch active high, Power source, ground
* Panic button
  + Switch active high, Power source, ground

The internal system input, which does not rely on user specification:

* Echoed Signal [Internal system] – The input needed to determine the distance from the obstacles
* PWM signal for servo motor.
  + 0 to 100% duty cycle + or – 0.02%

### System Outputs

The system main outputs are the haptic feedback provided to the user:

* Vibration left – Left motor vibration to indicate obstacle on the left side.
  + PWM output duty cycle + or - 0.02%
* Vibration right – Right motor vibration to indicate obstacle on the right side.
  + PWM output duty cycle + or - 0.02%
* Vibration center – Center motor vibration to indicate obstacle on the right side.
  + PWM output duty cycle + or - 0.02%
* Vibration hand – Hand-attached motor vibration to indicate obstacle in the direction pointed to by the user
  + PWM output duty cycle + or - 0.02%
* Bluetooth module – Send data to remote monitor via terminal

The internal system output, which does not rely on user specification:

* Transmitted Signal [Internal system] – The output from the system to echolocate the closest obstacles

### User Interface

The user has control of these inputs:

* On button – Allows the user to turn the system on
* Off button – Allows the user to turn the system off
* Reset – Allows the user to reset the system
* Acknowledge button – Allow user to prompt system to continue scanning in front
* Panic button – Allow user to prompt monitoring system for help

All outputs of the system will be in the form of haptic feedback or audio responses

* Vibration left
* Vibration right
* Vibration front
* Vibration hand

The system is not designed to include a display for the user. A monitoring system will be sent data wirelessly from the device to a terminal. The terminal displays the following information on the remote side:

|  |
| --- |
| 12 cm  Warning!!! Obstacle 20 cm ahead  Scanning left…  30 cm  Scanning right…  10 cm  User should turn left…  Panic!  K  Help on the way! |

## System Functional Specification

The ultrasonic wayfinder is intended to allow the user the ability to navigate close spaces without the dependency on the heavier equipment such as the white cane. The system also gives the user a sense of spatial awareness that the traditional cane was not able to. The control of the system is relatively simple as the only task the user is required to do is equipping the necessary items, then turn on the system.

The output of the system includes haptic feedbacks to the four motors; the conditions and intensities of these feedbacks correspond to the environment in which the system is used. Using a servo motor to turn the ultrasonic sensor, the space right and left of the user can be scanned for obstacles and suggest the right path the user should take. The motor only scans the left and right when an obstacle gets too close to the front of the user. An acknowledge button is pressed after the user decides on which path to take to direct the sensor to point forward and keep scanning for more obstacles. A panic button will be used to notify a remote monitoring system to call for help.

The hand-mounted motor vibrates whenever the user points their hand-mounted sensor at a nearby object. This hand-mounted device represents the traditional cane, but it is less wieldy and it offers the user more spatial awareness for any direction the user may want.

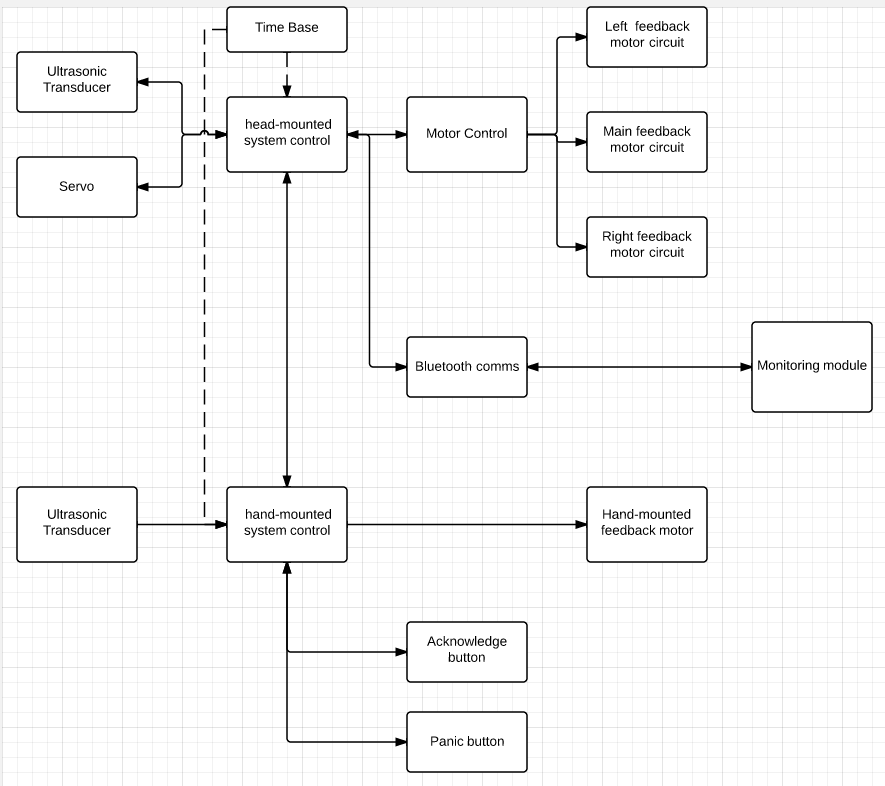


Figure 2. High level block diagram

**Input Subsystem** – The input subsystem includes the one ultrasonic receivers and the two switches for acknowledgement and panic function. These two apparatus collect the data required for the operation of the wayfinder. They route the input signals to the appropriate portion of the control subsystem, which is the main PIC microcontroller.

**Output Subsystem** – The output subsystem is the driver for the vibrator plates, and the ultrasonic transmitter. This portion of the system maintains the logic necessary to convey the necessary data to the user. Also the servo motor is moved by the head module processor when an object gets too close to the ultrasonic sensor which then causes the sensor to scan left and right.

**Time Base** – The time base is implemented as external 20MHz crystal oscillator, input to the main system through a one-wire connection. The crystal oscillator operates on a standard 5V input, similar to the main system. The time base provides the frequencies that are used to define the measurement windows for events such as the transmission rate of the transmitter, the measurement timing of the receiver as well as the idle time of the sensors. The PWM module, which drives the vibrators also operate based on the main system clock.

Frequency

* 20 MHz + or – 10 Hz

Period

* 5e-8 seconds + or – 1 ns

Events

Ultrasonic Transmission

* + Frequency – 17 Hz + or – 0.5 Hz
  + Period – 60 ms + or – 1 ms

Ultrasonic Reception

* Frequency – 17 Hz + or – 0.5 Hz
* Period – 60 ms + or – 1 ms

**Logic Subsystem** – The logic portion provides the main logic and control flow to the rest of the system.

When the system transmits an ultrasonic pulse, the main logic system starts a timer, which indicates the start time of the ranging period. Once the echoed pulse has been receive, the logic subsystem stops the counter, giving the system the necessary data to calculate how far away the wave has travelled before coming back to the device.

After measuring the distance, the logic subsystem determines the appropriate outputs. These output controls are fed into the output subsystem, completing one cycle of the control and logic task.

**Power Supply Subsystem** – The system will be powered by a 9 V battery. All three subsystems above operate on a 5 V input, which means it is essential for the supply subsystem to regulate the input voltage from the battery.

At power on, there shall be a negative going reset signal. The signal will remain in the low state indefinitely until an off signal is issued by the user.

## Operating Specifications

The system will be expected operate in a standard commercial environment. In the cases of harsh weather such as rain, snow, or other extremities, the system is not expected to operate due to the nature of it being an electronic device. A weatherproof prototype may be developed in the future if the product performs exceedingly well in the initial development phase.

Temperature range

* 20 – 80°C

Humidity

* 20 – 60% non condensing

Pressure

* 1 atmosphere

Power

* 9 VDC

## Reliability and Safety Specification

Because this system might ultimately be used by a person with disability, safety is one of the main design concerns. The following standards are followed during the design and implementation of the system:

* ANSI/ISA S82.02.02:1996 – Safety standard for electrical and electronic testing
* ECMA-287 – Standard of electronic equipment
* EN 50065-1:1991 – Signaling on low-voltage

During the current phase of the design, MTBF can’t be precisely estimated. The operation is expected to be working as intended for 1 month per demo purposes.

# Design Procedure

The first module designed was the head-mounted module. The system had to be able to read and analyze input from the ultrasonic sensor and control three vibration motors using PWM outputs to indicate left, right and center obtrusions. Then data would also be sent to a remote terminal via Bluetooth. This module consists of an 18F25K22 PIC microprocessor, ultrasonic sensor, Bluetooth module, three vibration motors, and a servo motor. The microprocessor uses the input from the ultrasonic sensor to control the device and give correct signals to the remote terminal via Bluetooth module. The servo motor is designed to scan left then right when an object approaches the user too close in front of them. The next step is to take in data from the hand-mounted module to tell the system when the user acknowledges a suggested path so the sensor can revert back to scanning the center being sent through I2C.

The second major module designed was the hand-mounted module. The design is similar to the head-mounted module. It consists of an 18F25K22 PIC microprocessor, two switches, ultrasonic sensor and one vibration motor. The user activates the active high switch for acknowledgement to the head-mounted module via I2C and also an active high panic switch when user is in trouble through I2C. Then the motor is controlled through a PWM output of the processor which varies the duty cycle depending on the distance reading analyzed through the input of the ultrasonic sensor.

# System Description

This specification describes and defines the design requirements and specifications for an Ultrasonic Wayfinder. The system comprises of a head-mounted module, hand-mounted module, and a Bluetooth module. The head-mounted module provides a user interface to a remote terminal via Bluetooth for a monitoring system where a family member of the primary user can see live data and situation they are in. The processor on the head module reads in data from an ultrasonic sensor and interprets the data to find the distance away from the closest object from the sensor to send to the remote monitoring system. When the ultrasonic sensor detects something in front of the user that is fairly close and needs to be avoided, it will activate a servo motor to turn the sensor left then right to scan for obstacles on both sides and accordingly suggest a path to the user by vibrating more intensively on either the right motor or left motor to indicate something is closer when intensity increases. The hand module will also consist of a ultrasonic sensor and processor that determines the distance of an object that the user points to which will vibrate the motor on the hand module according to how close the object is. The user also has two switches on the hand module. One is for acknowledgement of the suggested path and the other is for the panic swtich to notify the monitoring system that they need help. These signals are sent via I2C back to the head module.

The system inputs are ultrasonic sensor data, acknowledge switch, and panic switch. Ultrasonic sensor is always inputting data to the processor to monitor distance from closest object. The acknowledge switch is to notify the servo motor to continue scanning in front of the user after the user has been suggested a new path (left or right) because an object became too close in front of the user. The panic switch is for the user to notify the monitoring system that help is needed.

The system outputs are 4 different PWM outputs with varying duty cycles, data being sent through Bluetooth to remote terminal. The PWM outputs are for specific direction of vision. They are for notifying the user how close an object is in front, left, right, and/or pointing at by increasing their respective PWM output’s duty cycle the closer an object gets which will be used to increase the intensity of each vibration motor. The data being sent to remote terminal would be calculated data the head module produced to show how far an object is from the user and what direction and other useful messages.

The system side effects are the use of one ultrasonic sensor to control 3 different outputs and relying on the user acknowledgment to continue scanning normally in front of them.

# Software Implementation

## Head-mounted PIC18F25K22 Microcontroller

Two PIC18F25K22 MCUs were implemented as the main controllers for the hardware of the ultrasonic pathfinding device. The main PIC18F controlling the head-mounted subsystem was programmed to start by first initializing all the internal and external peripherals required for the operation of the module.

### Setup

To ensure that the initialization is done in the most efficient way, the setup of each peripheral is done via setting the individual bits within the corresponding control registers. First and foremost, the global interrupts are configured and enabled by setting the control bits RCONbits.IPEN, INTCONbits.GIE, and INTCONbits.PEIE. We determined that interrupts within our system should all be treated equally important, as no interrupt should be able to overwrite another. The system does not include any analog capabilities. Therefore, all pins are initially set to be in digital mode.

Due to the nature of the system, timing is an incredibly important aspect. Within the head-mounted subsystem, four different timer peripherals are utilized to provide the precise timing needed. All timers are configured to operate in 16-bit mode and run on the instruction clock cycle. Timer0 prescaler is set to 1:128. Both timer1 and timer3 utilize a prescaler value of 1:4. And timer2 runs at full speed with prescaler 1:1.

The servo’s driver signal is an output of pin 2 (RA0). The main transducer’s trigger is controlled with pin 4 (RA2), which is configured to be used as an output pin. The transducer’s echo signal is captured by pin 13 (RC2), which is mapped to the CCP1 peripheral. We configured this peripheral to work in capture mode. Three more CCP modules (CCP2, CCP3, and CCP4) are initialized as output to provide the PWM signal to the vibration motors. The time base for these PWM modules is provided by timer2.

Lastly, the I2C and USART peripheral are configured to provide the communication links between the MCUs and remote monitor.

After all the pins and peripherals are initialized. The main system can enter its operational loop.

### Distance Measurement

The measurement subsystem is the core component of the overall design. In order obtain a measurement, the ultrasonic transducer is provided a trigger signal through the output pin 4. The trigger pulse of width 10uS is sent to the transducer by setting port RA2 to high using the register control command LATAbits.LATA2 = 1, followed by a software delay of 70 instruction clock cycles, the port is then set back to low.

After the trigger pulse has been issued, the transducer transmits 8 cycles of 40kHz ultrasonic bursts and wait for the reflected ultrasonic burst. When the sensor detected ultrasonic from receiver, it will set the echo pin (RC2) to high and delay for a period (width) proportional to the time between transmission and receipt then set it back to low. The echo pin is mapped to a capture module. This capture module generates an interrupt when the value at the pin changes from low to high. In this initial interrupt handler, we save the value of timer1 as the “start time” using the helper function ReadTimer1() in the library “timers”, then set the capture module to generate an interrupt on the negative edge following this event. When the pin is set back to low, another interrupt occurs. Again, we read the value of timer1; this time, the value is stored as “end time”. The difference between end time and start time is the time that it takes for the ultrasonic burst to travel to the obstacle, then back to the receiver of the transducer. Since each tick of timer1 equates to .8 uS, the duration is converted to useful metric (centimeter) using the formula

distance\_cm = timer1Counts \* .8uS / 58

The distance measurement is fed to the output subsystem. The procedure is repeated until the user encounters an object less than 20 centimeters away. If this occurs, the system temporarily stops scanning the obstacles ahead of the user. A flag is set to initiate scanning routine that measures the distances on both sides of the user. This procedure is done by first rotating the servo to the left.

The servo control signals are 50Hz PWM pulses. Since we are utilizing our processors at a speed of 5 MHz, it’s practically impossible for us to drive the servo using a PWM output of a CCP module. A solution is to use a CCP module in compare mode which allows us to generate interrupts on a set interval. In setup, we configured CCP2 to be the compare module which drives the servo motor. The time base for the CCP2 module is provided by timer3. The compare module continuously compare the value of timer3 to the value stored in the registers CCPR2H:CCPR2L. When a match is found, the module generates an interrupt. To rotate the servo 90 degrees to the left, a pulse train of width 2.4 mS has to be continuously supplied until the servo completes the rotation leftwards. To do this, we initially pulse the servo control pin (RA0) to high, then set the value CCPR2H:CCPR2L to 000101110111000, this value represents number of count of timer3 that equates to 2.4 mS. After the value is set, we reset the value of timer3 to 0 to start the timing routine which allows the control signal to stay high for precisely 2.4 mS. When timer3 reaches the value specified, it generates an interrupt. In the interrupt handler, we set the servo control pin back to low, then reset the CCPR2H and CCPR2L pair to the binary value 0101010111110000 representing 17.6 mS. At the end of the 17.6 mS interval, we repeat the procedure by setting the servo control pin to high and the compare value to represent 2.4 mS. The routine generates the desired pulses to turn the servo all the way left. Correspondingly, the servo can be rotated 90 degrees to the right if provided a pulse train of width .6 mS. The pair of compared values to generate this pulse is 0000001011101110 and 0101111010111010.

With the servo properly set up and controlled, we can start scanning the left and right areas. After the flag indicating the start of the routine is thrown, we set the system to rotate the servo leftwards. After setting the servo all the way left by setting the compare values described above, we generate a software delay of .5 seconds in order for the servo to completely rotate. After which, we perform one complete routine of obtaining one single distance measurement. This measurement is then stored as “distanceLeft\_cm”. Similarly, to obtain the distance to the right of the user, we rotate the servo towards the right, then perform another distance measuring routine. This time, we store the measurement as “distanceRight\_cm”. By comparing these two values, we can determine which direction is more suitable for the user to travel. Then we send this calculation to the output subsystem, which relays the information to the user and the remote monitor.

For the hand-mounted module, the measurement routine is performed the same way. However, we do not use a servo to control the movement of the transducer as it’s up to the user which way they might want to point the transducer.

### Output

The output of the system includes the vibration motors on each of the head-mounted and hand-mounted modules, and the remote monitoring system.

The vibration motor for the head-mounted subsystem is driven by the PWM signals generated by the CCP3 module of the PIC18f. The module is configured to utilize timer2 as the time base. The output of the PWM is inversely mapped to the range of 0 to 100 cm. if an obstacle is recognized at 100 cm, the vibration motor will start to vibrate gently. As the user approaches the obstacle, the motor’s intensity increases. The intensity of the motor is based on the duty cycle of the PWM pulses. In order to generate a variable pulse width PWM, we wrote a helper function feedback( int duration ). The function accepts the distance from the user to the obstacle as an input, then generate a PWM with the corresponding duty cycle using the formula

duty cycle = (100 - duration) \* 10 / 1000

We use this duty cycle value to configure the PWM control bits CCP3CON.

Another output of the program is the remote monitor. As the system continually measure the distances and prompting the user of the optimal path, it streams these measurements and evaluations to the remote monitor where another person can view the activities of the user. In order to stream the information to the monitor, all the integers from the measurement routines have to be converted to strings. We do this with the help of the library function itoa(). Using itoa, we convert the measurements to a string format, which is then fed into the function puts1USART() in the USART library. This function streams the string to the monitor through the UART connection. For the various different warnings and acknowledgement, static strings are constructed to provide the according messages when needed.

The figure below shows the data flow of the ultrasonic sensor and how it controls the vibration motor.

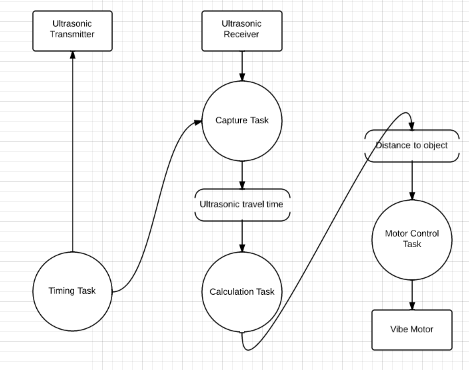


Figure 3. Ultrasonic data flow

### Communication

The communication between the two PIC18f MCUs is done via an I2C connection. The head-mounted MCU acts as the slave while the hand-mounted MCU is positioned as the master. Whenever the user presses the panic button on the hand-mounted module, an interrupt is generated that tells the program to relay the button press information to the head-mounted unit. The transmission of the information is done by first starting the I2C communication using the command StartI2C1(). After the communication channel is reestablish, the address of the slave MCU(0xB0) is passed onto the SDA line using putcI2C1(). The slave MCU generates an interrupt when the SDA line is filled with the address. In the interrupt handler, the slave checks the address that just got passed onto the SDA bus. After confirming the address, the slave now knows that it’s tasked to receive a data byte following the address. Now, the master sends the byte data through the SDA line using another putcI2C1() command. Another interrupt is generated on the slave, this time, it stores the data byte into a global character variable called “userInput”. After the information is stored, a integer flag called newUserInput is set to signify that there’s a new user input to be processed and relay back to the monitor. If no button is pressed, the I2C channel stays idle until another command is issued.

The communication channel between the head-mounted MCU and the remote monitor is implemented using a Bluetooth communication protocol. Bluetooth is essentially a wireless implementation of the UART protocol. We establish the connection by setting the RX1 and TX1 of the MCU. After the connection is establish and the communication channel confirmed by the remote monitor (using their native Bluetooth connection), new information can be passed through the communication channel. To send a byte of data to the remote monitor, a function called Write1USART() is utilized. However, to transmit more than one byte of data, we must utilize the function puts1USART(). The entire communication is done by converting information to string then relaying the char or string data.

# Hardware Implementation

The first major hardware component is the head-mounted module. The module consists of a PIC18F25K22 MCU, a 20 MHz external crystal oscillator, a servo, a transducer, a Bluetooth module, and three vibration motor driver circuits. The external crystal oscillator provides the main time base for the MCU. The transducer collects the distance measurements. The servo rotates the transducer to assist with the task of measuring the distances both to the left and to the right of the user. The Bluetooth module creates a UART communication channel between the MCU and a remote monitor. The vibration motor driver circuits are used to provide haptic feedback to the user.

In this module, the crystal oscillator’s CLKI pin is connected directly to the MCU’s pin 9. The servo motor control wire is connected to the MCD’s pin 2. The Vdd pin on the servo is connected to +5 V rail. The Gnd pin on the servo motor is connected to the GND rail. The trigger on the ultrasonic is connected to pin 4 on the MCU. The echo on the ultrasonic is connected to pin 13 on the MCU. The Vdd pin on the ultrasonic is connect to +5 V rail. The Gnd pin on the ultrasonic is connected to the GND rail. Pin 8 and Pin 19 on the MCU is connected to the GND rail. Pin 14 on the MCU is connected to a 4.7 Kohm pull up resistor and is also connected to Pin 14 on the hand mounted module MCU. Pin 15 on the MCU is connected to a 4.7 Kohm pull up resistor and is also connected to Pin 15 on the hand mounted module MCU. Pin 17 is connected to TXo on the Bluetooth. Pin 18 is connected to RXi on the Bluetooth. Pin 20 is connected to the +5 V rail. Pin 6, 21, and 26 is connected to the gates on different MOSFETs.

In the hand mounted module, the trigger on the ultrasonic is connected to pin 4 on the MCU. The echo on the ultrasonic is connected to pin 13 on the MCU. The Vdd pin on the ultrasonic is connect to +5 V rail. The Gnd pin on the ultrasonic is connected to the GND rail. Pin 8 and 19 is connected to the GND rail. Pin 20 is connected to the +5 V rail. The crystal oscillator’s CLKI pin is connected directly to the MCU’s pin 9. Pin 26 is connected to the gate on another MOSFET.

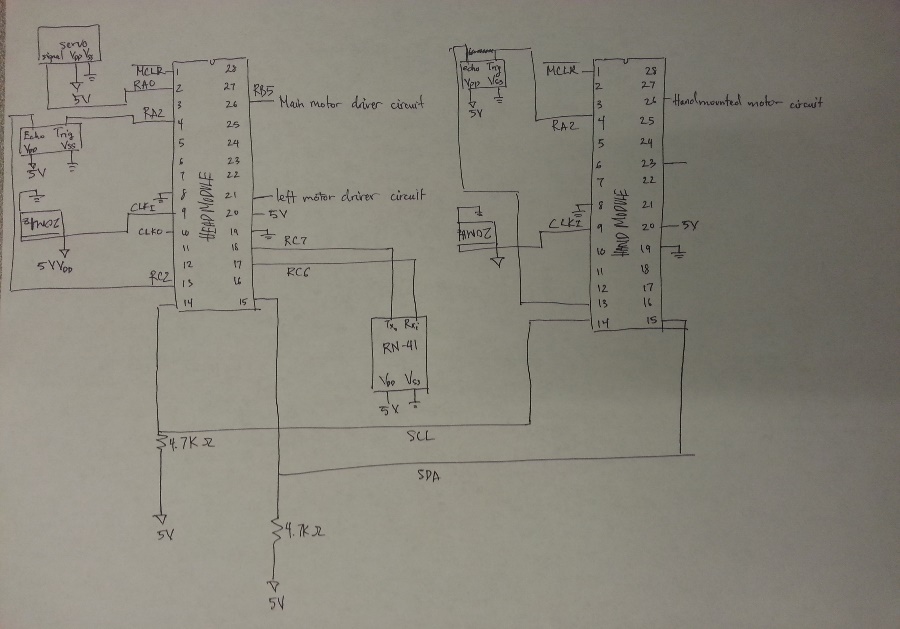


Figure 4. Hardware block diagram

This is the motor driver circuit. In the design implemented for this project, The Schottky Diode, bypass capacitor, and Pull-Down resistor was left out. The next design shown below was not implemented due to inefficient amount of time. The problems with the original motor driver design was found after connecting all 4 vibration motors together. This affected the PICs because of inductive interference which cause the PICs to not respond. For future development, this new design would be implemented to correctly drive all 4 vibration motors.

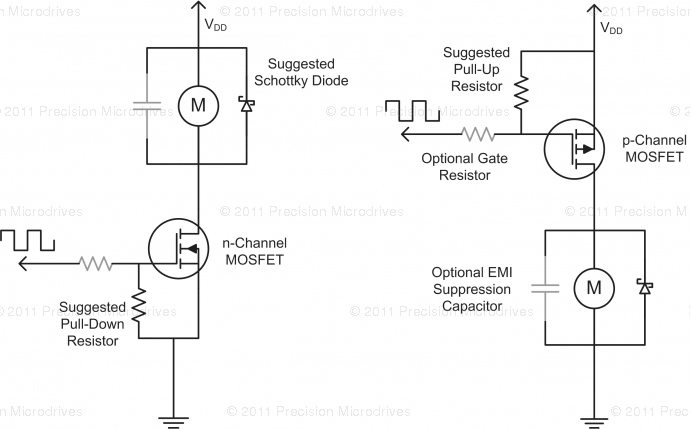


Figure 5. Motor Driver circuit

# Test Plan

The head mounted module needs to be tested to calculate the distance of the object in front of the user, the servo motor turning left and right to scan for surrounding objects, the haptic feedback from the vibration motors, I2C communication. The hand mounted module needs to be tested to calculate the distance of the object in the direction of the hand and the haptic feedback from the vibration motors. The ultrasonic sensors needed to be tested for signal being sent out and receiving it back. The servo motor needed to be tested to rotate left and right when the object is 20 cm in front of the user. The I2C communication between the hand module needs to be tested for the acknowledge and panic signals.

The plan to test if the circuit works is by observing the signals and PWM on the digital oscilloscope first. Then the loads would be applied such as the ultrasonic sensors and the vibration motors.

The ultrasonic sensors needs to be tested for limits on the closest and farthest distance it can detect. The vibration motor needs to be tested for its limits on its max 5 V rating and low 0 V rating. Also, the vibration motor needs to be tested with various voltages to test the intensity of the vibration.

# Test Specification

1. Ultrasonic Sensors

* The input trigger signal on the ultrasonic sensor is 0 V or 5 V.
* The output echo signal on the ultrasonic sensor is 0 V or 5 V.
* The trigger signal comes from the PIC on the head mounted or hand mounted module.
* The PIC must store when the trigger signal is sent to the ultrasonic senor and when the echo signal is received back from the ultrasonic sensor.

1. Servo Motor

* The control signal on the servo motor is a PWM signal from 0 V to 5 V.
* The angle of the servo motor depends on the pulse width of the control signal.
* The pulse width range is 10% to 90%.
* The angle of the servo motor will turn 90 degrees left and 180 degrees right.
* The PIC should be able to drive the servo motor with a 5 V power supply.

1. Vibration Motor

* The input power supply should be in a range from 0 V to 5 V.
* The PWM signal range will be from 0 V to 5 V.
* The PIC should be able to drive the vibration motor with the PWM signal and 5 V power supply.

1. Hand Mounted Module

* The acknowledge and panic button signals will be sent to the head mounted module to disable the right and left prompt vibration motor and the panic button will alert the system.
* When the acknowledge button is pressed, the right or left prompt vibration motor will turn off.
* When the panic button is pressed, a message alerting the system to will be printed to the terminal through Bluetooth.
* The vibration motor will vary in intensity depending on the distance calculated from the ultrasonic sensor on the hand mounted module.

# Test Cases

1. Ultrasonic Sensors

To test if the ultrasonic sensor trigger signal takes an input of 0 V or 5 V and outputs 0 V or 5 V, the digital oscilloscope was connected to check if the echo signal would have a 5 V output when 5 V was inputted to the trigger pin and vice versa with the 0 V. If the digital oscilloscope shows that the echo signal is 0 V when the trigger signal is 0 V then this is correct. If the digital oscilloscope shows that the echo signal is 5 V when the trigger signal is 5 V then this is correct. The trigger signal coming from the PIC was tested by connecting the pin to the digital oscilloscope and checking if the signal is high when the PIC triggers the pin. The storing capabilities of the trigger and echo signals were tested by going through the debugger and checking if the values were set when the trigger signal was transmitted and when the echo signal was received.

1. Servo Motor

To test if the control signal on the servo motor is a PWM signal from 0 V to 5 V, the procedure was to simply output a PWM signal with 50 % and see if the servo motor would rotate. Then the angle of rotation on the servo motor was tested by changing the duty cycle of the PWM signal. Then the limits of the PWM were tested to see when the motor would not respond anymore. After that the PWM was adjusted to accurately turn 90 degrees left and 180 degrees right. The PIC was tested to drive the servo motor with a 5 V power supply by simply check if the servo motor would rotate correctly depending on the duty cycle of the PWM.

1. Vibration Motor

To test if the vibration motor was working correctly, the first step was to vary the input voltage from 0 V to 5 V. With the varying voltage, the intensity of the vibration will change. With 5 V, the motor will vibrate at its max intensity. With 0 V, the motor will vibrate at its lowest intensity. The PWM signal range will be from 0 V to 5 V which will be tested by connecting it to a digital oscilloscope. The PIC will drive the vibration motor and be tested by connecting the vibration motor to the PWM output pin.

1. Hand Mounted Module

To the acknowledge and panic button signals being sent to the head mounted module, the digital oscilloscope will be used to check if the buttons was pushed. Then the vibration motor will be used to test if they turn off after the button was pushed. The left or right vibration motor will turn off when the acknowledge button is pushed. The message on the terminal is tested by checking if a message is displayed when the panic button is pressed.

# Results

## Head Module/ Hand-mounted Module

User Interface:

The figures below shows the monitoring interface for someone remotely watching over the user of the device. The protocol used to send this data was Bluetooth. The head module would send data over through USART protocol to a Bluetooth module and outputted into coolTerm. The monitoring system is designed for another user to check on the primary user remotely. The data they see is the measurement of the distance between the user and the object in front, left or right of them and shows the distance in centimeters. Also shows the route that the system suggests and shows when the user presses the panic button or has not acknowledge the suggested route within 1 minute.

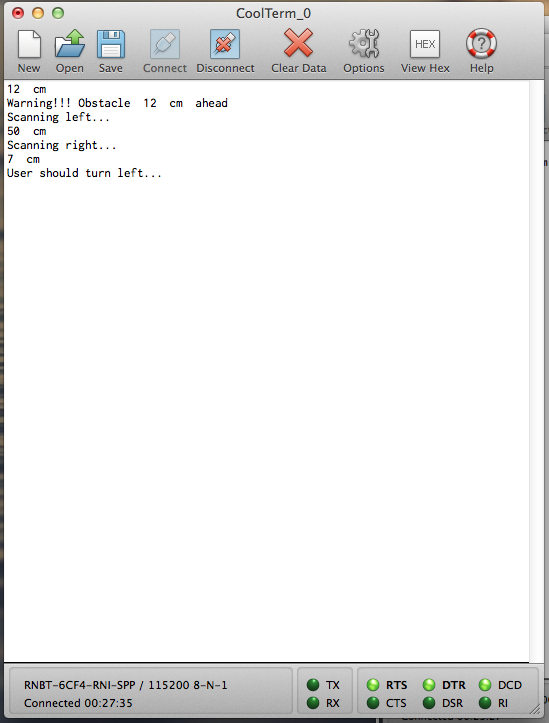


Figure 6. User interface for monitoring system when user should turn left

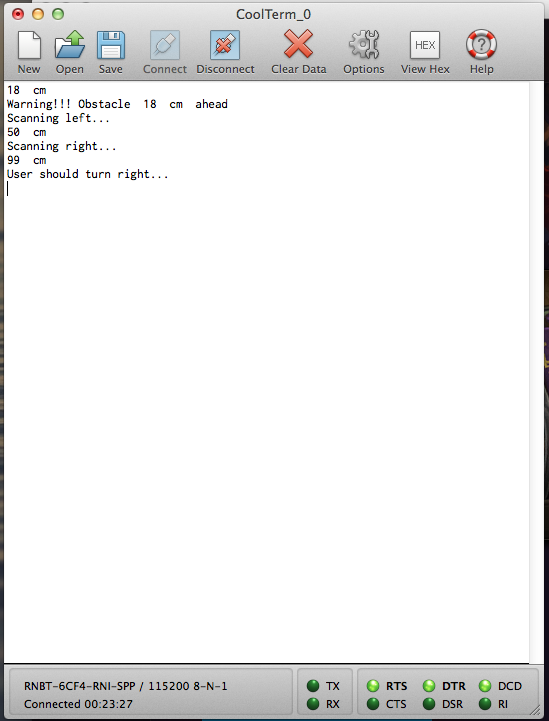


Figure 7. User interface for monitoring system when user should turn right

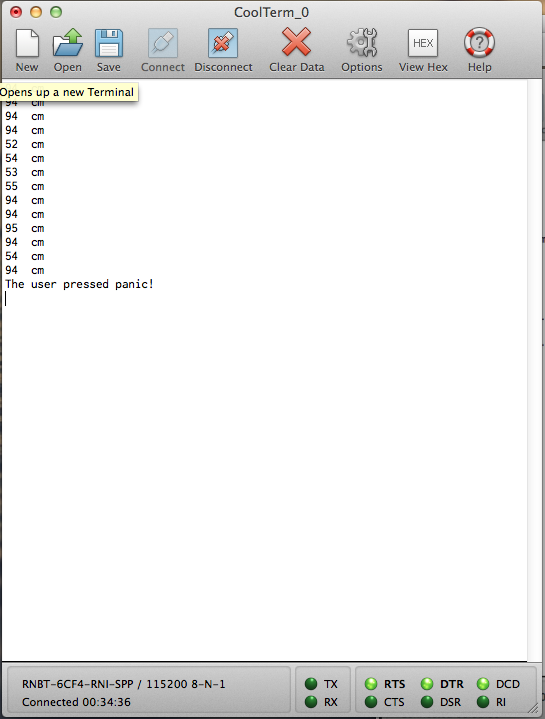


Figure 8. User interface for the monitoring system when the user presses the panic button

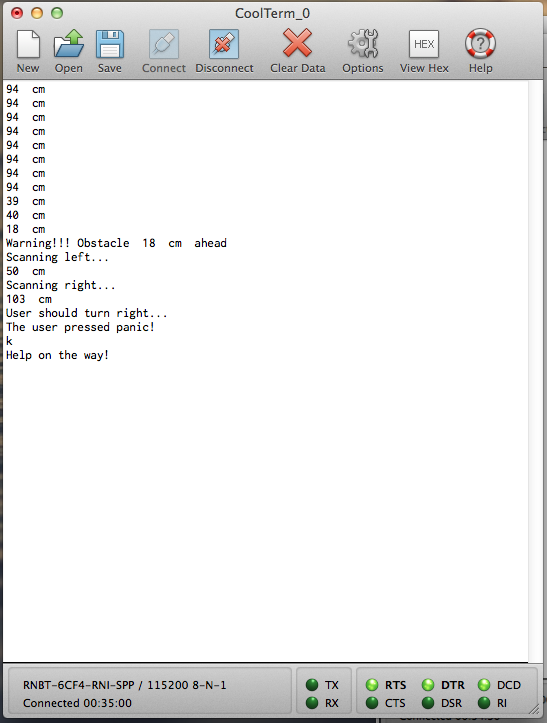


Figure 9. User interface for monitoring system when the character ‘k’ is sent to confirm help

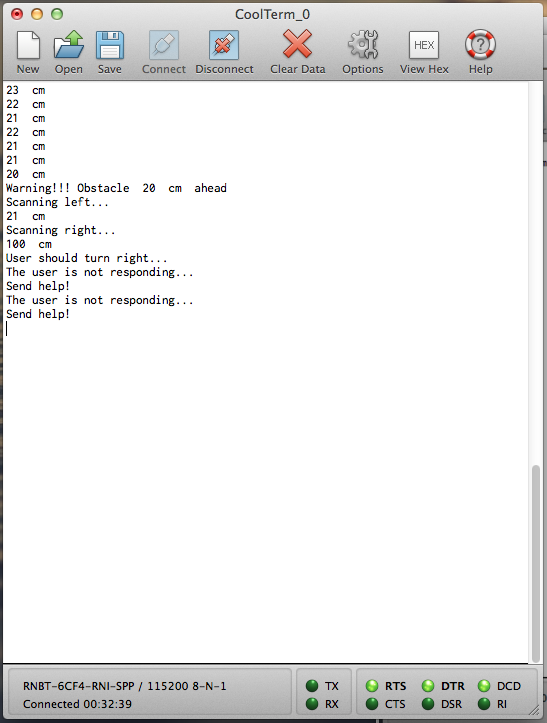


Figure 10. User interface for monitoring system when user does not respond within 1 minute

Front motor control:

The head module outputs a PWM when the ultrasonic sensor sends out a signal and receives it back in a certain amount of time. When the time is shorter the PWM duty cycle is higher indicating a vibrating motor to be more intense to warn the user an object is coming up close in front of them. The figures below show the PWM signal at different distances and the duty cycle they correspond to. When an object is about 25 cm. in front of it the PWM duty cycle is about 75%. For an object about 49 cm. the duty cycle is about 51%. 79 cm away the duty cycle is about 21% and for 95 cm away the duty cycle is about 5%. The hand module calculates the distance of an object the user points at and vibration feedback in the same way as the head module.

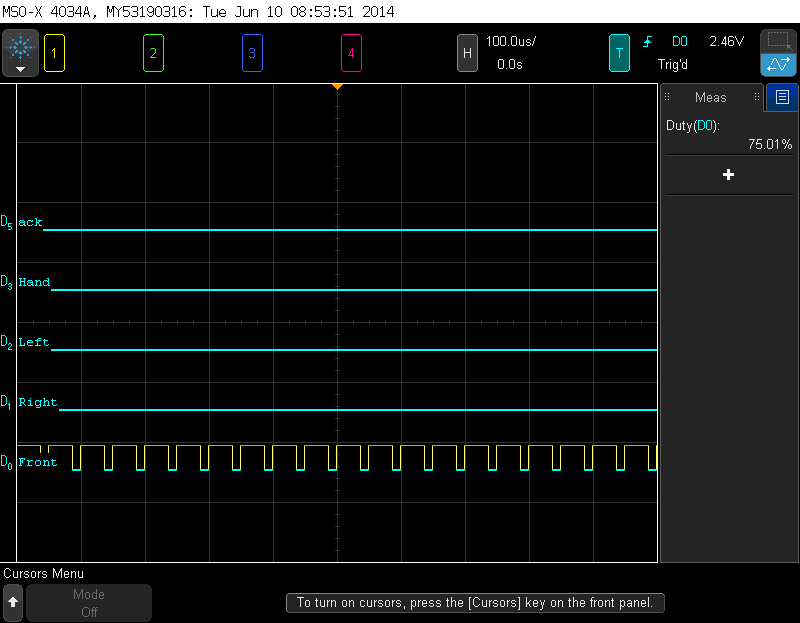


Figure 11. PWM signal when object is 25 cm. away in front of the user

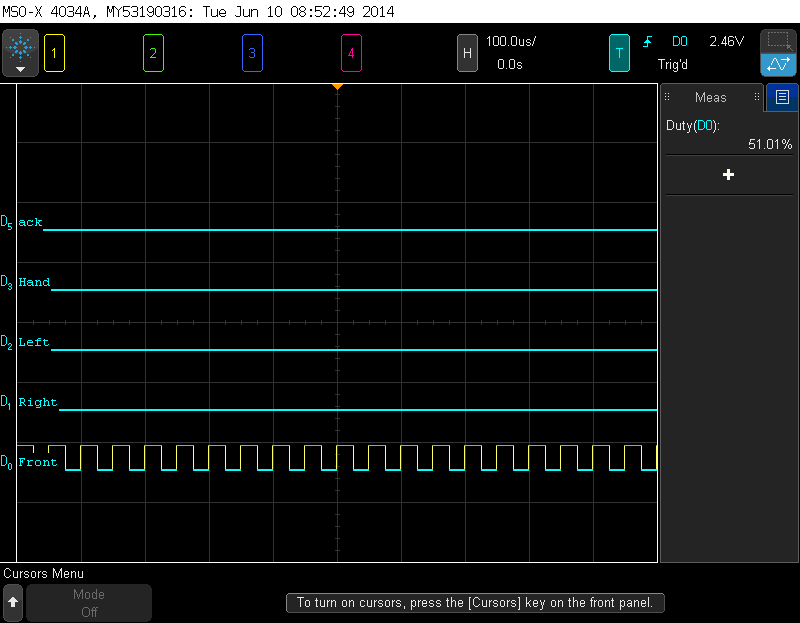


Figure 12. PWM signal when object is 49 cm. away in front of the user

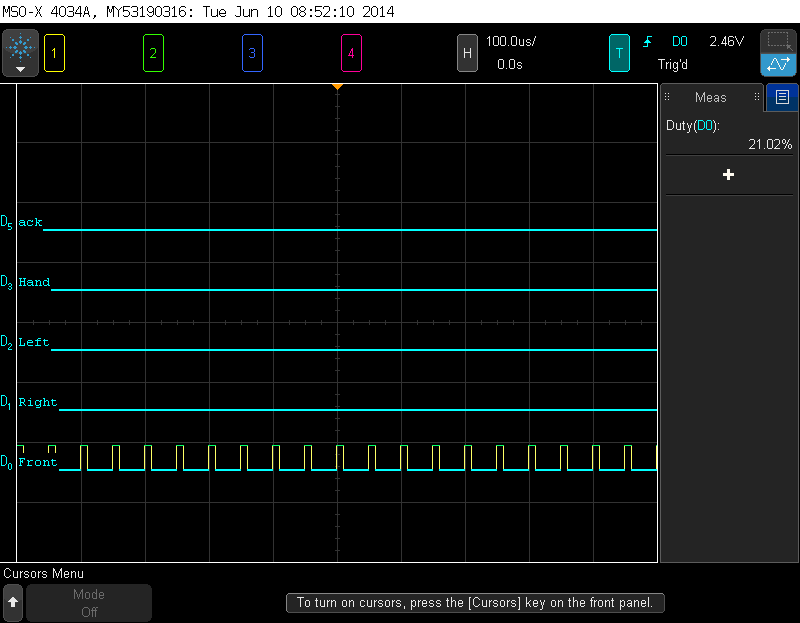


Figure 13. PWM signal when object is 79 cm. away in front of the user

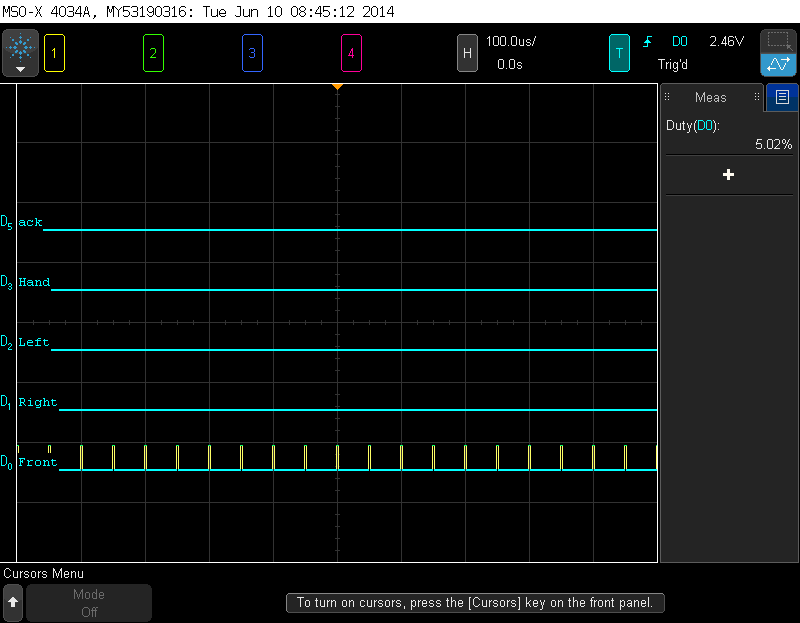


Figure 14. PWM signal when object is 95 cm. away in front of the user

I2C protocol:

The figures below shows the I2C communication between the hand module and the head module. When the user presses the acknowledge button on the hand module the data is sent to the head module notifying the system to look for an object in front of them. The same protocol is used when the panic button is pressed.

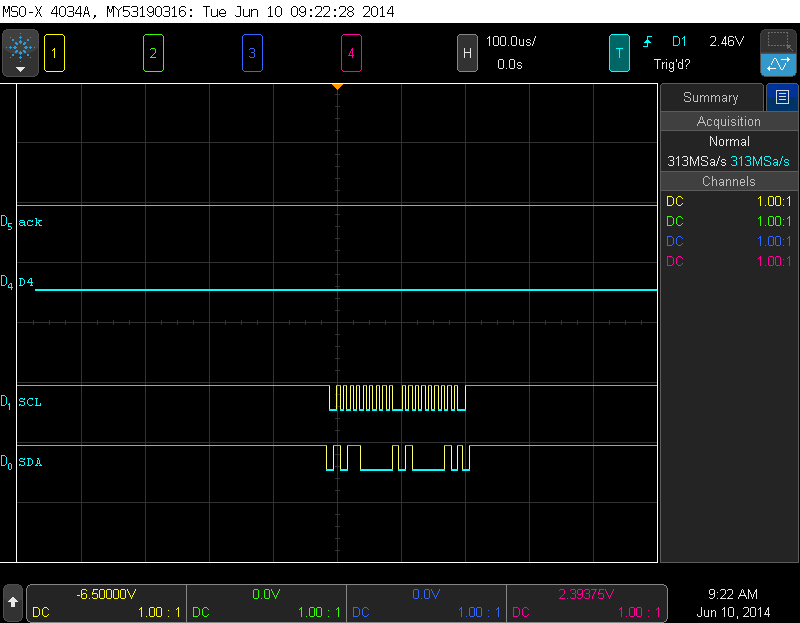


Figure 15. I2C acknowledge button signal from hand mount to head mount

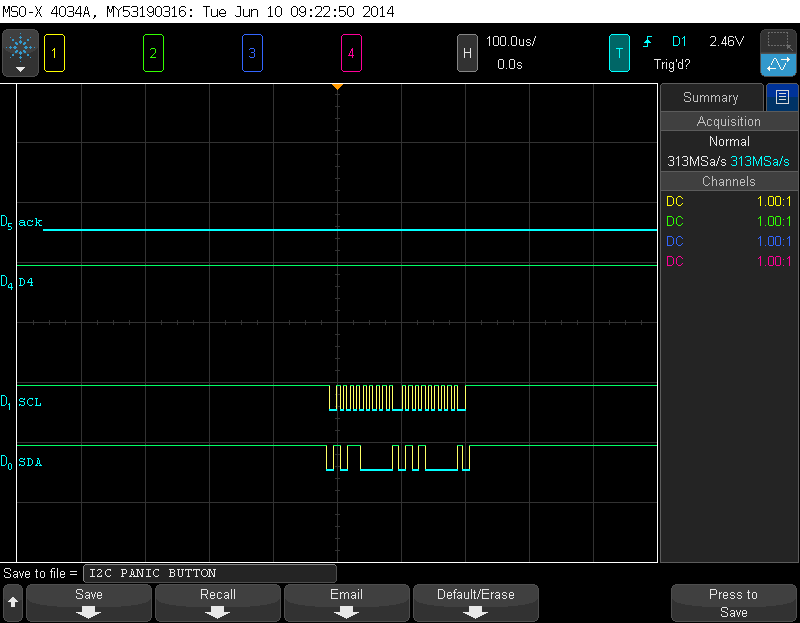


Figure 16. I2C panic button signal from hand mount to head mount

Turn motor control:

The left motor and right motor control works the same way as the front motor and hand-mounted motor control. When the object in front of the user is too close to them the ultrasonic sensor will turn left then right measuring the distance of both sides and outputting a PWM output with a duty cycle depending on the distance of the closes thing. In the figures below they show when the system prompts the user left or right. It clearly shows the difference between the two cases when there is something on the left of the user (causing a PWM output) and when the right PWM output is flat. This indicates the user to go right since it’s the clearest and vice versa.

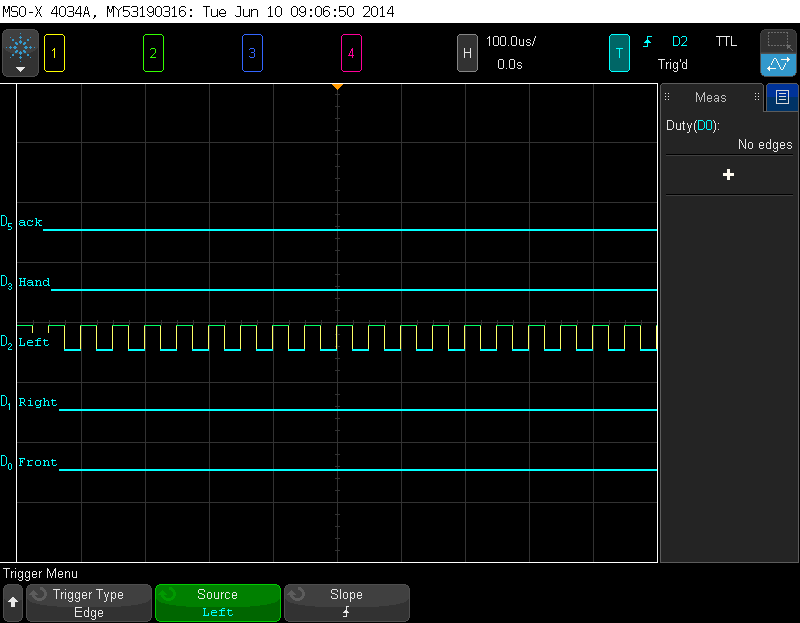


Figure 17. Case when system prompts user the turn right

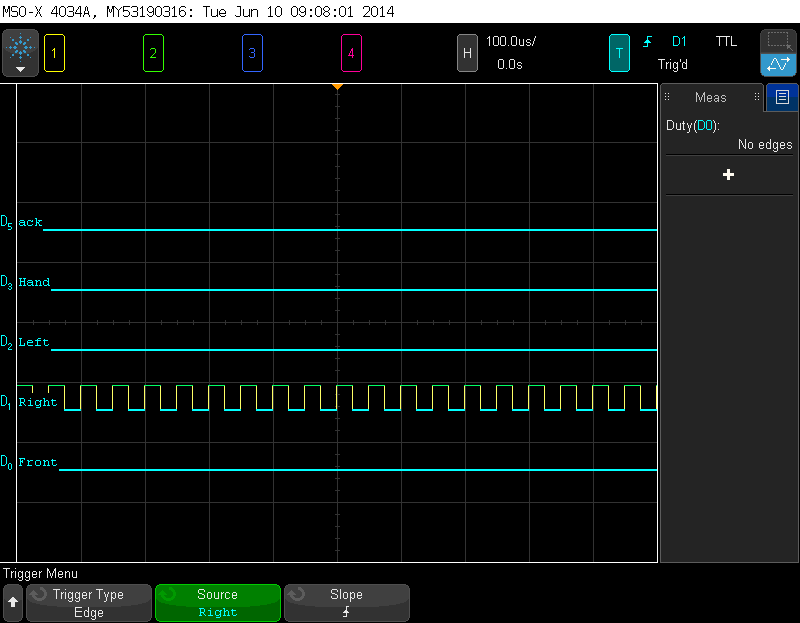


Figure 18. Case when system prompts user the turn left

USART protocol via Bluetooth module:

The data from the head mount processor uses USART to send data to a Bluetooth module which is sent to a remote system displaying information on a terminal as mentioned above. In the figures below wave forms of the transmission and receiving data lines are shown to confirm functionality of the communication. To test communication characters were received from the terminal and sent back. The user of the monitoring system on the terminal side could send a character ‘k’ when the user requests for help in order to confirm that help is on the way.

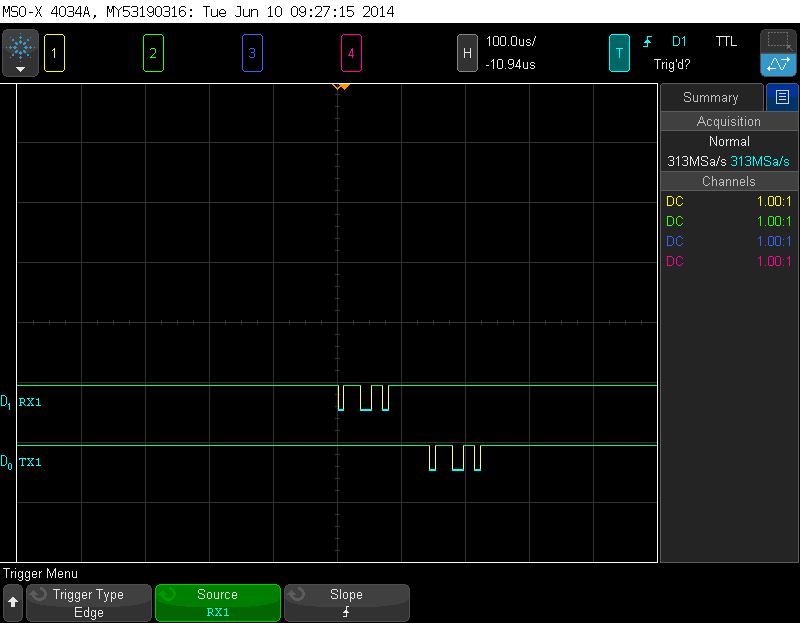


Figure 19. USART receiving the character 'g'

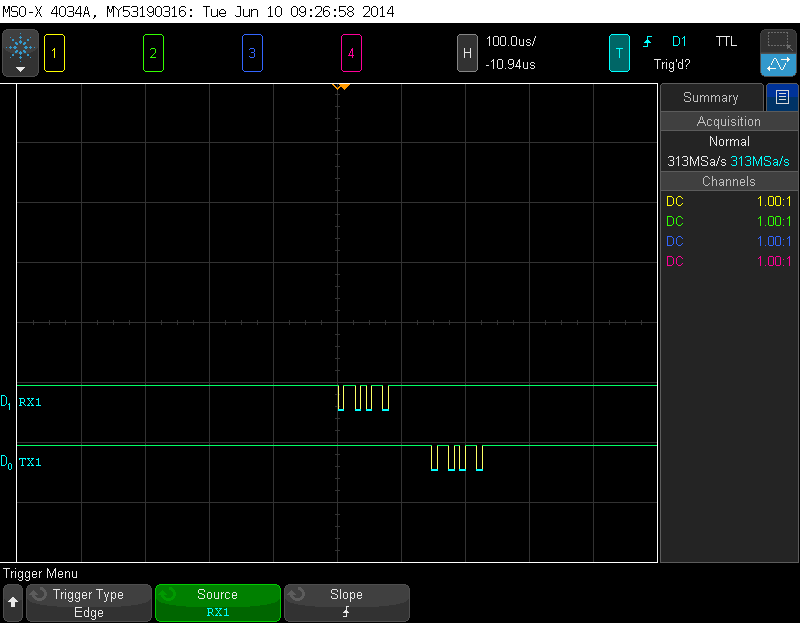


Figure 20. USART receiving the character 'k'

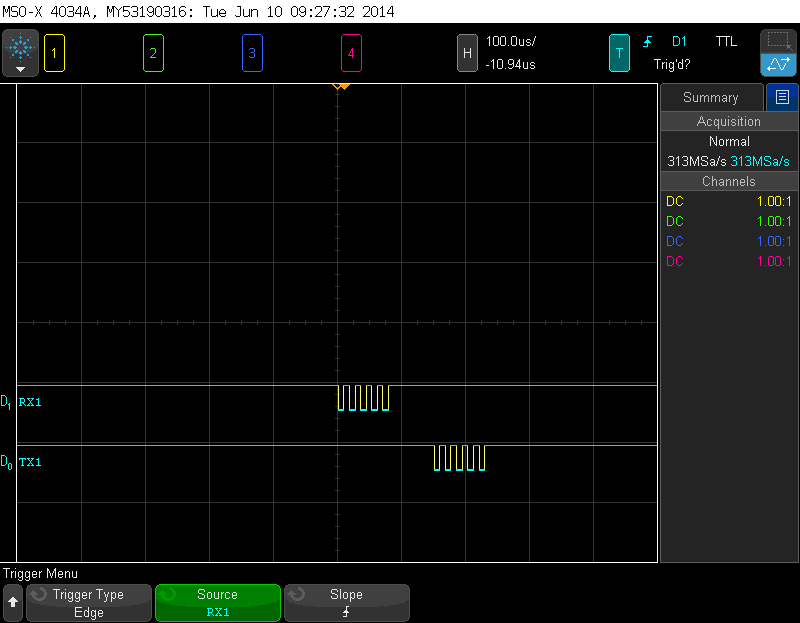


Figure 21. USART receiving the character 'u'

In the figures below they show the head module sending information to the remote terminal to display it. The transmission shows different data transmitting from the processor to the Bluetooth module serially. Different data shown below is sent to remote system via Bluetooth like the distance of an object and important messages.

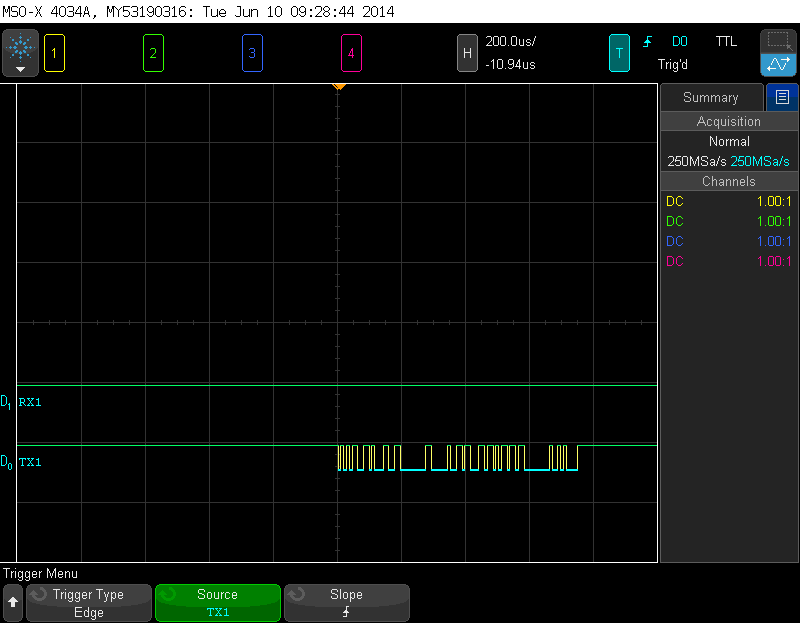


Figure 22. USART sending 51 cm

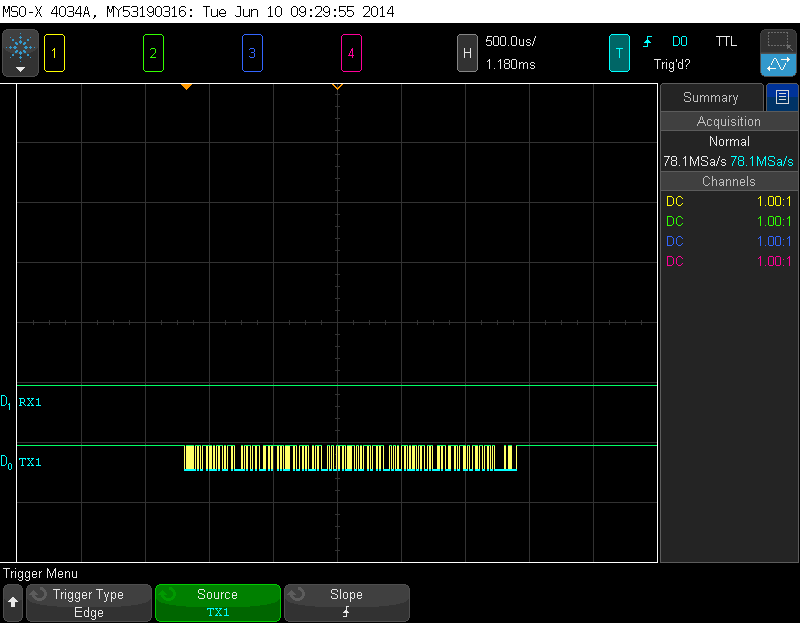


Figure 23. USART sending a string

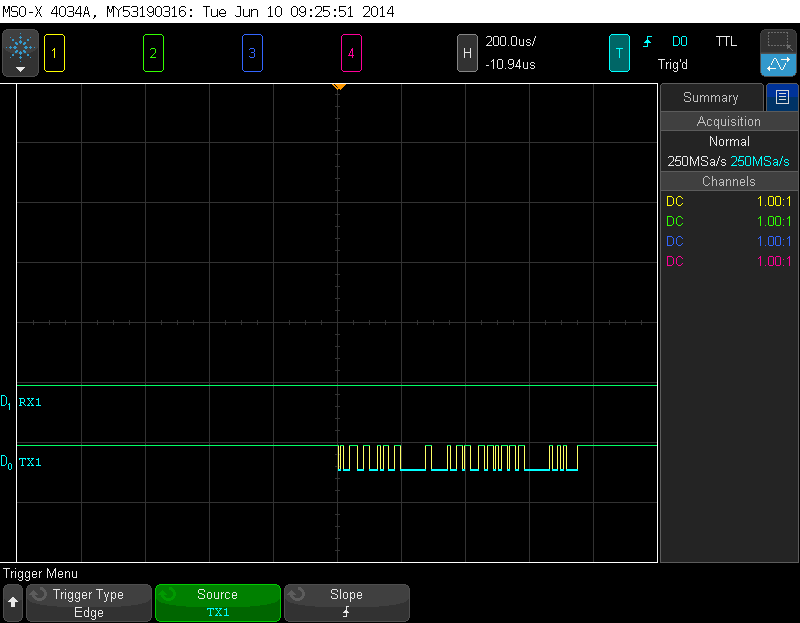


Figure 24. USART sending 94 cm.

# Error Analysis

Most of the errors in this project was solved through debugging. The major error was the connecting the components on a prototype board. The layout and soldering was a little difficult. After each component was mounted on the board, there would be testing to make sure the module worked. There were couple loose connections that cause the modules to not work correctly. An example would be the ultrasonic sensors not responding due to the trigger pin being loose. Another major problem was using the vibration motor. The use of these motors caused inductive interference to the system. More details are discussed in the next section.

# Analysis of Why Project May Not Have Worked

One main problem that cause the project to not have worked was the interference with the vibration motors. When 4 vibration motors were connected, the PIC for the head and hand mounted modules did not respond. After researching this issue, the vibration motor causes inductive noise that interferes with the PIC causing it to not respond. This issue has to be solved by implementing an isolation circuit that will protect the PIC microcontrollers from the inductive noise.

# Summary

The ultrasonic pathfinder system was designed to assist individuals with impaired vision navigate the environment without using a walking cane. The system consisted of a head and hand mounted module. The hand module was designed to scan freely while the head module was designed to scan for objects in front, left and right of the user.

The head module consists of a servo motor, ultrasonic sensor, and 3 vibration motors. When an object is 20 cm in front of the user, then the servo motor will turn left to scan for the distance of the object on the left. Then the servo motor will turn right 180 degrees to scan the distance of the object on the right. Then the system will compare the distances and vibrate the right motor if the distance of the object of the right is farther away. The system will vibrate the left motor if the distance of the object of the left is farther away. The acknowledge button on the hand module has to be pressed to stop the vibration of the motor.

The hand module consist of an ultrasonic sensor, vibration motor, and two buttons. The sensor will measure the distance of the object that the sensor is facing and the vibration motor will vibrate with varying intensity depending on the distance of the object. The acknowledge button is used to stop the right or left motor on the head module from vibrating. The panic button is used to request for help when the user is an abnormal condition.

Overall the project was a success except the isolation circuit for the vibration motors were not implemented. For future development, an analog circuit to drive the motor would help the system work more efficiently.

# Conclusion

This report summarizes the process of designing and implementing an ultrasonic pathfinder. All of the parts of this project were successfully completed. The use of USART and I2C was crucial to the success of the project. Having the ability to transmit data across hardware modules is important in real world applications. Then working with ultrasonic sensors to determine distance was important it allows the user to understand his or her surroundings better. The PIC18F25K22 was also crucial to the system because it controlled the system and determined which operation to do. With the help of MSO, the system was the easier to debug and determine what data was being inputted. By controlling the duty cycle of the PWM. Then using this signal, a voltage was able to be generated to drive a vibration motor. By controlling the duty cycle of the PWM, the intensity of the vibration was able to be controlled. The motors would vibrate with higher intensity if the object was closer and less intense if the object was farther. Overall, this project is a major stepping stone to improve biomedical devices.

# ABET

This device holds to the ABET realistic constraints. Economically this device holds to standards with its optimized costs with no materials or functions unused, besides the extra functions the PIC microcontroller has. Environmental because ultrasonic waves are harmless to the environment. Social because it does not affect the society in a negative way. Ethical because Bluetooth is a two way connection to insure privacy. Health because device causes no harm to the user with controlled PWM output to control vibrating motors to a comfortable level.

# Appendix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bills of Material** | | | | |
| **Item** | **Quanitity** | **Price** | **Shipping** | **Item Total Price** |
| PIC18F25K22 | 2 | $6.84 | $0.00 | $13.68 |
| Breadboard | 1 | $26.00 | $0.00 | $26.00 |
| Servo motor | 1 | $5.00 | $0.00 | $5.00 |
| 20MHz Crystal Oscillator | 2 | $2.40 | $0.00 | $4.80 |
| Bluetooth module | 1 | $40.00 | $0.00 | $40.00 |
| Jumper wire 75 pcs | 1 | $5.09 | $0.00 | $5.09 |
| Vibration motor | 4 | $1.95 | $10.00 | $17.80 |
| Ultrasonic distance sensor | 2 | $14.98 | $0.00 | $29.96 |
| Push button | 2 | $0.49 | $0.00 | $0.98 |
| 9 volt battery | 1 | $3.50 | $0.00 | $3.50 |
| Total Cost |  |  |  | $147.00 |

Figure 25. Bill of materials



Figure 26. Functional Decomposition



Figure 27. Schedule

## Head Module

### Setup.h

#ifndef SETUP\_H

#define SETUP\_H

void init(void);

#endif /\* SETUP\_H \*/

### ISR.h

#ifndef ISR\_H

#define ISR\_H

extern unsigned int edge;

extern unsigned int timeRise;

extern unsigned int timeFall;

extern unsigned int duration;

extern unsigned int newDistance;

extern char frombluetooth;

extern unsigned int newBluetoothCmd;

extern unsigned int pulse\_max;

extern unsigned int pulse\_top;

extern unsigned int top\_value;

extern unsigned int servoEdge;

extern unsigned int servoDirection;

extern unsigned int timer0Count;

extern char userInput;

extern unsigned int newUserInput;

#endif /\* ISR\_H \*/

### Wayfinder.h

#ifndef WAYFINDER\_H

#define WAYFINDER\_H

#endif /\* WAYFINDER\_H \*/

### ISR.c

#include <p18f25k22.h>

#include <timers.h>

#include <usart.h>

#include "ISR.h"

#define SERVO\_LEFT\_ON\_LO\_COMP 0b10111000 // 2.4 ms

#define SERVO\_LEFT\_ON\_HI\_COMP 0b00001011

#define SERVO\_LEFT\_OFF\_LO\_COMP 0b11110000 // 17.6 ms

#define SERVO\_LEFT\_OFF\_HI\_COMP 0b01010101

#define SERVO\_RIGHT\_ON\_LO\_COMP 0b11101110 // 0.6 ms

#define SERVO\_RIGHT\_ON\_HI\_COMP 0b00000010

#define SERVO\_RIGHT\_OFF\_LO\_COMP 0b10111010 // 19.4 ms

#define SERVO\_RIGHT\_OFF\_HI\_COMP 0b01011110

#define SERVO\_MID\_ON\_LO\_COMP 0b11010110 // 1.4 ms

#define SERVO\_MID\_ON\_HI\_COMP 0b00000110

#define SERVO\_MID\_OFF\_LO\_COMP 0b11010010 // 18.6 ms

#define SERVO\_MID\_OFF\_HI\_COMP 0b01011010

#define RISING\_EDGE 0

#define FALLING\_EDGE 1

#define POINT\_FORWARD 0

#define POINT\_LEFT 1

#define POINT\_RIGHT 2

char addr;

// FUNCTION PROTOTYPES

void intHandler( void );

//void lowPriorityINT( void );

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INTERRUPT HANDLERS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma code isr = 0x08

void isr()

{

\_asm GOTO intHandler \_endasm // Branch to interrupt function

}

#pragma code

#pragma interrupt intHandler

void intHandler()

{

// LEFT SENSOR INTERRUPT SERVICE

if (PIR1bits.CCP1IF == 1) // Capture flag set event

{

if (edge == RISING\_EDGE) // If rising edge

{

timeRise = 0;

WriteTimer1(0);

CCP1CONbits.CCP1M0 = 0; // CCP1M = 0100 -> switch to falling edge capture

CCP1CONbits.CCP1M1 = 0;

CCP1CONbits.CCP1M2 = 1;

CCP1CONbits.CCP1M3 = 0;

edge = FALLING\_EDGE;

}

else if (edge == FALLING\_EDGE)

{

timeFall = ReadTimer1();

duration = timeFall - timeRise;

newDistance = 1;

CCP1CONbits.CCP1M0 = 1; // CCP1M = 0101 -> switch to rising edge capture

CCP1CONbits.CCP1M1 = 0;

CCP1CONbits.CCP1M2 = 1;

CCP1CONbits.CCP1M3 = 0;

edge = RISING\_EDGE;

}

PIR1bits.CCP1IF = 0; // Clear Interrupt flag

}

// SERVO CONTROL

if ( PIR2bits.CCP2IF )

{

if ( servoEdge == RISING\_EDGE )

{

LATAbits.LATA0 = 1;

if ( servoDirection == POINT\_FORWARD )

{

CCPR2H = SERVO\_MID\_ON\_HI\_COMP;

CCPR2L = SERVO\_MID\_ON\_LO\_COMP;

}

else if ( servoDirection == POINT\_LEFT )

{

CCPR2H = SERVO\_LEFT\_ON\_HI\_COMP;

CCPR2L = SERVO\_LEFT\_ON\_LO\_COMP;

}

else if ( servoDirection == POINT\_RIGHT )

{

CCPR2H = SERVO\_RIGHT\_ON\_HI\_COMP;

CCPR2L = SERVO\_RIGHT\_ON\_LO\_COMP;

}

servoEdge = FALLING\_EDGE;

WriteTimer3(0);

}

else if (servoEdge = FALLING\_EDGE)

{

LATAbits.LATA0 = 0;

if ( servoDirection == POINT\_FORWARD )

{

CCPR2H = SERVO\_MID\_OFF\_HI\_COMP;

CCPR2L = SERVO\_MID\_OFF\_LO\_COMP;

}

else if ( servoDirection == POINT\_LEFT )

{

CCPR2H = SERVO\_LEFT\_OFF\_HI\_COMP;

CCPR2L = SERVO\_LEFT\_OFF\_LO\_COMP;

}

else if ( servoDirection == POINT\_RIGHT )

{

CCPR2H = SERVO\_RIGHT\_OFF\_HI\_COMP;

CCPR2L = SERVO\_RIGHT\_OFF\_LO\_COMP;

}

servoEdge = RISING\_EDGE;

WriteTimer3(0);

}

PIR2bits.CCP2IF = 0;

}

// BLUETOOTH COMMS

else if ( PIR1bits.RC1IF )

{

frombluetooth = Read1USART();

Write1USART(frombluetooth);

newBluetoothCmd = 1;

PIR1bits.RC1IF = 0;

}

// TIMER0

else if ( INTCONbits.TMR0IF )

{

timer0Count++;

INTCONbits.TMR0IF = 0;

}

// I2C

else if (PIR1bits.SSP1IF) // SSP interrupt

{

if (SSP1STATbits.BF)

{

if (!SSP1STATbits.R\_NOT\_W) // Slave reading

{

if (SSP1STATbits.D\_A == 0) // address not data

{

addr = SSP1BUF; // dummy read to clear BF flag

//SSP1STATbits.BF = 0; // explicitly clear the buffer full flag

//PIR1bits.SSP1IF = 0; // clear interrupt flag

}

else if (SSP1STATbits.D\_A == 1) // data

{

userInput = SSP1BUF;

newUserInput = 1;

//newSpeed = 1;

//SSP1STATbits.BF = 0;

//PIR1bits.SSP1IF = 0;

}

}

else if (SSP1STATbits.R\_NOT\_W) // slave writing

{

if (SSP1STATbits.D\_A == 0) // addr

{

addr = SSP1BUF; // dummy read to clear BF

//SSP1STATbits.BF = 0; // explicitly clear the buffer full flag

SSP1BUF = userInput;

// WriteI2C1(speed);

//PIR1bits.SSP1IF = 0; // clear interrupt flag

//SSP1CON1bits.CKP = 1; // release CLK

}

else if (SSP1STATbits.D\_A == 1) // data

{

SSP1BUF = userInput;

//SSP1STATbits.BF = 0; // explicitly clear the buffer full flag

//PIR1bits.SSP1IF = 0; // clear interrupt flag

//SSP1CON1bits.CKP = 1; // release CLK

}

}

}

SSP1CON1bits.CKP = 1; // release CLK

PIR1bits.SSP1IF = 0;

}

}

#pragma code /\* return to the default code section \*/

/\*

#pragma code low\_isr = 0x18

void low\_isr()

{

\_asm GOTO lowPriorityINT \_endasm // Branch to interrupt function

}

#pragma code

#pragma interrupt lowPriorityINT

void lowPriorityINT()

{

// Timer interrupt (for servo control)

if (INTCONbits.TMR0IF)

{

pulse\_max++;

pulse\_top++;

if (pulse\_max >= MAX\_VALUE)

{

pulse\_max = 0;

pulse\_top = 0;

LATAbits.LATA0 = 0;

}

if (pulse\_top == top\_value)

{

LATAbits.LATA0 = 1;

}

WriteTimer0(240);

INTCONbits.TMR0IF = 0;

}

}

#pragma code

\*/

### Setup.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <p18f25k22.h>

#include <delays.h>

#include <timers.h>

#include <i2c.h>

#include <usart.h>

unsigned int spbrg = 10; // spbrg = 10 -> brgh = 1 -> baudrate = 113636

// FUNCTION PROTOTYPES

void setupGlobalInterrupts( void );

void setupTimer1( void );

void setupTimer3( void );

void setupTimer0( void );

void setupTimer0\_INT( void );

void setupCompare( void );

void setupCompare\_INT( void );

void setupServo( void );

void setupTransducer( void );

void setupCapture( void );

void setupCapture\_INT( void );

void setupPWM( void );

void setupLeftPWM( void );

void setupRightPWM( void );

void setupBluetooth( void );

void setupBluetooth\_INT( void );

void setupI2C\_slave( void );

void setupI2C\_INT\_slave( void );

/\*-----------------------------------------------------------

INIT ROUTINE

-----------------------------------------------------------\*/

void init()

{

setupGlobalInterrupts();

setupTimer1();

setupTransducer();

setupCapture(); // CCP module setup

setupCapture\_INT();

setupPWM();

setupLeftPWM();

setupRightPWM();

setupTimer0();

setupTimer0\_INT();

setupTimer3();

setupCompare();

setupCompare\_INT();

setupServo();

setupBluetooth();

setupBluetooth\_INT();

setupI2C\_slave();

setupI2C\_INT\_slave();

PIR1 = 0x00;

PIR2 = 0x00;

Delay100TCYx(1); // delay 100 cycles

}

void setupGlobalInterrupts()

{

RCONbits.IPEN = 0; // Priority Enable OFF

INTCONbits.GIE = 1; // Global Interrupt ON

INTCONbits.PEIE = 1; // Peripheral Interrupt Enable

INTCON2bits.NOT\_RBPU = 1; // PORTB pull up OFF

}

void setupTimer0()

{

T0CONbits.T08BIT = 0; // 16-bit timer

T0CONbits.T0CS = 0; // instruction cycle clock

T0CONbits.T0SE = 0; // low to high edge

T0CONbits.PSA = 0; // use prescaler

T0CONbits.T0PS0 = 0; // prescale = 110 = 1 : 128. Period per tick = 12.8 us

T0CONbits.T0PS1 = 1; // overflow period = 1.68 s. 1 minute = 35.7 overflows

T0CONbits.T0PS2 = 1;

}

void setupTimer0\_INT()

{

INTCONbits.TMR0IE = 1;

}

// TMR1

void setupTimer1()

{

T1CONbits.TMR1CS0 = 0; // TMR1CS = 01 -> Use instruction clock 20MHz/4, FOSC/4

T1CONbits.TMR1CS1 = 0;

T1CONbits.T1CKPS0 = 0; // T1CKPS = 10 -> 1:4 prescale

T1CONbits.T1CKPS1 = 1;

T1CONbits.T1SOSCEN = 0; // Secondary Oscillator Disabled

T1CONbits.T1RD16 = 0; // Enable register read/write in one 16-bit operations. Registers TMR1L and TMR1H

T1CONbits.TMR1ON = 1; // Enable TMR1/3/5

WriteTimer1(0); // Clear timer1 values

}

void setupBluetooth()

{

unsigned char USARTConfig = USART\_TX\_INT\_OFF & USART\_RX\_INT\_ON & USART\_ASYNCH\_MODE

& USART\_EIGHT\_BIT & USART\_BRGH\_HIGH & USART\_CONT\_RX;

Close1USART(); // close if USART was open earlier

Open1USART( USARTConfig, spbrg );

// TXSTA bit CONFIG

TXSTA1bits.BRGH = 1; // High Baud Rate

TXSTA1bits.TXEN = 1; // Transmit Enable

TXSTA1bits.SYNC = 0; // Asynch

TXSTA1bits.TX9 = 0; // 9-bit transmit off

// RCSTA bit CONFIG

RCSTA1bits.SPEN = 1; // Serial Port Enable bit

RCSTA1bits.CREN = 1; // Continuous Receive Enable bit

RCSTA1bits.RX9 = 0; // 9-bit receive off

// PIC automatically decides which one is input and which one is output

ANSELCbits.ANSC6 = 0; // Set C Digital

ANSELCbits.ANSC7 = 0; // Set C Digital

TRISCbits.RC6 = 1; // TX pin set as output

TRISCbits.RC7 = 1; // RX pin set as input

Open1USART(USARTConfig, spbrg);

}

void setupBluetooth\_INT() {

PIE1bits.RC1IE = 1; // RX1 Interrupt Enable Bit

//IPR1bits.RC1IP = 1; // RX1 Interrupt High priority

}

/\*-----------------------------------------------------------

SERVO CONTROL

-----------------------------------------------------------\*/

//RA0 controls the servo

void setupServo()

{

TRISAbits.RA0 = 0; // RA2 Trigger (Output)

ANSELAbits.ANSA0 = 0; // Digital select

LATAbits.LATA0 = 0;

}

void setupTimer3()

{

T3CONbits.TMR3CS0 = 0; // TMR1CS = 01 -> Use instruction clock 20MHz/4, FOSC/4

T3CONbits.TMR3CS1 = 0;

T3CONbits.T3CKPS0 = 0; // T1CKPS = 10 -> 1:4 prescale

T3CONbits.T3CKPS1 = 1;

T3CONbits.T3SOSCEN = 0; // Secondary Oscillator Disabled

T3CONbits.T3RD16 = 0; // Enable register read/write in one 16-bit operations. Registers TMR1L and TMR1H

T3CONbits.TMR3ON = 1; // Enable TMR1/3/5

WriteTimer3(0); // Clear timer3 values

}

// compare uses CCP2

void setupCompare()

{

CCP2CONbits.CCP2M0 = 0; // generate software interrupt on compare match

CCP2CONbits.CCP2M1 = 1;

CCP2CONbits.CCP2M2 = 0;

CCP2CONbits.CCP2M3 = 1;

CCPTMRS0bits.C2TSEL0 = 1; // Set CCP2 to use TMR3

CCPTMRS0bits.C2TSEL1 = 0;

CCPR2H = 0b01100001;

CCPR2L = 0b10101000; // Values to compare the timer to, initially 0110000110101000 for 25000 ticks = 20ms

}

void setupCompare\_INT()

{

PIE2bits.CCP2IE = 1; // Enable CCP2 Interrupt

//IPR2bits.CCP2IP = 1; // compare high interrupt

}

/\*-----------------------------------------------------------

SENSOR AND FEEDBACK

-----------------------------------------------------------\*/

void setupTransducer()

{

TRISAbits.RA2 = 0; // RA2 Trigger (Output)

ANSELAbits.ANSA2 = 0; // Digital select

LATAbits.LATA2 = 0;

}

// CCP1

void setupCapture()

{

TRISCbits.TRISC2 = 1; // CCP1 to input capture

ANSELCbits.ANSC2 = 0; // digital select

CCP1CONbits.CCP1M0 = 1; // CCP1M = 0101 => Capture mode: every rising edge 0101. Reset to falling edge after interrupt

CCP1CONbits.CCP1M1 = 0;

CCP1CONbits.CCP1M2 = 1;

CCP1CONbits.CCP1M3 = 0;

CCPTMRS0bits.C1TSEL0 = 0; // C1TSEL = 00 => capture mode uses Timer1

CCPTMRS0bits.C1TSEL1 = 0;

}

void setupCapture\_INT()

{

PIE1bits.CCP1IE = 1; // Enable CCP1 Interrupt

//IPR1bits.CCP1IP = 1; // Transducer capture high interrupt

}

//Setups PWM CCP3

void setupPWM()

{

// 20MHz

ANSELBbits.ANSB5 = 0; // PWM ANSEL

TRISBbits.RB5 = 1; // TRIS bit to start the setup, prevent spurious outputs during setup

//PSTR1CONbits.STR1A = 1; // Steering enable on port A, probably not needed

//CCPTMRS0 = 0x00; // Set CCP1 to use Timer2

CCPTMRS0bits.C3TSEL0 = 0; // CCP3 uses timer2

CCPTMRS0bits.C3TSEL1 = 0;

PR2 = 0b11111001; // Set Timer2; Period register; PR2 = 249

CCP3CON = 0b00101100; // Set to PWM mode; 5:4 are 2 LSB bits of duty cycle

CCPR3L = 0b00111110; // Set 8 MSB of duty cycle; Set duty cycle to 0

T2CON = 0b00000100; // Set Postscaler and prescaler to 1 and turn on PWM, TMR2

CCP3CONbits.DC3B1 = 1;

CCP3CONbits.DC3B0 = 0;

TRISBbits.RB5 = 0; // Set PWM bit for output to enable PWM. Done with setup

}

// CCP4

// Port RB0

void setupLeftPWM()

{

// 20MHz

ANSELBbits.ANSB0 = 0; // PWM ANSEL

TRISBbits.RB0 = 1; // TRIS bit to start the setup, prevent spurious outputs during setup

//PSTR1CONbits.STR1A = 1; // Steering enable on port A, probably not needed

//CCPTMRS0 = 0x00; // Set CCP1 to use Timer2

CCPTMRS1bits.C4TSEL0 = 0; // CCP2 uses timer2

CCPTMRS1bits.C4TSEL1 = 0;

PR2 = 0b11111001; // Set Timer2; Period register; PR2 = 249

CCP4CON = 0b00101100; // Set to PWM mode; 5:4 are 2 LSB bits of duty cycle

CCPR4L = 0b00111110; // Set 8 MSB of duty cycle; Set duty cycle to 0

T2CON = 0b00000100; // Set Postscaler and prescaler to 1 and turn on PWM, TMR2

CCP4CONbits.DC4B1 = 1;

CCP4CONbits.DC4B0 = 0;

TRISBbits.RB0 = 0; // Set PWM bit for output to enable PWM. Done with setup

}

// CCP5

void setupRightPWM()

{

// 20MHz

//ANSELAbits.ANSA4 = 0; // PWM ANSEL

TRISAbits.RA4 = 1; // TRIS bit to start the setup, prevent spurious outputs during setup

//PSTR1CONbits.STR1A = 1; // Steering enable on port A, probably not needed

//CCPTMRS0 = 0x00; // Set CCP1 to use Timer2

CCPTMRS1bits.C5TSEL0 = 0; // CCP2 uses timer2

CCPTMRS1bits.C5TSEL1 = 0;

PR2 = 0b11111001; // Set Timer2; Period register; PR2 = 249

CCP5CON = 0b00101100; // Set to PWM mode; 5:4 are 2 LSB bits of duty cycle

CCPR5L = 0b00111110; // Set 8 MSB of duty cycle; Set duty cycle to 0

T2CON = 0b00000100; // Set Postscaler and prescaler to 1 and turn on PWM, TMR2

CCP5CONbits.DC5B1 = 1;

CCP5CONbits.DC5B0 = 0;

TRISAbits.RA4 = 0; // Set PWM bit for output to enable PWM. Done with setup

}

/\*-----------------------------------------------------------

I2C

-----------------------------------------------------------\*/

//I2C setup for the slave

void setupI2C\_slave()

{

CloseI2C1();

//TRISCbits.TRISC3 = 1;

//TRISCbits.TRISC4 = 1;

ANSELCbits.ANSC3 = 0; // Set C Digital

ANSELCbits.ANSC4 = 0; // Set C Digital

SSP1CON2bits.SEN = 0; // no clock stretching

OpenI2C1(SLAVE\_7, SLEW\_OFF);

SSP1ADD = 0xB0;

}

//I2C interrupt setup for the slave

void setupI2C\_INT\_slave()

{

//RCONbits.IPEN = 1; // Enable LOW/HIGH Interrupt Priority

//INTCONbits.GIEH = 1; // Global Interrupt Enable HIGH

//INTCONbits.GIEL = 1; // Global Interrupt Enable LOW

//INTCONbits.PEIE = 1; // Peripheral Interrupt Enable

PIE1bits.SSP1IE = 1; // SSP1 Interrupt Enable Bit

//IPR1bits.SSP1IP = 1; // SSP1 Interrupt Priority HIGH

}

### Wayfinder.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include <p18f25k22.h>

#include <delays.h>

#include <usart.h>

#include <i2c.h>

#include <pwm.h>

#include <PWM.h>

#include "setup.h"

#define US\_PER\_TICK .8

#define DISPLAY\_THRESHOLD 200

#define WARNING\_THRESHOLD 20

#define RISING\_EDGE 0

#define FALLING\_EDGE 1

#define POINT\_FORWARD 0

#define POINT\_LEFT 1

#define POINT\_RIGHT 2

#define START 1

#define STOP 0

/\*-----------------------------------------------------------

CONFIG BITS

-----------------------------------------------------------\*/

#pragma config WDTEN = OFF // Watch Dog Timer Enable OFF

#pragma config LVP = OFF // Low Voltage Programming (3.3V) OFF

#pragma config PRICLKEN = OFF // Primary Clock OFF

#pragma config IESO = ON // Switch to External Clock

#pragma config FOSC = ECHP // High speed external OSC

#pragma config XINST = OFF // Extended instr set OFF

/\*-----------------------------------------------------------

GLOBAL VARS

-----------------------------------------------------------\*/

// Left sensor utilities

unsigned int edge = RISING\_EDGE; // 0 is rising edge, 1 is falling edge

unsigned int timeRise = 0;

unsigned int timeFall = 0;

unsigned int duration = 0;

unsigned int newDistance = 0;

unsigned int intensity = 0;

unsigned int distanceFront\_cm = 0;

unsigned int distanceLeft\_cm = 0;

unsigned int distanceRight\_cm = 0;

unsigned int scanning = 0;

char dataString[6]; // to write to remote monitor

unsigned int newUserInput = 0;

char userInput = 'A';

unsigned int timer0Count = 0;

unsigned int minute = 0;

char minuteString[2];

// Servo control

unsigned int servoEdge = RISING\_EDGE; // 0 is rising edge

unsigned int servoDirection = POINT\_FORWARD; // 0 = mid, 1 = left, 2 = right

char frombluetooth = 'g';

unsigned int newBluetoothCmd = 0;

char cm[] = " cm\0";

char danger0[] = "Warning!!! Obstacle \0";

char danger1[] = " ahead\0";

char alert[] = "The user is not responding...\nSend help!\0";

char good[] = "Plenty of space...\0";

char help[] = "Help on the way!\0";

//char panic[] = "User has pressed the panic button... Better check on them!\0";

char scanningLeft[] = "Scanning left...\0";

char scanningRight[] = "Scanning right...\0";

char promptLeft[] = "User should turn left...\0";

char promptRight[] = "User should turn right...\0";

/\*-----------------------------------------------------------

FUNCTION PROTOTYPES

-----------------------------------------------------------\*/

void feedback( int input );

int calculateDistance( int ticks );

void trigger( void );

void leftFeedback( int startStop );

void rightFeedback( int startStop );

/\*-----------------------------------------------------------

MAIN ROUTINE

-----------------------------------------------------------\*/

void main()

{

init();

while(1)

{

if ( newBluetoothCmd )

{

if ( frombluetooth == 'k' )

{

Write1USART( '\n' );

puts1USART( help );

}

newBluetoothCmd = 0;

}

if ( timer0Count >= 36 )

{

puts1USART( alert );

Write1USART( '\n' );

timer0Count = 27; // continuously alert every 15 seconds

}

if ( newUserInput == 1)

{

if ( userInput == 'P')

{

puts1USART( alert );

Write1USART( '\n' );

}

newUserInput = 0;

}

if ( !newDistance && userInput == 'A')

{

timer0Count = 0; // reset alert timer

leftFeedback( 0 );

rightFeedback( 0 );

trigger();

while ( !newDistance );

}

if ( newDistance )

{

if ( servoDirection == POINT\_FORWARD ) // Normal operation, scanning distance ahead

{

distanceFront\_cm = calculateDistance( duration );

itoa( distanceFront\_cm, dataString );

if ( distanceFront\_cm <= DISPLAY\_THRESHOLD ) // only display the distance on the remote monitor if less than threshold

{

puts1USART( dataString ); // display distance measurements on remote monitor

puts1USART( cm );

Write1USART( '\n' );

}

if ( distanceFront\_cm <= WARNING\_THRESHOLD ) // if the distance in front is less than

{

puts1USART( danger0 ); // display warning info onto the remote monitor

puts1USART( dataString );

puts1USART( cm );

puts1USART( danger1 );

Write1USART( '\n' );

puts1USART( scanningLeft );

Write1USART( '\n' );

servoDirection = POINT\_LEFT; // change the sensor direction to point left

scanning = 1;

feedback( 100 );

newDistance = 0;

Delay10KTCYx(0); // delay to complete change of direction

}

newDistance = 0;

// MAY NEED TO RECALCULATE SINCE MOTOR RUNS @ 2-5 V

if ( distanceFront\_cm >= 100 && scanning == 0 )

{

feedback( 100 );

}

else if ( distanceFront\_cm < 100 && scanning == 0)

{

feedback( distanceFront\_cm );

}

}

else if ( servoDirection == POINT\_LEFT ) // Scanning: scanning left and right for better path

{

distanceLeft\_cm = calculateDistance( duration );

itoa( distanceLeft\_cm, dataString );

if ( distanceLeft\_cm > 300 )

{

puts1USART( good );

}

else

{

puts1USART( dataString );

puts1USART( cm );

}

Write1USART( '\n' );

puts1USART( scanningRight );

Write1USART( '\n' );

servoDirection = POINT\_RIGHT; // change the sensor direction to point right

newDistance = 0;

Delay10KTCYx(0); // delay to complete change of direction

Delay10KTCYx(120);

}

else if ( servoDirection == POINT\_RIGHT ) // Scanning: scanning right

{

distanceRight\_cm = calculateDistance( duration );

itoa( distanceRight\_cm, dataString );

if ( distanceRight\_cm > 300 )

{

puts1USART( good );

}

else

{

puts1USART( dataString );

puts1USART( cm );

}

Write1USART( '\n' );

servoDirection = POINT\_FORWARD; // return the transducer to original pos

newDistance = 0;

Delay10KTCYx(0); // delay to complete change of direction

if ( distanceLeft\_cm > distanceRight\_cm )

{

leftFeedback( 1 );

puts1USART( promptLeft );

Write1USART( '\n' );

}

else

{

rightFeedback( 1 );

puts1USART( promptRight );

Write1USART( '\n' );

}

scanning = 0;

userInput = 'S'; // stop the functions until the user acknowledge

}

}

}

}

// duty cycle = input / 200

// or value / 1000

void feedback( int input )

{

// 200 cm = lowest intensity

// 0 cm = highest intensity

int value = ( 100 - input ) \* 10;

//int value = input \* 17;

CCP3CONbits.DC3B0 = value & 0x01; //set low bit

CCP3CONbits.DC3B1 = (value >> 0x01) & 0x01; //set second lowest

CCPR3L = (value >> 2); //set highest eight

}

void leftFeedback( int startStop )

{

int value;

if ( startStop == START)

{

value = 500;

}

else

{

value = 0;

}

CCP4CONbits.DC4B0 = value & 0x01; //set low bit

CCP4CONbits.DC4B1 = (value >> 0x01) & 0x01; //set second lowest

CCPR4L = (value >> 2); //set highest eight

}

void rightFeedback( int startStop )

{

int value;

if ( startStop == START)

{

value = 500;

}

else

{

value = 0;

}

CCP5CONbits.DC5B0 = value & 0x01; //set low bit

CCP5CONbits.DC5B1 = (value >> 0x01) & 0x01; //set second lowest

CCPR5L = (value >> 2); //set highest eight

}

// 1 tick = .8uS

// Time (us) = # of ticks \* .8us

// Distance in inches = time / 148

// Distance in centimeters = time / 58

int calculateDistance( int ticks )

{

float time\_uS = 0.0f;

int distance\_cm = 0;

time\_uS = ticks \* US\_PER\_TICK;

distance\_cm = (int) (ceil(time\_uS / 58));

return distance\_cm;

}

void trigger()

{

LATAbits.LATA2 = 1;

Delay10TCYx(7);

LATAbits.LATA2 = 0;

Delay1KTCYx(0);

Delay1KTCYx(0);

}

## Hand Module

### Setup.h

/\*

\* File: setup.h

\* Author: vienly

\*

\* Created on May 29, 2014, 4:50 PM

\*/

#ifndef SETUP\_H

#define SETUP\_H

void init( void );

#endif /\* SETUP\_H \*/

### ISR.h

#ifndef ISR\_H

#define ISR\_H

extern unsigned int edge;

extern unsigned int timeRise;

extern unsigned int timeFall;

extern unsigned int duration;

extern unsigned int newDistance;

extern unsigned int newUserInput;

extern char userInput;

#endif /\* ISR\_H \*/

### Setup.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <p18f25k22.h>

#include <delays.h>

#include <timers.h>

#include <i2c.h>

#define ADDRESS 0x31

// FUNCTION PROTOTYPES

void setupGlobalInterrupts( void );

void setupTimer1( void );

void setupTranducer( void );

void setupCapture( void );

void setupCapture\_INT( void );

void setupPWM( void );

void setupAckButton( void );

void setupPanicButton( void );

void setupI2C\_master( void );

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INIT ROUTINE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void init()

{

setupGlobalInterrupts();

setupTranducer();

setupTimer1();

setupCapture(); // CCP module setup

setupCapture\_INT();

setupPWM();

setupAckButton();

setupPanicButton();

setupI2C\_master();

LATAbits.LATA2 = 0; // Initial trigger state = 0

PIR1 = 0x00;

Delay100TCYx(1); // delay 100 cycles

}

void setupGlobalInterrupts()

{

RCONbits.IPEN = 0; // Priority Enable OFF

INTCONbits.GIE = 1; // Global Interrupt ON

INTCONbits.PEIE = 1; // Peripheral Interrupt Enable

INTCON2bits.NOT\_RBPU = 1; // PORTB pull up OFF

}

// TMR1

void setupTimer1()

{

T1CONbits.TMR1CS0 = 0; // TMR1CS = 01 -> Use instruction clock 20MHz/4, FOSC/4

T1CONbits.TMR1CS1 = 0;

T1CONbits.T1CKPS0 = 0; // T1CKPS = 10 -> 1:4 prescale

T1CONbits.T1CKPS1 = 1;

T1CONbits.T1SOSCEN = 0; // Secondary Oscillator Disabled

T1CONbits.T1RD16 = 0; // Enable register read/write in one 16-bit operations. Registers TMR1L and TMR1H

T1CONbits.TMR1ON = 1; // Enable TMR1/3/5

WriteTimer1(0); // Clear timer1 values

}

/\*-----------------------------------------------------------

SENSOR SETUP

-----------------------------------------------------------\*/

void setupTranducer()

{

TRISAbits.RA2 = 0; // RA2 Trigger (Output)

ANSELAbits.ANSA2 = 0; // Digital select

LATAbits.LATA2 = 0;

}

// CCP1

void setupCapture()

{

TRISCbits.TRISC2 = 1; // CCP1 to input capture

ANSELCbits.ANSC2 = 0; // digital select

CCP1CONbits.CCP1M0 = 1; // CCP1M = 0101 => Capture mode: every rising edge 0101. Reset to falling edge after interrupt

CCP1CONbits.CCP1M1 = 0;

CCP1CONbits.CCP1M2 = 1;

CCP1CONbits.CCP1M3 = 0;

CCPTMRS0bits.C1TSEL0 = 0; // C1TSEL = 00 => capture mode uses Timer1

CCPTMRS0bits.C1TSEL1 = 0;

}

void setupCapture\_INT()

{

PIE1bits.CCP1IE = 1; // Enable CCP1 Interrupt

}

//Setups PWM CCP3

void setupPWM()

{

// 20MHz

ANSELBbits.ANSB5 = 0; // PWM ANSEL

TRISBbits.RB5 = 1; // TRIS bit to start the setup, prevent spurious outputs during setup

CCPTMRS0bits.C3TSEL0 = 0; // Set CCP3 to use TMR2

CCPTMRS0bits.C3TSEL1 = 0;

PR2 = 0b11111001; // Set Timer2; Period register; PR2 = 249

CCP3CON = 0b00101100; // Set to PWM mode; 5:4 are 2 LSB bits of duty cycle

CCPR3L = 0b00111110; // Set 8 MSB of duty cycle; Set duty cycle to 0

T2CON = 0b00000100; // Set Postscaler and prescaler to 1 and turn on PWM, TMR2

CCP3CONbits.DC3B1 = 1;

CCP3CONbits.DC3B0 = 0;

TRISBbits.RB5 = 0; // Set PWM bit for output to enable PWM. Done with setup

}

void setupI2C\_master()

{

CloseI2C1(); // Close i2c if was opened prior

//TRISCbits.TRISC3 = 0;

//TRISCbits.TRISC4 = 0;

ANSELCbits.ANSC3 = 0; // Set C Digital SCL

ANSELCbits.ANSC4 = 0; // Set C Digital SDA

OpenI2C1( MASTER, SLEW\_OFF );

SSP1ADD = ADDRESS; // 49 = SSPADD Baud Register used to calculate I2C clock speed (100Khz)

}

// RB1

// INT1

// flag is INTCON3bits.INT1IF

void setupAckButton()

{

ANSELBbits.ANSB1 = 0;

TRISBbits.TRISB1 = 1; // CCP1 to input capture

INTCON2bits.INTEDG1 = 1; // rising edge

INTCON3bits.INT1IE = 1; // enable INT1

}

// RB2

// INT2

// flag is INTCON3bits.INT2IF

void setupPanicButton()

{

ANSELBbits.ANSB2 = 0;

TRISBbits.TRISB2 = 1;

INTCON2bits.INTEDG2 = 1;

INTCON3bits.INT2IE = 1;

}

### ISR.c

#include <p18f25k22.h>

#include <timers.h>

#include "ISR.h"

// FUNCTION PROTOTYPES

void ISR1( void );

void distance\_ISR( void );

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INTERRUPT HANDLERS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*-----------------------------------------------------------

LEFT SENSOR ROUTINE

-----------------------------------------------------------\*/

#pragma code ISR1 = 0x08

void ISR1()

{

\_asm GOTO distance\_ISR \_endasm // Branch to interrupt function

}

#pragma code

#pragma interrupt distance\_ISR

void distance\_ISR()

{

// LEFT SENSOR INTERRUPT SERVICE

if ( PIR1bits.CCP1IF == 1 ) // Capture flag set event

{

if (edge == 0) // If rising edge

{

timeRise = 0;

WriteTimer1(0);

CCP1CONbits.CCP1M0 = 0; // CCP1M = 0100 -> switch to falling edge capture

CCP1CONbits.CCP1M1 = 0;

CCP1CONbits.CCP1M2 = 1;

CCP1CONbits.CCP1M3 = 0;

edge = 1;

}

else if ( edge == 1 )

{

timeFall = ReadTimer1();

duration = timeFall - timeRise;

newDistance = 1;

CCP1CONbits.CCP1M0 = 1; // CCP1M = 0101 -> switch to rising edge capture

CCP1CONbits.CCP1M1 = 0;

CCP1CONbits.CCP1M2 = 1;

CCP1CONbits.CCP1M3 = 0;

edge = 0;

}

PIR1bits.CCP1IF = 0; // Clear Interrupt flag

}

// ACK BUTTON

else if ( INTCON3bits.INT1IF )

{

userInput = 'A';

newUserInput = 1;

INTCON3bits.INT1IF = 0;

}

// PANIC BUTTON

else if ( INTCON3bits.INT2IF )

{

userInput = 'P';

newUserInput = 1;

INTCON3bits.INT2IF = 0;

}

}

#pragma code /\* return to the default code section \*/

### Hand\_mounted\_unit.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include <p18f25k22.h>

#include <delays.h>

#include <usart.h>

#include <i2c.h>

#include <pwm.h>

#include "setup.h"

#define US\_PER\_TICK .8

// CONFIG BITS

#pragma config WDTEN = OFF // Watch Dog Timer Enable OFF

#pragma config LVP = OFF // Low Voltage Programming (3.3V) OFF

#pragma config PRICLKEN = OFF // Primary Clock OFF

#pragma config IESO = ON // Switch to External Clock

#pragma config FOSC = ECHP // High speed external OSC

#pragma config XINST = OFF // Extended instr set OFF

// GLOBAL VARIABLES

/\*

\* LEFT SENSOR UTILITIES

\*/

unsigned int edge = 0; // 0 is rising edge, 1 is falling edge

unsigned int timeRise = 0;

unsigned int timeFall = 0;

unsigned int duration = 0;

unsigned int newDistance = 0;

unsigned int intensity = 0;

unsigned int distance\_cm = 0;

unsigned int newUserInput = 0;

char userInput = '\0';

// FUNCTION PROTOTYPES

void feedback( int input );

int calculateDistance( int ticks );

void trigger( void );

void sendNewInput( void );

// ISR not yet working \*\*\*\*\*\*\*\*\*\*\*\*0

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MAIN ROUTINE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main() {

init();

while(1)

{

if ( newDistance == 1 )

{

distance\_cm = calculateDistance( duration );

if ( distance\_cm >= 100 )

{

feedback( 100 );

//feedback( 100 );

}

else if ( distance\_cm < 100 )

{

feedback( distance\_cm );

//feedback( 100 );

}

newDistance = 0;

}

else if ( newDistance == 0 )

{

trigger();

}

if ( newUserInput == 1)

{

sendNewInput();

newUserInput = 0;

}

}

}

// duty cycle = input / 200

// or value / 1000

// testing: input from 0 to 60 cm

// 60 = 100% duty

void feedback( int input )

{

// 200 cm = lowest intensity

// 0 cm = highest intensity

int value = ( 100 - input ) \* 10;

//int value = input \* 17;

CCP3CONbits.DC3B0 = value & 0x01; //set low bit

CCP3CONbits.DC3B1 = (value >> 0x01) & 0x01; //set second lowest

CCPR3L = (value >> 2); //set highest eight

}

// 1 tick = .8uS

// Time (us) = # of ticks \* .8us

// Distance in inches = time / 148

// Distance in centimeters = time / 58

int calculateDistance( int ticks )

{

float time\_uS = 0.0f;

int distance\_cm = 0;

time\_uS = ticks \* US\_PER\_TICK;

distance\_cm = (int) (ceil(time\_uS / 58));

return distance\_cm;

}

void trigger()

{

LATAbits.LATA2 = 1;

Delay10TCYx(7);

LATAbits.LATA2 = 0;

Delay1KTCYx(0);

Delay1KTCYx(0);

}

void sendNewInput()

{

IdleI2C1(); // SEN = 1

StartI2C1();

IdleI2C1();

putcI2C1( 0xB0 ); //send address

IdleI2C1();

putcI2C1( userInput ); //send data

IdleI2C1();

StopI2C1();

}