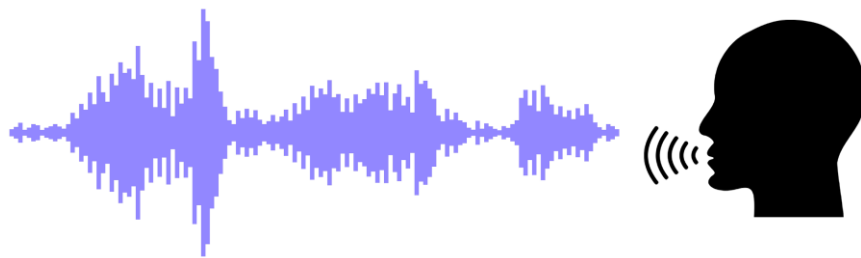




SUNY Oswego  
**Electrical and Computer Engineering Department**



ECE 492  
**Voice Controlled Robot**  
By  
**Brandon Galletta, Kory McTague, Stephanie Valley**

Supervisors  
**Marianne Hromalik & Mario Bkassiny**

*Submitted in Partial Fulfillment of the Requirement for the B.Sc.  
Degree,  
Electrical and Computer Engineering Department,*

**Fall 2019**

### ***Project Abstract***

The proposed project is a voice-controlled robot, where the primary focus of the project is speech recognition. The final design will demonstrate signal processing to analyze speech and then decipher a voice command given by the user. The command is then sent to the robot via wireless communication to perform said command. It is expected to work under specified conditions discussed later in the report. Speech is very distinct and has many characteristics and features that separate words from each other. Using different algorithms and the knowledge from the observations made from experimentation, the voice recognition system will be able to take the characteristics from a sound input and compute if the spoken word is one of the valid commands.

### ***Acknowledgment***

We would like to thank all of the Electrical and Computer Engineering professors for their countless hours of work and support to not just this capstone group, but all students here at SUNY Oswego. Professors Mario Bkassiny and Marianne Hromalik have been the advisors for this capstone project, and the input they have given have helped this project get to where it is. The feedback given has made a huge impact on a successful design. Dennis Quill also has played a large role in the development of our design—from talk of basic circuit design, to going out of his way to discuss his ideas with the advisers himself. The project as a whole would not have come this far without the help from the advisers and the faculty in the ECE department, and this group is very grateful for their continuous efforts.

### ***Student Statement***

Things such as safety, honesty, and ethical design are important factors for engineering design. From ECE 401, students learned the importance of ethics in engineering and used IEEE's Code of Ethics as a guideline into developing the final design of the project. Throughout the design of the project, proper ethical design was discussed and implemented in order to ensure the safety of the students and the public.

## **Table of Contents**

1.0   Problem Statement.....	5
2.0   Problem Formulation.....	5
3.0   Project Specifications.....	5
4.0   Detailed Engineering Analysis and Design Presentation.....	7
4.1   Design Concept.....	7
4.2   Voice Recognition and Algorithm.....	8
4.2.1   Theory and Overview.....	8
4.2.2   Speech Detection and Preprocessing.....	9
4.2.3   Lower Pitch Speech Algorithm.....	10
4.2.4   Higher Pitch Speech Algorithm.....	11
4.3   Wireless Transmission of Data.....	12
4.4   Robot Design and Implementation.....	13
4.5   System Characteristics and Performance.....	14
5.0   Deliverables.....	15
6.0   Cost Analysis & Bill of Materials.....	15
6.1   Estimated Cost.....	15
6.2   Component Selection.....	16
7.0   Hazards and Failure Analysis.....	17
8.0   Learning Outcomes.....	18
9.0   Standards Used in the Design.....	18
10.0   Conclusions.....	19
10.1   Summary and Results.....	19
10.2   Improvements and Future Work .....	20
References.....	21
Appendices.....	21
A   Processing, Power, and I/O.....	21
B   Code and Graphics.....	26
C   XBee Documentation.....	50

## **List of Figures, Tables, and Code Blocks**

### ***Figures:***

Figure 1 – Raspberry Pi 3 B+.....	6
Figure 2 – Jounivo USB Microphone.....	6
Figure 3 – Full System Block Diagram.....	7
Figure 4 – Spectrogram of Brandon saying “Go”.....	8
Figure 5 – Spectrogram of Stephanie saying “Go”.....	9
Figure 6 – Average Magnitude Envelope over an arbitrary waveform.....	9
Figure 7 – Speech Recognition Algorithm tree diagram.....	10
Figure 8 – XBee S2C Module.....	12
Figure 9 – Speech Recognition Accuracy Measurements.....	14
Figure 10 – Raspberry Pi 3 B+ Power Measurements.....	16
Figure 11 – Raspberry Pi 3 B+ GPIO pins.....	22
Figure 12 – Raspberry Pi Data Sheet.....	23
Figure 13 – Raspberry Pi Physical Specifications.....	24
Figure 14 – Tiva C Launchpad I/O.....	25
Figure 15 – Tiva C Physical Connections.....	25
Figure 16 – Fourier Transform (FT) of Brandon saying “Back” 5x.....	47
Figure 17 – FT of Brandon saying “Go” 5x.....	47
Figure 18 – FT of Brandon saying “Right” 5x.....	48
Figure 19 – FT of Brandon saying “Left” 5x.....	48
Figure 20 – Spectrogram of Brandon saying all four commands.....	49
Figure 21 – Spectrogram of Stephanie saying all four commands.....	49
Figure 22 – 3D representation of a Spectrogram.....	50
Figure 23 – Frame formatting for XBee modules.....	50
Figure 24 – XBee I/O pin layout.....	51
Figure 25 – XBee I/O Descriptions.....	51

### ***Tables:***

Table 1 – Cost Analysis.....	16
------------------------------	----

### ***Code Blocks:***

Python Block 1 – Main Speech Recognition Script.....	27
--	----

Python Block 2 – Custom Python functions for main speech recognition algorithm.....	28
Python Block 3 – All custom functions written throughout the semester; some unused.....	29
Python Block 4 – Record audio using the Jounivo USB microphone.....	31
Python Block 5 – Analyzing Fourier Transform results on speech waveforms.....	32
Python Block 6 – Script to record speaker multiple times and plot multiple FTs.....	34
MATLAB Block 1 – Normalize Function.....	35
MATLAB Block 2 – Function to find beginning of speech signal.....	35
MATLAB Block 3 – Function to find end of speech signal.....	35
MATLAB Block 4 – Script to record audio samples for analysis.....	36
MATLAB Block 5 – Calculate STFT and plot Spectrogram of all audio samples.....	37
MATLAB Block 6 – STFT analytical analysis through trial-and-error.....	41
MATLAB Block 7 – Zero Crossing Rate (ZCR) function.....	42
MATLAB Block 8 – Using ZCR on speech and non-speech.....	43
MATLAB Block 9 – Cepstrum and Quefrequency testing.....	44
MATLAB Block 10 – Continuous Wavelet Transform testing.....	44
Embedded C Block 1 – Tiva-C Robot Movement Code.....	45

## **1 | Problem Statement**

Currently, smart technology is growing in popularity all around the world. Devices such as smartphones can be connected to things like refrigerators, washing machines, home security systems, and many more interesting appliances. In some cases, a user can speak to their smartphone and have it convert speech into a command for a given appliance. However, it is not common to see some kind of “robot” act out a voice command. Having a personal robot around to fetch a coffee, potentially move heavy objects, or other physical actions could help people with physical disabilities. This robot could even help individuals with impaired vision acting as a guide, similar to how a guide dog could be utilized. Devices are beginning to be seen such as the iRobot vacuum that perform household activities. Speech recognition and robotics are two disciplines that are quickly growing, and this project combines those two disciplines to complete a task. The constraints on the proposed method mainly include performing speech recognition offline. Current applications of speech recognition use deep learning, which utilizes the internet and cloud computing to help decipher words. Completing this project without using the internet will allow the students to learn the building blocks of deciphering speech.

## **2 | Problem Formulation**

Similar systems that influenced this project’s design are the Google Home and Alexa products. Those products use speech recognition to perform various tasks such as internet searches, playing music, and other instructions. The Google Home, for example, in particular can control devices from washing machines to home security systems, but it currently does not control any moving devices or robots. Our proposed project performs a niche role that common home products do not yet do and could be the solution to various problems in everyday life.

## **3 | Project Specifications**

The Raspberry Pi will be used to take a voice input from the Jounivo USB Microphone and also do all of the signal processing of that input.

This will be done offline without the usage of cloud computing [1]. Once the voice is processed, the command must be sent to the robot for it to perform the command.



*Figure 1 - Raspberry Pi 3 B+*

This transmission of data will be wireless, using two XBee modules to do so. With the UART component on the Raspberry Pi and Tiva-C, a single character is written and read to the XBee modules. XCTU is used to configure the XBees to communicate wirelessly with each other via their specific MAC Addresses. Once configured the XBee connected to the Raspberry Pi is controlled with Python and the XBee on the Tiva-C is controlled with Embedded C. The robot will be controlled by a microcontroller, specifically Texas Instrument's Tiva C. This microcontroller is very affordable while also being very powerful, which is very beneficial when involving multiple systems within a single project.

Some of the factors that can affect performance of the final design include signal-to-noise ratio (SNR), efficient programming, the speed of our wireless transmission, and various computation times will dictate the overall performance of this project. Without the implementation of Deep Learning, this speech recognition system will be relatively limited to the users it can accurately recognize. The two largest impacts on its accuracy come from the accent and language of the speaker.



*Figure 2 - Jounivo USB Microphone*

This system cannot recognize any language outside of the English language. It also struggles when a given user has a different accent than the group members. If the commands are spoken very clearly, accents are less of an issue and increase its performance. This system performs adequately in relation to the deliverables given for this project. The graph below depicts the general accuracy of the system between Males and Females (low and high pitch) and the individual's accent. From the accuracy results, within the project group the system recognized their speech very well. Even when other individuals with similar accents tested the system, it maintained relatively good accuracy; people such as Alec Suits, Zachary Gathmann, Samantha Carey, and others. However, when students with much different accents tested the system it much more inaccurate. The thicker the individual's accent, the worse the accuracy got. The final design is more tuned to work for people's accents that are relatable to the group members.

## **4 | Detailed Engineering Analysis and Design Presentation**

### ***4.1 / Design Concept***

The project design has three major subsystems; speech recognition, wireless communication, and the robot. A more in-depth view of the flow of the project is shown in the figure below. Each portion has a very important role in the final design to not only make the project unique, but also have it function the desired way. The final design takes a voice command input and samples it for computational analysis. Once the speech is compared to the list of pre-determined commands, the Raspberry Pi will send the first letter of the command through the XBee modules to the robot. The robot then uses the universal asynchronous receiver/transmitter (UART) to take in the letter received and perform the proper command based on that letter. Once the command has been completed, the robot will echo the command back to the speech processor letting them know the robot is ready for another command.



*Figure 3 - Full System Block Diagram*



## 4.2 / Voice Recognition and Algorithm

### 4.2.1 / Theory and Overview

Starting off, the voice recognition system is the main focus of this project and has the most significant role. The core of the system is a Raspberry Pi that will take a voice signal from a USB microphone, detects if there is speech or not, and then processes any speech in the signal to determine what command was spoken. This process is all done offline using the Python programming language. Recognizing speech is typically done using feature extraction, where the audio signal is broken up into very small segments and each segment is analyzed sequentially. Sound is an entity that changes as time passes, so speech must be analyzed and processed.

The main tool used in this algorithm is the Short-time Fourier Transform (STFT).

$$\text{STFT}\{x[n]\}(m, \omega) \equiv X(m, \omega) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-j\omega n}$$

Speech is fundamentally sequential – which means it must also be analyzed sequentially. The STFT is an equation to show Frequency vs. Time vs. Magnitude, which analytically shows how the frequencies and their respective magnitudes change over a given time interval. This is very powerful for audio since it can show how a sound(s) change over time, even if it is a relatively short amount of time. A key thing to note is men and women speak at higher fundamental frequencies – i.e. women speak roughly 100Hz higher than men on average. Looking at the 0Hz row in the STFT, the algorithm can determine if the speaker is male or female, which then branches off into their respective comparisons to analyze their speech. This branching is talked about in more detail in sections 4.2.3 and 4.2.4.

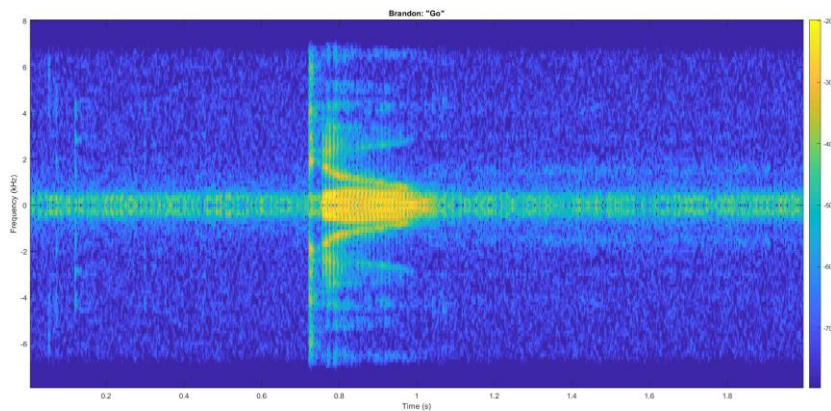


Figure 4 - Spectrogram of Brandon saying "Go"

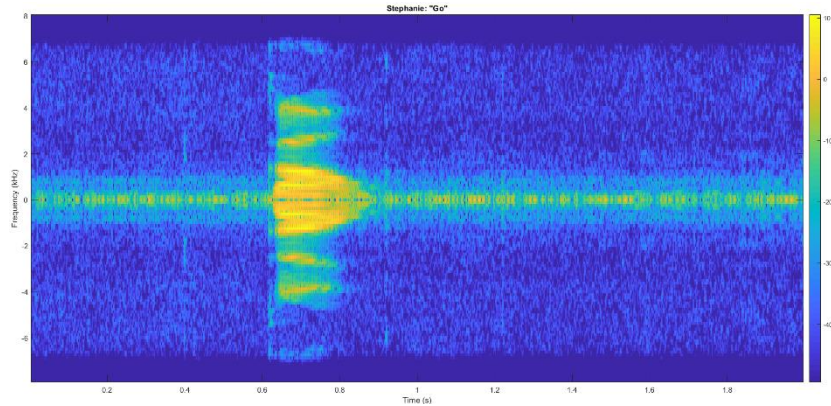


Figure 5 - Spectrogram of Stephanie saying "Go"

Once the algorithm knows the pitch of the speaker it can continue to analyze the audio signal's features which have a correlation with the pitch it determined.

#### 4.2.2 / Speech Detection and Pre-processing

To analyze speech, it must be concluded whether or not there is speech within a given audio signal. The environment in which the voice recognition system is in determines this project's method of speech-detection. Typically, this project is tested and demonstrated in any of the Engineering lab rooms which all roughly have the same acoustic properties. Using a running average along the audio signal and then applying an amplitude threshold to that running average removed anything that was considered non-speech.

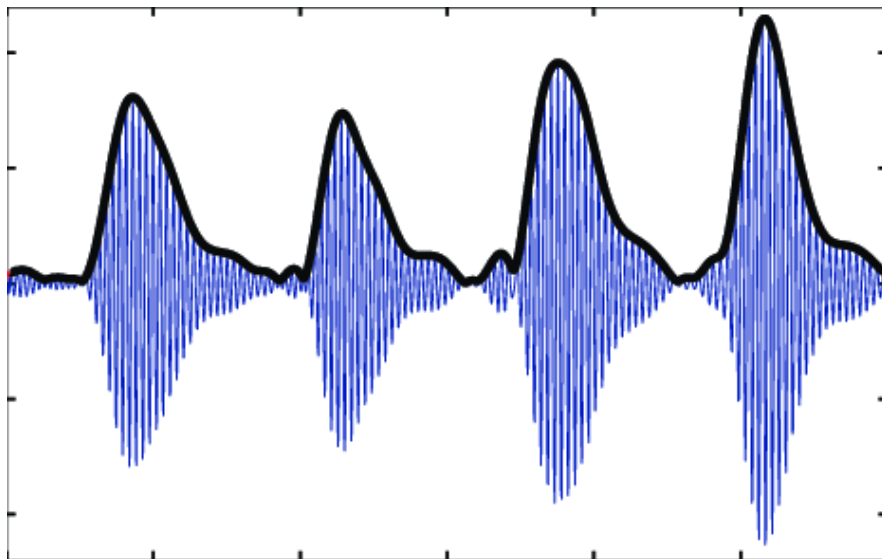


Figure 6 - Average magnitude envelope over an arbitrary waveform

The threshold was made up from recording ambient sound within Shineman 410 and Shineman 236 and seeing how loud the signal got in the times when nobody was speaking. Other useful

techniques for comparing speech vs. non-speech are algorithms are things like the Zero Crossing Rate (ZCR), Short Term-Energy (STE), and Short-Term Autocorrelation (STA). These techniques used in parallel with each other provide a strong prediction of where speech is within an audio signal, and to an extent, can depict what sound or word lies within said audio signal. However, noise can distort the results from these functions meaning they are only good for *detecting* speech in a signal, not actually deciphering what the speech actually is.

Human speech typically does not typically have any important frequency information beyond 8kHz, so the even though the microphone takes in audio at 48kHz it is down sampled to 16kHz. This down sampling is done using an 8<sup>th</sup> order Chebyshev Type 1 IIR Lowpass filter, and then every 3<sup>rd</sup> sample is kept while the rest are discarded. Audio is also sampled as mono rather than stereo to further reduce the number of samples to run calculations on. When the STFT is performed, this preprocessing cuts out a lot of computation time, while retaining vital information on the speech within the signal.

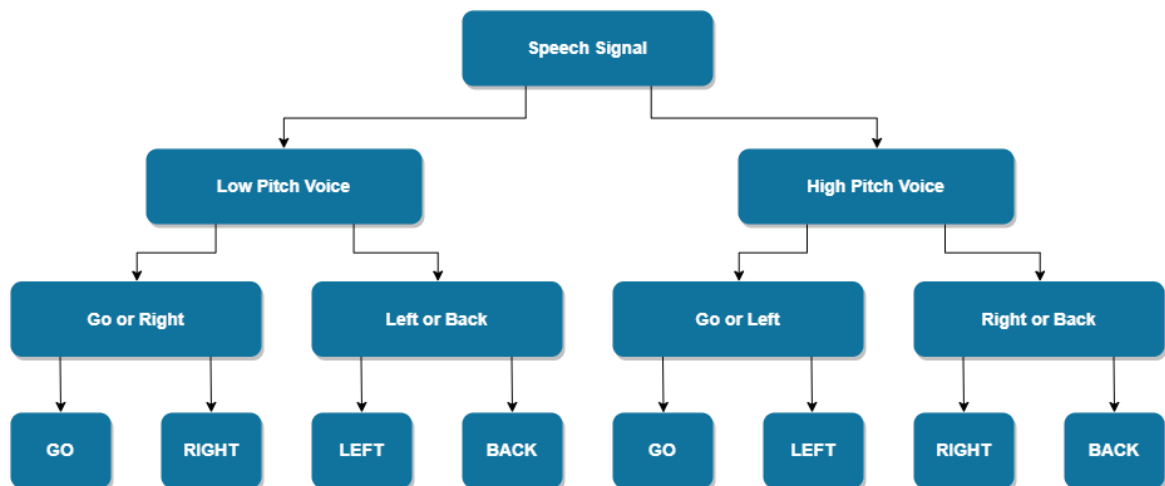


Figure 7 - Speech Recognition Algorithm

#### 4.2.3 / Lower Pitch Speech Algorithm

Assuming the speaker has a lower pitch voice, there are still four commands the system expects. This needs to be narrowed down by deciding whether or not the speaker said “Go” and/or “Right”, or they said “Left” and/or “Back.” This is done by immediately deciding which two of the four commands could it be, and then looking at more features of the signal to pick one choice from the group of two. To narrow down the selection of four commands to two, the algorithm checks the strongest magnitude in the STFT. For “Go” and “Right, on average the strongest magnitude lies between 500Hz and 625Hz. For “Left” and “Back” the strongest magnitude

typically lies between 625Hz and 750Hz. If the algorithm sees the strongest magnitude between 500Hz and 625Hz, it immediately assumes the word must be “Go” or “Right,” which excludes the risk of mistaking the command for being “Left” or “Back.” Likewise, it will ignore “Go” and “Right” if the magnitude falls between 625Hz and 750Hz.

If the algorithm decides the word must be “Go” or “Right” it will look at the frequency bands at 1kHz and 2kHz. In this range on the spectrogram, there is an arm-like structure that appears indicating that over time the magnitude of the frequencies are changing. In “Go” from 1-2kHz, the magnitude of the STFT decreases over time. In “Right” from 1-2kHz, the magnitude of the STFT increases over time. The algorithm checks for the maximums at 1kHz and 2kHz and finds out the indices of the maximums which relate to the time. If the index at 1kHz is larger at 2kHz, this implies there is a decrease in STFT magnitude over time – the algorithm sees this decrease over time and predicts the speaker said “Go.” Otherwise, if there was an increase in magnitude over time from 1kHz to 2kHz shown by the index values, this would imply the speaker said “Right” instead.

In the other scenario where the algorithm thinks the speaker said “Left” or “Back,” it will look at the magnitude of the STFT over time in the frequency ranges of 1-1.5kHz, and 1.5-2kHz. Comparing the overall magnitude of the STFT between the two ranges is how this system predicts if the word is “Left” or “Back.” In “Left” from 1-1.5kHz, the overall magnitude of the STFT is larger – compared to the magnitude of the STFT from 1.5-2kHz. Taking the sum of all STFT values in the two frequency ranges and comparing the two different sums determines whether the word was “Left” or “Back.” If the sum of magnitudes from 1-1.5kHz is larger than the sum from 1.5-2kHz, the spoken word is likely “Left;” otherwise it must be “Back.”

#### *4.2.4 / Higher Pitch Speech Algorithm*

If the speaker is deemed to have a higher pitch voice, the algorithm behaves slightly different compared to the low pitch comparisons. It still splits the four commands into two groups of two, but the two groups are now “Go” and “Left,” or “Back” and “Right.” From spectrogram analysis, it was much easier to split the tree of comparisons this way.

To split the comparisons in this way, the algorithm looks at the 2kHz row from the STFT. If there is a considerably large magnitude at 2kHz, the spoken word is likely to be “Right” or “Back.” To contrast the words “Right” and “Back,” the algorithm looks at the 3kHz and 4kHz

rows in the STFT – and if there is a larger magnitude at 3kHz compared to 4kHz, the spoken word was likely to be “Back”; else it must be “Right.”

However on the other side of the tree, if the algorithm is checking “Go” and “Left” it will be looking at the 2.5kHz and 3kHz rows in the STFT. If there is a stronger magnitude at 2.5kHz compared to 3kHz, the word was likely “Go,” else it must be “Left.”

### ***4.3 / Wireless Transmission of Data***

The XBee is a module that is used for wireless communication and can be configured to act as a receiver and/or a transmitter. There are two XBees in this project: one connected to the Raspberry Pi, and another connected to the Tiva-C. Using the XCTU software developed by Digi, the XBee modules are configured to communicate only with each other using their specific MAC addresses. Using MAC addresses will prevent any other XBee modules from interfering with the transmissions related to this project.



*Figure 8 - XBee S2C Module*

In this project, the XBees are sending and receiving ASCII characters, which must be encoded in 8-bit Unicode Transformation Format (UTF-8). This encoding makes it so when writing to or reading from the XBees, the Raspberry Pi and Tiva-C can look for letters rather than bytes. There is no real difference, but it makes it easier to read the code. For ease in explanation, the XBee connected to the Raspberry Pi will be called XBee A, and the XBee connected to the Tiva-C will be called XBee B.

XBee A is connected to a XBee shield that converts serial to USB, which allows for a very simple USB connection from the XBee to the Raspberry Pi. To read from and write to XBee A, Python and the library *serial* are used. Python comes with the serial library, which is used in this project mainly to write to. The commands the speech recognition algorithm looks for are “Go”,

“Right”, “Left”, and “Back” which respectively correlate to ASCII characters g, r, l, and b. For example: if the program determines the user said “Back” then it will write a ‘b’ to the serial port connected to XBee A. Once the ‘b’ is written to the serial port, the XBee will immediately transmit the character to XBee B on the Tiva-C to control the robot.

XBee B is connected to the Tiva-C using the Universal Asynchronous Receiver-Transmitter (UART) component. First off, the UART must be initialized in order to be used. In this case Port B, pin zero is being used as the receiver pin (RX), and Port B, pin one is being used as the transmitter pin (TX). The character that is received by XBee B is stored in the UART buffer. The program takes the character received and performs a specific command. In the example above, the program would receive the character ‘b’ and perform the goBack() function. The robot will then set the wheels to go in reverse for about two seconds, a delay of around twenty-million clock cycles, and then stop. After the command has been fully completed, XBee B will then send the same character it receives, letting the voice recognition system know the robot is ready to perform another command.

#### ***4.4 / Robot Design and Implementation***

The robot will be controlled by a Tiva-C microcontroller. Once the command is received from the XBee, the robot will perform the given command. The Tiva-C will not have a part in the voice recognition or analysis, it will simply take a command that was sent from the Raspberry Pi and perform the given command for about two seconds. Pulse Width Modulated (PWM) signals sent to the motors will dictate the robot’s overall movement [7]. Based on the specifications for the motor controller, a one millisecond pulse width will make the wheels go full reverse, a two-millisecond pulse width will make the wheels go full forward, and one and a half millisecond pulse width will make the wheels stay neutral. As previously stated, the robot is not the main focus so it will be doing very basic commands; e.g. go, back, left, right. Simple movement functions in combination with the use of universal asynchronous receiver/transmitter (UART) will allow for the functionality needed. There are a few primary functions involved in the coding of the robot. Those include the UART initialization, motor initialization, motor movement functions, and the main function which determines which command to perform based on the command received.

#### 4.5 / System Characteristics and Performance

In the end, the speech recognition algorithm performed very well considering the limitations set at the beginning of the project. Figure 9 depicts how accurate the algorithm performed by the end of the project's design. It struggled more with female voices, as the initial design was so heavily influenced by a male speaker's characteristics. Despite being fundamentally built off of male speaker samples and analysis, it has respectable accuracy. Within the group it was over 95% accurate and retained fair accuracy for other individuals who tested the system. The largest drop in accuracy comes from other accents that vary from the group's accents. This recognition algorithm is not going to perform well for speakers that do not have a relatively similar accent to the group members, but if the speaker's accent is similar it has proven to perform very well overall.

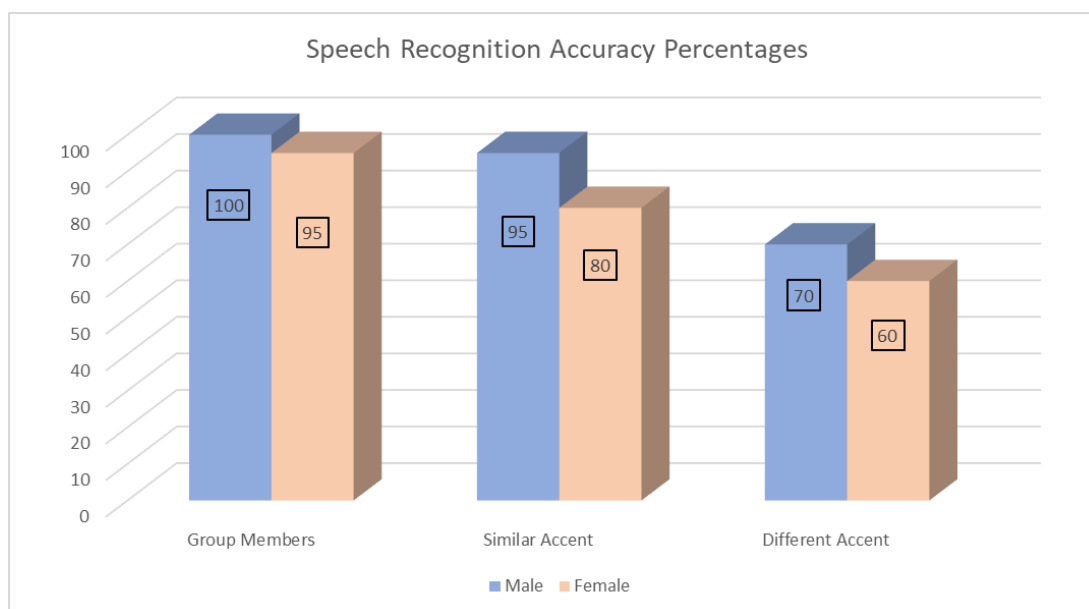


Figure 9 - Speech Recognition Accuracy Measurements

As for the robot, the power consumption is fairly low. The battery runs at three thousand milliamp hours. The components that draw from the battery include the XBee, the Tiva, and the two Vex motors. The XBee draws forty-five milliamps, the Tiva draws thirty milliamps, and the motors draw one and a half amps while in motion and twelve milliamps when idle. It is estimated that the system will be idle approximately ninety percent of the time and will be active about ten percent of the time. With those numbers, the following system run time can be computed.

*Idle System Run Time:  $3000\text{mAH} - 99\text{mAH} \approx 30 \text{ hours}$*

*Active System Run Time:  $3000\text{mAH} - 3075\text{mAH} \approx 0.97 \text{ hours}$*

*Total System Run Time:  $(30 * .90) + (0.97 * .10) \approx 27$  hours*

## **5 | Deliverables**

1. Finished project will be a voice-controlled robot.
2. Speech recognition system is a stationary system on a table or desk.
  - a. Will utilize wall outlet power
  - b. Processing done by a Raspberry Pi
3. Speech recognition is done offline.
  - a. The entire process is local
4. The robot will perform basic movement commands:
  - a. Go, back, left, right.
5. Wireless Capability
  - a. Speech recognition system and robot communicate wirelessly
6. Characterization of our System
  - a. Performance will be evaluated based on variables such as noise and the delay between the user speaking and the robot reacting to the command
  - b. Power consumption of the system comparable to baseline values measured by the Raspberry Pi Foundation
7. The system will be always listening.

## **6 | Cost Analysis & Bill of Materials**

### ***6.1 / Estimated Cost***

The cost of this project was approximately \$307. That is not including the requirements of needing to pay for some sort of continuous power supply (i.e. electrical bill). Regarding the lifetime cost of this system, there is not much to be replaced. The 7.2V battery has been shown in section 4.5 to last a little over one day of on-and-off usage. The speech recognition system is powered via wall outlet, and the Pi 3B+ is rated to use approximately 5 Watts at 400% CPU load [8]. This system never reached above 50% CPU usage, which means it was drawing roughly 2.5 Watts during the times it was trying to analyze speech. Users are not typically speaking commands continuously with no breaks, so it was estimated the system is idle roughly 90% of any given day. Figure 10 listed below shows various power ratings of the Raspberry Pi 3B+.



## Raspberry Pi 3 B+ Baseline

Pi State	Power Consumption
Idle	350 mA (1.9W)
ab -n 100 -c 10 (uncached)	950 mA (5.0 W)
400% CPU load (stress --cpu 4)	980 mA (5.1 W)

Figure 10 - Raspberry Pi 3B+ power measurements

### 6.2 / Component Selection

The components that were chosen include the Raspberry Pi, Jounivo USB Microphone, XBee S2C, and the Tiva-C microcontroller. The Pi has a very fast ARM processor running at around 1.4 GHz, giving us relatively fast processing capabilities [8]. The Jounivo microphone could be replaced with any standard USB microphone, but it was cost effective and performed adequately. XBee modules are commonly considered an easy way to wirelessly communicate between two devices, making this a fairly obvious choice for our project's implementation [6]. The Tiva-C microcontrollers are very well documented, cheap, and powerful. Their 80 MHz clock rate will help make the system operate as fast as possible [9]. The following table shows our components with their respective quantities, costs, and their manufacturers. A quantity of "X" in the table implies we are unsure of how many of that given component will actually be needed and/or used.

Component	Quantity	Cost per Part (\$)	Manufacturer
Raspberry Pi	1	\$60.00	Raspberry Pi Foundation
Microcontroller (Tiva-C)	1	\$15.00	Texas Instruments
USB Microphone	1	\$15.00	Jounivo
XBee S2C Modules	2	\$20.00	Digi
XBee Shields	2	\$20.00	Digi
Vex Motors	2	\$15.00	Vex Robotics
Various Gears	1	\$13.00	Vex Robotics
Motor Controllers	2	\$10.00	Vex Robotics

Battery (robot)	1	\$30.00	Vex Robotics
Metal Frame (robot)	1	\$20.00	N/A
Screws, Bolts, etc.	X	\$5.00	N/A
<b><i>Total Estimated Cost (\$)</i></b>			
<b>\$274.50</b>			

*Table 1 - Cost Analysis*

## **7 | Hazards and Failure Analysis**

This design has no potential threats to an individual's safety. The speech recognition is done on an immobile system using safe, standardized wall outlet power while the robot is the only thing that could be a cause for concern. The robot could be designed with sensors to avoid damaging anything or injuring people while it is in motion, but with it being designed on a small scale it is unlikely to harm any individual or any property even if a collision possibly could occur. Potential for future application is to remove the batteries from the robot and implement renewable energy instead, such as solar power.

## **8 | Learning Outcomes**

- 1. an ability to identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics**

*This project uses several methods of problem solving and techniques learned in engineering. Programs such as MATLAB and XCTU will be used for testing and to then find the best method of analyzing the voice signal implementing an algorithm on the raspberry pi.*

- 2. an ability to apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors**

*This learning outcome was met by considering environmental costs and creating a low power system. Public safety was a topic for discussion but due to the scale of the project will not be an issue.*

- 3. an ability to communicate effectively with a range of audiences**

*Implementing the voice-controlled robot has given the group multiple chances to give presentations to faculty as well as students. These opportunities include presenting during the SRC donation as well as during the end of the year final presentation.*

- 4. an ability to recognize ethical and professional responsibilities in engineering situations and make informed judgments, which must consider the impact of engineering solutions in global, economic, environmental, and societal contexts**

*This learning outcome was addressed in the project by building a small robot that would not cause harm to society. Economically speaking, the prototype would be somewhat costly, but one could produce these much cheaper with few future improvements.*

- 5. an ability to function effectively on a team whose members together provide leadership, create a collaborative and inclusive environment, establish goals, plan tasks, and meet objectives**

*Each group member was assigned a specific role at the beginning of the project. After weekly meetings with our advisor, those tasks would be updated and goals would be set before the next meeting occurred.*

- 6. an ability to develop and conduct appropriate experimentation, analyze and interpret data, and use engineering judgment to draw conclusions**

*Through the project the group gained a lot of experience in experimenting, analyzing and interpreting data. Having to figure out the most effective method for speech recognition required a lot of data analysis and testing.*

- 7. an ability to acquire and apply new knowledge as needed, using appropriate learning strategies**

*The entire project for the group was completely new ground. Research done on speech recognition included looking up methods on how to perform the needed class, including deep learning and neural networks that are not covered in the curriculum of the given courses. As for the robot, the Tiva-C was a completely new microprocessor that required completion of multiple labs to gain familiarity with the processor. Similarly the wireless communication was a completely new idea to the group that no members were familiar with.*

## **9 | Standards Used in the Design**

1. Zigbee (IEEE 802.15.4)

- a. We chose Zigbee because they are low power, and for our project's implementation the wireless transfer of data does not need to go a very far distance. XBee modules under the Zigbee standard are a good choice for the scope of our project.
- 2. Floating-Point Arithmetic (IEEE 754)
  - a. This is the technical standard for floating-point arithmetic that the Tiva-C microcontroller uses.
- 3. USB Industry Standard (Tiva-C Launchpad)
  - a. USB 2.0 for the device interface
  - b. USB Micro-B plug to a USB-A plug cable
- 4. Microphone Specifications (AN-1112)
  - a. This standard details various specifications a microphone will be designed by.
- 5. PEP 8 – Style Guide for Python Code
  - a. Regarding syntax, imports, and other fundamental aspects of writing python code
  - b. Libraries used:
    - i. capstoneLib (custom functions written by the group)
    - ii. NumPy
    - iii. PySoundFile
    - iv. Matplotlib
    - v. Serial
    - vi. SciPy
    - vii. SoundDevice
    - viii. Pandas
    - ix. Time
    - x. PyAudio
- 6. Embedded C Coding Standard
  - a. Designed to minimize bugs in firmware
  - b. Also improves maintainability and portability of embedded software
  - c. Details set of guiding principles, naming conventions, and much more

## **10 | Conclusions**

### ***10.1 / Summary and Results***

This project is a voice-controlled robot of three subsystems: Speech Recognition, Wireless Communication, and the Robot itself. This project was successfully completed and characterized in detail in section 4.5. However, one of the deliverables was not met: Always Listening (AL). This deliverable ended up becoming a much larger challenge than any of the group members and advisors expected. In the end, it ended up being a programming issue rather than a conceptual issue. Programming in real-time to process speech continuously without slowing down the system was a barrier the group could not overcome within the project's deadline. With speech recognition typically being done with Deep Learning algorithms and models, the group's form of speech recognition could not perform well in real-time, leading to poor results. The material needed to make this project function properly was already a lot to learn, and then with AL requiring real-time programming and speech recognition it became much more difficult. Using PyAudio's *stream* class, it let the group initialize an infinite stream of audio through the microphone – but properly storing data in memory to analyze without slowing down the program became a challenge.

Despite the group's struggle with real-time programming on a totally new system and coding language, the remainder of the project turned out exceptionally well. From the moment the Raspberry Pi stops recording for a voice command to the robot actually performing its task, there is only a few seconds of delay. This was a concern in initial design, as we had no idea if this was going to perform in a timely manner or not. From section 4.5, the power consumption of the robot and the accuracy of the speech recognition is exceptional to our predictions from the previous semester. With the speech recognition being over 95% accurate within the group, and the robot lasting over a full day on a single battery charge, the final results turned out to be much better than expected.

## ***10.2 / Improvements and Future Work***

There are infinitely many ways to improve this system. For speech recognition, there could be the usage of popular databases from Google and Amazon to handle all of the speech recognition and allow individual to improve the robot further. Robots are now getting to the point where they have their own Artificial Intelligence (AI) which lets them think, speak, and respond to a user conversing with them. Sophia is an example of a robot that uses speech recognition to respond to its user; where she can actually have full conversations with whoever speaks to her.

Removing the robot entirely would allow individuals to fully focus on speech recognition. Retaining the same offline restriction, students could focus more on the programming of speech

processing and implement a custom written neural network. With enough training data, students could write a very respectable offline speech recognition system that could potentially recognize full sentences rather than just single-word commands.





















## **References**

- [1] Gazetic, E. (2017). *Comparison Between Cloud-based and Offline Speech Recognition Systems*. [online] Mediatum.ub.tum.de. Available at: <https://mediatum.ub.tum.de/doc/1399984/1399984.pdf>
- [2] IEEE Guide for Developing System Requirements Specifications. (1998). [ebook] IEEE. Available at: <https://ieeexplore.ieee.org/document/741940/references#references>.
- [3] Zhang, Y. (2019). *Speech Recognition Using Deep Learning Algorithms*. [online] Cs229.stanford.edu. Available at: [http://cs229.stanford.edu/proj2013/zhang\\_Speech%20Recognition%20Using%20Deep%20Learning%20Algorithms.pdf](http://cs229.stanford.edu/proj2013/zhang_Speech%20Recognition%20Using%20Deep%20Learning%20Algorithms.pdf).
- [4] S. Ault, R. Perez, C. Kimble and J. Wang, "On Speech Recognition Algorithms", *International Journal of Machine Learning and Computing*, vol. 8, no. 6, 2018. Available: <http://www.ijmlc.org/vol8/739-DA0052.pdf>.
- [5] Weisstein, Eric W. "Fourier Transform." From MathWorld--A Wolfram Web Resource. <http://mathworld.wolfram.com/FourierTransform.html>
- [6] "Digi XBee3 ZigBee 3.0", Digi.com, 2019. [Online]. Available: <https://www.digi.com/products/embedded-systems/rf-modules/2-4-ghz-modules/XBee3-zigbee-3>.
- [7] "What is a Pulse Width Modulation (PWM) Signal and What is it Used For?", Knowledge.ni.com, 2018. [Online]. Available: <https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019OkFSAU>.
- [8] Raspberry Pi 3 Model B+ Specifications. Raspberry Pi Foundation, 2018.
- [9] TM4C123GH6PM Microcontroller Data Sheet. Texas Instruments, 2014.
- [10] 2.7V 4-Channel/8-Channel 10-Bit A/D Converters with SPI Serial Interface. Microchip Technology Inc., 2008.

## **Appendices**

### Appendix A

*Processing, Power and I/O*

Raspberry Pi 3 GPIO Header				
Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I <sup>2</sup> C)		DC Power 5v	04
05	GPIO03 (SCL1 , I <sup>2</sup> C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I <sup>2</sup> C ID EEPROM)		(I <sup>2</sup> C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Rev. 2  
29/02/2016

[www.element14.com/RaspberryPi](http://www.element14.com/RaspberryPi)

*Figure 11 - Pi GPIO pins*

## Specifications

<b>Processor:</b>	Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz
<b>Memory:</b>	1GB LPDDR2 SDRAM
<b>Connectivity:</b>	<ul style="list-style-type: none"> <li>■ 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE</li> <li>■ Gigabit Ethernet over USB 2.0 (maximum throughput 300Mbps)</li> <li>■ 4 × USB 2.0 ports</li> </ul>
<b>Access:</b>	Extended 40-pin GPIO header
<b>Video &amp; sound:</b>	<ul style="list-style-type: none"> <li>■ 1 × full size HDMI</li> <li>■ MIPI DSI display port</li> <li>■ MIPI CSI camera port</li> <li>■ 4 pole stereo output and composite video port</li> </ul>
<b>Multimedia:</b>	H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics
<b>SD card support:</b>	Micro SD format for loading operating system and data storage
<b>Input power:</b>	<ul style="list-style-type: none"> <li>■ 5V/2.5A DC via micro USB connector</li> <li>■ 5V DC via GPIO header</li> <li>■ Power over Ethernet (PoE)–enabled (requires separate PoE HAT)</li> </ul>
<b>Environment:</b>	Operating temperature, 0–50 °C
<b>Compliance:</b>	For a full list of local and regional product approvals, please visit <a href="http://www.raspberrypi.org/products/raspberry-pi-3-model-b+">www.raspberrypi.org/products/raspberry-pi-3-model-b+</a>
<b>Production lifetime:</b>	The Raspberry Pi 3 Model B+ will remain in production until at least January 2023.

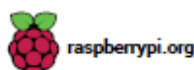
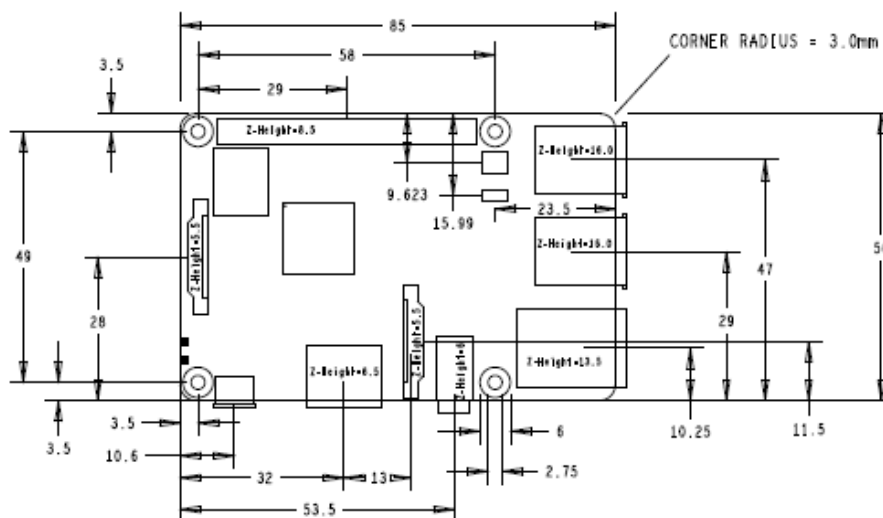


Figure 12 - Pi 3B+ Data Sheet



## Physical specifications



### Warnings

- This product should only be connected to an external power supply rated at 5V/2.5A DC. Any external power supply used with the Raspberry Pi 3 Model B+ shall comply with relevant regulations and standards applicable in the country of intended use.
- This product should be operated in a well-ventilated environment and, if used inside a case, the case should not be covered.
- Whilst in use, this product should be placed on a stable, flat, non-conductive surface and should not be contacted by conductive items.
- The connection of incompatible devices to the GPIO connection may affect compliance, result in damage to the unit, and invalidate the warranty.
- All peripherals used with this product should comply with relevant standards for the country of use and be marked accordingly to ensure that safety and performance requirements are met. These articles include but are not limited to keyboards, monitors, and mice when used in conjunction with the Raspberry Pi.
- The cables and connectors of all peripherals used with this product must have adequate insulation so that relevant safety requirements are met.

### Safety instructions

To avoid malfunction or damage to this product, please observe the following:

- Do not expose to water or moisture, or place on a conductive surface whilst in operation.
- Do not expose to heat from any source, the Raspberry Pi 3 Model B+ is designed for reliable operation at normal ambient temperatures.
- Take care whilst handling to avoid mechanical or electrical damage to the printed circuit board and connectors.
- Whilst it is powered, avoid handling the printed circuit board, or only handle it by the edges to minimise the risk of electrostatic discharge damage.

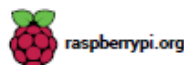


Figure 13 - Pi 3B+ Physical Specifications

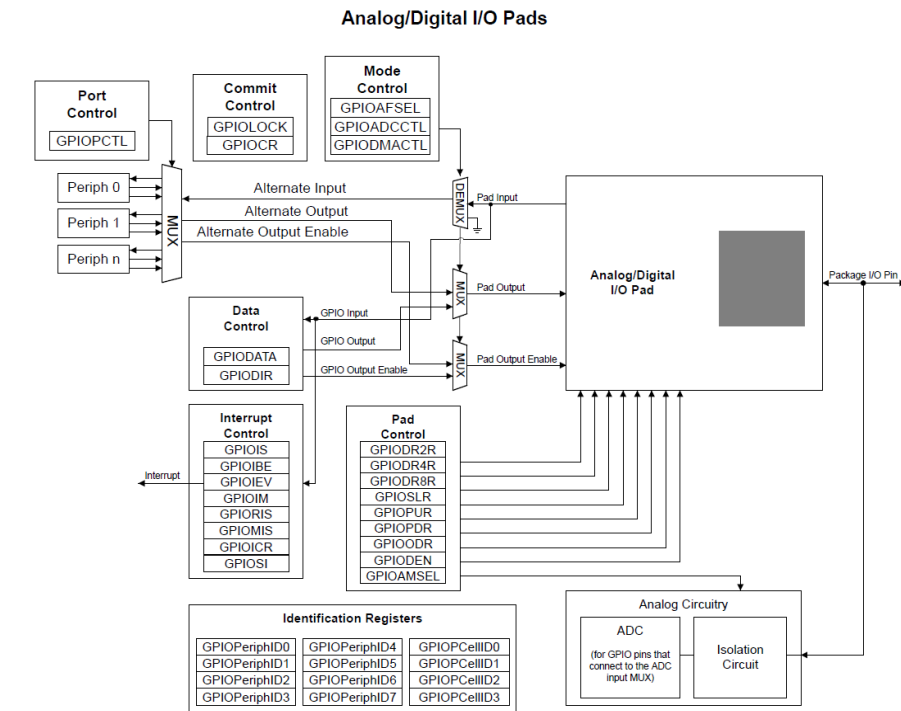


Figure 14 - Tiva-C Launchpad I/O

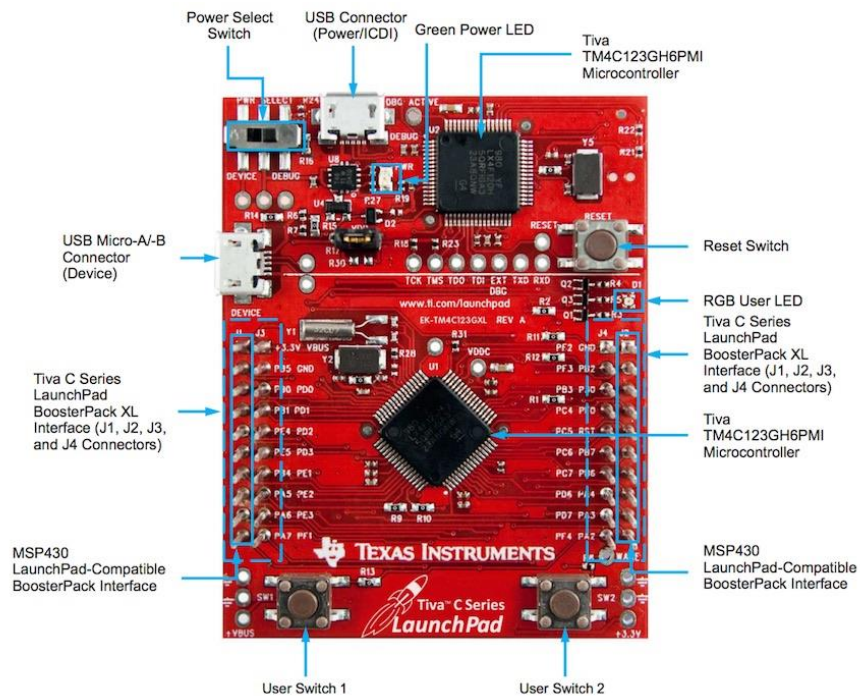


Figure 15 - Tiva-C Launchpad Physical Connections

## Appendix B

### *Code and Graphics*

```

import capstoneLib as cap
import numpy as np
import scipy.signal as sig
import sounddevice as sd
import serial
import pandas as pd
import time

##### Sampling Rate values and recorder initialization
fs = 48000 # initial sample rate
ds = 16000 # downsampled rate
sec = 3 # seconds
duration = int(sec*fs)
sd.default.channels = 1 # mono recording

##### Tx XBee
ser1 = serial.Serial()
ser1.baudrate = 9600
ser1.port = 'COM5'
ser1.open()

##### Record speaker for "sec" seconds
time.sleep(1)
print('Talk now')
voice = sd.rec(duration, fs)
sd.wait()
print('Done...')

##### Speech Detection and Preprocessing
voice = voice[:,0]
voice = cap.normalize(voice)
dsVoice = sig.decimate(voice, 3)
mask, noise = cap.envelope(dsVoice, ds, 0.04)
speech = dsVoice[mask]

##### Do STFT
F, T, STFT = sig.stft(x=speech,
                      fs=ds,
                      window='hann',
                      nperseg=128,
                      noverlap=96,
                      nfft=128,
                      return_onesided=False)

# Magnitude of the STFT
aSTFT = abs(STFT)

# Check High or Low pitch (Male vs. Female)
zeroHz = aSTFT[ [0], ]
avgZ = np.mean(zeroHz)

if avgZ >= 0.0125:
    print('Low pitch voice.')
    ##### Go and Right are typically stronger at 500Hz
    ##### Left and Back are typically stronger at 625Hz
    sec4 = aSTFT[ [4], ] # 500Hz
    low = np.sum(sec4)

    sec5 = aSTFT[ [5], ] # 625Hz
    high = np.sum(sec5)

    if low > high: # Compare RIGHT and GO
        row1 = aSTFT[ [8], ] # Row for 1kHz
        I1 = np.where( row1 == np.amax(row1) )

        row2 = aSTFT[ [16], ] # Row for 2kHz
        I2 = np.where( row2 == np.amax(row2) )

        if I1[1] > I2[1]:
            print('You said "Go."')
            msg = bytes('g', 'utf-8')
        else:
            print('You said "Right."')
            msg = bytes('r', 'utf-8')

    else: # Compare LEFT and BACK
        first = []
        for w in range(9,13): # 1125 - 1500 Hz
            fRow = aSTFT[ [w], ]
            first.append(fRow)
        fArr = np.array(first)
        firstSum = np.sum(fArr)

        second = []
        for q in range(13,17): # 1625 - 2000 Hz
            sRow = aSTFT[ [q], ]
            second.append(sRow)
        sArr = np.array(second)
        secondSum = np.sum(sArr)

```

```

        if firstSum > secondSum:
            print('You said "Left."')
            msg = bytes('l', 'utf-8')
        else:
            print('You said "Back."')
            msg = bytes('b', 'utf-8')

    else:
        print('High pitch voice.')
        ##### Split to GO/LEFT or RIGHT/BACK
        twoK = aSTFT[ [16] , ] # Row for 2kHz
        check = np.mean(twoK)

        if check < 0.009:
            ##### GO vs LEFT
            cut1 = aSTFT[ [20] , ] # Row for 2.5kHz
            avg1 = np.mean(cut1)

            cut2 = aSTFT[ [24] , ] # Row for 3kHz
            avg2 = np.mean(cut2)

            if avg1 > avg2:
                print('You said "Go."')
                msg = bytes('g', 'utf-8')
            else:
                print('You said "Left."')
                msg = bytes('l', 'utf-8')
        else:
            ##### RIGHT vs BACK
            cut1 = aSTFT[ [24] , ] # Row for 3kHz
            avg1 = np.mean(cut1)

            cut2 = aSTFT[ [32] , ] # Row for 4kHz
            avg2 = np.mean(cut2)

            if avg1 > avg2:
                print('You said "Back."')
                msg = bytes('b', 'utf-8')
            else:
                print('You said "Right."')
                msg = bytes('r', 'utf-8')
    print('~~~~~')

##### Write to the Tx XBee so it transmits to the Rx XBee
##### on the Tiva-C
ser1.write(msg)
ser1.close()

```

*Python Block 1 - Full Speech Recognition Algorithm*

```

'''
Custom functions written for ECE492
-----
Libraries used: numpy, pandas
'''
import numpy as np
import pandas as pd

def normalize(data):
    # Normalizes input values based on absolute maximum
    # Requires: numpy
    dArray = np.array(data)
    dMax = max(abs(dArray))
    return dArray/dMax

def envelope(data, rate, threshold):
    # Returns indices in data below the specified threshold
    # Requires: pandas, numpy
    mask = []
    noise = []
    data = pd.Series(data).apply(np.abs)
    dataMean = data.rolling(window=int(rate/10),
                           min_periods=1,
                           center=True).mean()
    for mean in dataMean:
        if mean > threshold:
            mask.append(True)
            noise.append(False)
        else:
            mask.append(False)
            noise.append(True)
    # Mask = above threshold
    # Noise = below threshold
    return mask, noise

def ster2mono(data):
    # Crude Stereo to Mono conversion
    return ( data[:,0] + data[:,1] )/2

```

*Python Block 2 - Custom Python Library for main speech recognition algorithm*

```

'''
Custom functions written for ECE492
-----
Libraries used: numpy, pandas
'''
import numpy as np
import pandas as pd

def normalize(data):
    # Normalizes input values based on absolute maximum
    # Requires: numpy
    dArray = np.array(data)
    dMax = max(abs(dArray))
    return dArray/dMax

def envelope(data, rate, threshold):
    # Returns indices in data below the specified threshold
    # Note: "mask" are indices, NOT the cropped signal
    # Requires: pandas, numpy
    mask = []
    noise = []
    data = pd.Series(data).apply(np.abs)
    dataMean = data.rolling(window=int(rate/10),
                           min_periods=1,
                           center=True).mean()

    for mean in dataMean:
        if mean > threshold:
            mask.append(True)
            noise.append(False)
        else:
            mask.append(False)
            noise.append(True)
    return mask, noise

def ster2mono(data):
    # Stereo to Mono conversion
    return ( data[:,0] + data[:,1] )/2

def quantize(data,nbits):
    # Quantize values to any positive integer bit value, n
    # Requires: normalize, numpy
    n = (2**nbits)
    a = normalize(data)
    b = a+1
    c = b/2
    d = c*(n-1)
    e = np.around(d)
    Q = 2/n
    return (Q/2) + (e*Q) - 1

def fftprops(data, Fs):
    # Generates FFT and Freq. axis related to data input
    # Requires: numpy, normalize
    # Setting up the frequency axis
    data = normalize(data)
    N = len(data)
    K = np.arange(N)
    T = N/Fs
    freq = K/T # Double sided spectrum
    freq = freq[range(int(N/2))] # Single sided spectrum
    # Performing the fft of input data
    FFT = np.fft.fft(data)/N # normalized by length of signal
    FFT = FFT[range(int(N/2))]
    # Returns the magnitude of the FFT, and the frequencies in a vector
    # Note: Set up for single sided spectrum
    return FFT, freq

def findStart(data):
    N = len(data)
    x = normalize(data)
    xP = np.power(x,2)

    M = 1000
    rect = np.ones((M,))
    xP = np.convolve(xP,rect,mode='same')
    xP = normalize(xP)
    Thresh = 0.15

    for i in range(N):
        if xP[i] >= Thresh:
            if i+1 > N:
                start = N
            else:
                start = i+1
            if start+500 > N:
                end = N
            else:
                end = start+500
            pRange = np.array(xP[start:end])
            avgP = np.sum(pRange) / len(pRange)

```

```

        if avgP > Thresh:
            index = i
            break
        else:
            index = 0
    signal = x[index:N]
    return signal, index

def findEnd(signal):
    # Finds end of a signal
    # Requires: numpy, findStart
    xFlip = np.flip(signal)
    yFlip, yIndex = findStart(xFlip)
    y = np.flip(yFlip)
    end = len(signal) - yIndex
    return y, end

def hanSig(data):
    # Takes raw input data and windows it with a Hanning Window
    # Requires: numpy, findStart, findEnd
    X, S = findStart(data)
    Y, E = findEnd(data)
    N = len(data)
    M = E - S # Length of signal Start(S) to End(E)

    if S-(M//4) < 0:
        start = S
    else:
        start = S-(M//4)

    if E+(M//4) > N:
        end = N
    else:
        end = E+(M//4)

    wHan = np.hanning(int(end-start))

    windowed = []
    j = 0
    for i in range(start,end):
        val = data[i]*wHan[j]
        windowed.append(val)
        j += 1
    return windowed, M

def sinc_Ntaps(fcut, fs, N):
    # Makes sinc h[n] of N taps
    # Input the desired cutoff frequency, sampling rate, and odd value of N
    # Requires: numpy
    fc = fcut/fs
    n = np.arange( -(N-1)/2 , (N-1)/2, 1)
    h = 2*fc*np.sinc(2*fc*n)
    return h

def peakFreq(FFT, fAxis):
    # Given an FFT and Frequency Axis, it finds the peak frequency
    indx = 0
    for i in range(len(FFT)):
        if FFT[i] == max(FFT):
            indx = i
            break
    peakF = fAxis[indx]
    return peakF

def limiter(data):
    # Any large values greater than typical sound amplitude are
    # reduced to limit=0.055.
    negLim = -0.055
    posLim = 0.055
    N = len(data)
    for i in range(N):
        if data[i] >= posLim:
            data[i] = posLim
        elif data[i] <= negLim:
            data[i] = negLim
        else:
            data[i] = data[i]
    return data

def SNR_dB(voice, noise_pwr):
    # Crude SNR calculation function that ended up not
    # being ideal for proper analysis.
    # Requires: numpy and envelope function from capstoneLib
    mask = envelope(voice, 48000, 0.015)
    speech = voice[mask]
    signal_pwr = np.mean(np.power(abs(speech), 2))
    ratio = int(signal_pwr / noise_pwr)
    snr_dB = 10*np.log10(ratio)
    return snr_dB

```

*Python Block 3 - All custom functions used throughout the project*

```

''' Record audio and save as .wav file '''

import capstoneLib as cap
import numpy as np
import soundfile as sf
import scipy.io.wavfile
import scipy.signal as sig
import matplotlib.pyplot as plt
import sounddevice as sd

# Jounivo Microphone: Fs = 48kHz, Channels = 1(mono) or 2(stereo)
fs = 48000 # Hz
duration = 2 # seconds
sd.default.channels = 1 # mono recording

# Record for duration (seconds) at sampling rate fs
print('Begin')
voice = sd.rec( int(duration*fs), fs)
sd.wait()
print('Done...')

mVoice = cap.ster2mono(voice)
N = len(mVoice)
time = np.arange(0, N*ts, ts) # time, seconds
samples = np.arange(0, N, 1) # samples, n

# Plot to check if the recorded audio is worth keeping
plt.plot(time, mVoice)
plt.xlabel('Time, s')
plt.ylabel('Amplitude')
plt.show()

filename = 'testfile.wav'
scipy.io.wavfile.write(filename, fs, mVoice)

```

*Python Block 4 - Script to record audio using the Jounivo USB Microphone*



```

import capstoneLib as cap
import numpy as np
import soundfile as sf
import scipy.io.wavfile
import scipy.signal as sig
import matplotlib.pyplot as plt
import sounddevice as sd

'''
Jounivo Microphone:
    Fs = 48kHz
    Channels=1 if mono | Channels=2 if stereo
    16-bit ADC
'''

# Jounivo Microphone: Fs = 48kHz, Channels = 1(mono) or 2(stereo)
fs = 48000 # Hz
duration = 2 # seconds
ts = 1/fs # period
sd.default.channels = 2 # stereo recording

# Record for duration (seconds) at sampling rate fs
print('Talk now')
voice = sd.rec( int(duration*fs), fs)
sd.wait()
print('Done recording...')

# Convert Stereo to Mono, limit amplitude, normalize, downsample
dsVoice = sig.decimate(voice, 3)
mVoice = cap.ster2mono(dsVoice)
limited = cap.limiter(mVoice)
normVoice = cap.normalize(limited)
N = len(normVoice)
time = np.arange(0, N, 1)

# Crop start/end of signal and multiply cropped data by Hanning Window
X, S = cap.findStart(normVoice)
Y, E = cap.findEnd(normVoice)
M = int(E - S) # Length of signal Start(S) to End(E)

# Conditional statements to make sure there is no indexing outside of the signal
if S-(M//10) < 0:
    start = S
else:
    start = S-(M//10)

if E+(M//10) > N:
    end = N
else:
    end = E+(M//10)

K = int(end - start)
if K%2 == 0: # Make window Length odd number
    K += 1
start += 1 # Signal dimension must equal Window dimension
wHan = np.hanning(int(K))

windowed = []
j = 0
for k in range(start,end):
    val = normVoice[k]*wHan[j]
    windowed.append(val)
    j += 1
winTime = np.arange(0,len(windowed),1)

# Fourier Transform of windowed voice signal and peak frequency
FFT, fax = cap.fftprops(windowed,fs)
FTN = cap.normalize(FFT)

# Smooth Fourier Transform with convolution of 10 ones
ft = abs(FFT)
F = len(FFT)
Y = 10
rect = np.ones((Y,))
ftN = np.convolve(ft,rect,mode='same')
ftN = cap.normalize(ftN)

peakF = cap.peakFreq(ftN, fax)
peakF = np.around(peakF)
print("Peak Frequency Component:",peakF,"Hz")

''' 4 Subplots
1. Raw time domain voice signal
2. Windowed/Cropped time domain signal
3. Spectrum of Windowed WITHOUT smoothing/convolution
4. Spectrum of Windowed WITH smoothing/convolution '''
fig, ax = plt.subplots(2, 2, figsize=(8,6))
ax[0,0].plot(time, mVoice)
ax[0,0].set_xlabel('Samples')
ax[0,0].set_ylabel('Amplitude')

```

```

ax[0,1].plot(winTime, windowed)
ax[0,1].set_xlabel('Samples')
ax[0,1].set_ylabel('Windowed Amp.')

ax[1,0].plot(fax, abs(FFTN))
ax[1,0].set_xlabel('Frequency, Hz')
ax[1,0].set_ylabel('Magnitude')
ax[1,0].axis([0, 1000, 0, 1])
ax[1,0].set_xticks(np.arange(0,1000,100))

ax[1,1].plot(fax, ftN)
ax[1,1].set_xlabel('Frequency, Hz')
ax[1,1].set_ylabel('Magnitude')
ax[1,1].axis([0, 1000, 0, 1])
ax[1,1].set_xticks(np.arange(0,1000,100))

fig.suptitle('"Go" at ' + str(peakF) + 'Hz')
plt.savefig('C:\\Users\\brand\\Desktop\\CapstoneGraphs\\Go15 Sig-Window-FFT.png',
            bbox_inches='tight')
plt.show()

#### Uncomment to save recorded audio
# scipy.io.wavfile.write('file.wav', fs, mVoice)

```

*Python Block 5 - Trial script analyzing the Fourier Transform on speech waveforms*

```

import numpy as np
import capstoneLib as cap
import sounddevice as sd
import matplotlib.pyplot as plt

'''
This script records your voice for *duration* amount of seconds for
*AMOUNT* number of times and plots the Magnitude Spectrum
of the word you spoke.

Outputs 5 waveforms on 1 graph, and the highest frequency component
within each iteration
'''

# JOUNIVO mic: Fs = 48kHz, Channels = 1(mono) or 2(stereo)
fs = 48000 # Hz
duration = 1.5 # seconds
ts = 1/fs # period
sd.default.channels = 2 # stereo recording

FFTs = []
faxes = []
Freqs = []
SampleNums = []
AMOUNT = 5 # How many times do you want to say the word?
print("----- BEGINNING -----")
for q in range(AMOUNT):

    # Record for duration (seconds) at sampling rate fs
    print("Talk now")
    voice = sd.rec(int(duration*fs), fs)
    sd.wait()
    print("Done recording \n~~~~~")

    # Convert Stereo to Mono, Limit amplitude, normalize from [-1,1], make time axis
    mVoice = cap.ster2mono(voice)
    limited = cap.limiter(mVoice)
    normVoice = cap.normalize(limited)
    N = len(normVoice)
    time = np.arange(0, N, 1)

    # Crop start/end of signal and multiply cropped data by Hanning Window
    X, S = cap.findStart(normVoice)
    Y, E = cap.findEnd(normVoice)
    N = len(normVoice)
    M = int(E - S) # Length of signal Start(S) to End(E)
    ## SampleNums.append(M) #####
    if S-(M//4) < 0:
        start = S
    else:
        start = S-(M//4)

    if E+(M//4) > N:
        end = N
    else:
        end = E+(M//4)

    K = int(end - start)
    if K%2 == 0: # Make window length odd number
        K += 1
        start += 1 # Signal dimension must equal Window dimension
    wHan = np.hanning(int(K))

    windowed = []
    j = 0
    for k in range(start, end):
        val = normVoice[k]*wHan[j]
        windowed.append(val)
        j += 1
    winTime = np.arange(0, len(windowed), 1)

    # Fourier Transform of windowed voice signal and peak frequency
    FFT, fax = cap.fftprops(windowed, fs)
    FFTN = cap.normalize(FFT)
    faxes.append(fax) #####

    ft = abs(FFT)
    F = len(FFT)
    Y = 10
    rect = np.ones((Y,))
    ftN = np.convolve(ft, rect, mode='same')
    ftN = cap.normalize(ftN)
    FFTs.append(ftN) #####

    peakF = cap.peakFreq(ftN, fax)
    peakF = np.around(peakF)
    Freqs.append(peakF)
print("Peak frequencies from each recorded word:\n", Freqs)

#### Plotting multiple waves on the same plot

```

```

plt.figure(figsize=(10,6))
for g in range(AMOUNT):
    plt.plot(faxes[g], FFTs[g], label=str(g+1))

#### Plot customization
plt.axis([0, 1000, 0, 1])
plt.xticks(np.arange(0,1000,100))
plt.yticks(np.arange(0,1,0.1))
plt.title('Brandon "Back"')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude')
plt.legend()
plt.show()

```

*Python Block 6 - Records the speaker  $N$  amount of times and plots each FFT on top of each other*

```

function [y] = normalize432(x)
check = sum(sum(x));

y = x./max(abs(x));

if check==0 % Is Matrix all 0s?
    y = []
end

end

```

*MATLAB Block 1 - Function to normalize data based on its absolute maximum*

```

function [y, index] = FindSignalStart(x)
N = length(x);
xn = normalize432(x);
xP = (xn).^2; % power signal

M = 1000;
rect = ones(M,1);
xP = conv(xP, rect, 'same');
xP = normalize432(xP);
Threshold = 0.08;

index = find(xP > Threshold, 1);

y = x([index : end]);

end

```

*MATLAB Block 2 - Function to find the "beginning" of speech*

```

function [y, index] = FindSignalEnd(x)

x_flip = flipud(x); % Change [start:end] to [end:start]

% Calcs end because start is flipped
% y_end is index of final non-zero value in signal
[y_flip, y_end] = FindSignalStart(x_flip);

y_fix = flipud(y_flip); % Flip again to fix orientation

N = length(x);

y = y_fix;
index = N-(y_end); % 8000 - "Start" = End index

end

```

*MATLAB Block 3 - Find the "end" of speech*

```
clc
close all

% audiodevinfo
fs = 48000; % sample rate 48kHz
nbits = 16; % bits per sample
channels = 1; % mono
duration = 2.5; % seconds

% Slow down MATLAB
pause(2)

% Record speaker audio
recObj = audiorecorder(fs, nbits, 1);
disp('Talk now')
recordblocking(recObj, duration);
disp('Done...')

% Get values from the object / make X-axis
data = getaudiodata(recObj);

% Plot to see if signal is worth keeping
plot(data)

% Save the file
% Note: 'filename' needs to change depending on
% where you want to save the wav file.
filename = 'C:\Users\brand\Desktop\noiseRM236.wav';
audiowrite(filename, data, fs);
```

*MATLAB Block 4 - Record audio for future testing and analysis*

```

clc
close all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Python STFT Defaults
% Segment Length = 256
% Window Length = 256 (Hanning)
% Overlap = 128
% FFT Length = 256

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% MATLAB STFT Defaults
% Segment Length = 128
% Window Length = 128 (Hanning)
% Overlap = 96
% FFT Length = 128

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Importing wav files (fs is always 48,000Hz)
fs = 48000;
% Kory Go
[KG1, ~] = audioread('KM_go1.wav');
[KG2, ~] = audioread('KM_go2.wav');
[KG3, ~] = audioread('KM_go3.wav');
[KG4, ~] = audioread('KM_go4.wav');
[KG5, ~] = audioread('KM_go5.wav');
% Kory Right
[KR1, ~] = audioread('KM_right1.wav');
[KR2, ~] = audioread('KM_right2.wav');
[KR3, ~] = audioread('KM_right3.wav');
[KR4, ~] = audioread('KM_right4.wav');
[KR5, ~] = audioread('KM_right5.wav');
% Brandon Go
[BG1, ~] = audioread('BG_go1.wav');
[BG2, ~] = audioread('BG_go2.wav');
[BG3, ~] = audioread('BG_go3.wav');
[BG4, ~] = audioread('BG_go4.wav');
[BG5, ~] = audioread('BG_go5.wav');
[BG6, ~] = audioread('BG_go6.wav');
[BG7, ~] = audioread('BG_go7.wav');
[BG8, ~] = audioread('BG_go8.wav');
[BG9, ~] = audioread('BG_go9.wav');
[BG10, ~] = audioread('BG_go10.wav');
% Brandon Left
[BL1, ~] = audioread('BG_left1.wav');
[BL2, ~] = audioread('BG_left2.wav');
[BL3, ~] = audioread('BG_left3.wav');
[BL4, ~] = audioread('BG_left4.wav');
[BL5, ~] = audioread('BG_left5.wav');
[BL6, ~] = audioread('BG_left6.wav');
[BL7, ~] = audioread('BG_left7.wav');
[BL8, ~] = audioread('BG_left8.wav');
[BL9, ~] = audioread('BG_left9.wav');
[BL10, ~] = audioread('BG_left10.wav');
% Brandon Back
[BB1, ~] = audioread('BG_back1.wav');
[BB2, ~] = audioread('BG_back2.wav');
[BB3, ~] = audioread('BG_back3.wav');
[BB4, ~] = audioread('BG_back4.wav');
[BB5, ~] = audioread('BG_back5.wav');
[BB6, ~] = audioread('BG_back6.wav');
[BB7, ~] = audioread('BG_back7.wav');
[BB8, ~] = audioread('BG_back8.wav');
[BB9, ~] = audioread('BG_back9.wav');
[BB10, ~] = audioread('BG_back10.wav');
% Brandon Right
[BR1, ~] = audioread('BG_right1.wav');
[BR2, ~] = audioread('BG_right2.wav');
[BR3, ~] = audioread('BG_right3.wav');
[BR4, ~] = audioread('BG_right4.wav');
[BR5, ~] = audioread('BG_right5.wav');
[BR6, ~] = audioread('BG_right6.wav');
[BR7, ~] = audioread('BG_right7.wav');
[BR8, ~] = audioread('BG_right8.wav');
[BR9, ~] = audioread('BG_right9.wav');
[BR10, ~] = audioread('BG_right10.wav');
% Brandon Go, Right, Back, Left
[BA1, ~] = audioread('BG_all_GRLB1.wav');
[BA2, ~] = audioread('BG_all_GRLB2.wav');
[BA3, ~] = audioread('BG_all_GRLB3.wav');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Cropping signals and using STFT
buff = 2048; % arbitrary number
one = BG1;
two = BG2;
three = BG3;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Crop noise out of signals
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Downsample to 16kHz
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Do STFT and plot spectrogram
[xR1, sR1] = FindSignalStart(one);
[yR1, eR1] = FindSignalEnd(one);
SR1 = sR1-buff;

```

```

ER1 = eR1+buff;
new1 = one([SR1:ER1]);
test = downsample(new1,3);
figure
stft(test,16000)
title('one')

[xR2, sR2] = FindSignalStart(two);
[yR2, eR2] = FindSignalEnd(two);
SR2 = sR2-buff;
ER2 = eR2+buff;
new2 = two([SR2:ER2]);
figure
stft(new2,fs)
title('two')

[xR3, sR3] = FindSignalStart(three);
[yR3, eR3] = FindSignalEnd(three);
SR3 = sR3-buff;
ER3 = eR3+buff;
new3 = three([SR3:ER3]);
figure
stft(new3,fs)
title('three')

%%%%%%%% Take max of STFT along time axis
%%%%%%%% Find peaks along the new maximum vector
%%%%%%%% Print peak frequencies
%%%%%%%% Idea recommendation from Prof. Mario Bkassiny
[s, f, t] = stft(new2,fs);
sv = max(abs(s), [], 2);
figure
plot(f,sv)
[pks, locs] = findpeaks(sv);
f(locs)

%%%%%%%% Plotting STFTs under different parameters %%%%%%%%%
%%%%%%%%

%%%%%%%% Brandon Right, Python Parameters
% figure
% stft(BR1,fs, 'Window', hann(256,'periodic'),...
%   'OverlapLength', 128, 'FFTLenght', 256)
% figure
% stft(BR2,fs, 'Window', hann(256,'periodic'),...
%   'OverlapLength', 128, 'FFTLenght', 256)
% figure
% stft(BR3,fs, 'Window', hann(256,'periodic'),...
%   'OverlapLength', 128, 'FFTLenght', 256)
% figure
% stft(BR4,fs, 'Window', hann(256,'periodic'),...
%   'OverlapLength', 128, 'FFTLenght', 256)
% figure
% stft(BR5,fs, 'Window', hann(256,'periodic'),...
%   'OverlapLength', 128, 'FFTLenght', 256)
% figure
% stft(BR6,fs, 'Window', hann(256,'periodic'),...
%   'OverlapLength', 128, 'FFTLenght', 256)
% figure
% stft(BR7,fs, 'Window', hann(256,'periodic'),...
%   'OverlapLength', 128, 'FFTLenght', 256)
% figure
% stft(BR8,fs, 'Window', hann(256,'periodic'),...
%   'OverlapLength', 128, 'FFTLenght', 256)
% figure
% stft(BR9,fs, 'Window', hann(256,'periodic'),...
%   'OverlapLength', 128, 'FFTLenght', 256)
% figure
% stft(BR10,fs, 'Window', hann(256,'periodic'),...
%   'OverlapLength', 128, 'FFTLenght', 256)

%%%%%%%% Comparing Brandon saying (Go, Right Left, Back) using default MATLAB
%%%%%%%% STFT values vs. Python's default STFT values noted above
% figure
% stft(BA1,fs)
% title('Brandon: Go,Right,Left,Back - MATLAB Defaults')
% figure
% stft(BA1,fs, 'Window', hann(256,'periodic'), 'OverlapLength', 128, 'FFTLenght', 256)
% title('Brandon: Go,Right,Left,Back - Python Parameters')

%%%%%%%% Changing Overlap Length in the same signal
% figure
% stft(BA1,fs)
% title('Brandon: Go,Right,Left,Back - Default Overlap=96')
% figure
% stft(BA1,fs, 'OverlapLength', 8)
% title('Brandon: Go,Right,Left,Back - Custom Overlap=8')

%%%%%%%% Comparing Brandon "Go" (BG4) with

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% MATLAB defaults vs Python Defaults
% figure
% stft(BG4, fs)
% title('BG4 MATLAB Defaults')
% figure
% stft(BG4, fs, 'Window', hann(256,'periodic'), 'OverlapLength', 128, 'FFTLength', 256)
% title('BG4 Python Defaults')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Comparing Brandon and Stephanie saying
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Go, Right, Left, Back with MATLAB Defaults
% figure
% stft(BA1,fs)
% title('Brandon: Go,Right,Left,Back')
% figure
% stft(SA1,fs)
% title('Stephanie: Go,Right,Left,Back')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Marianne Go
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% figure
% stft(MG1,fs)
% title('Marianne, Go')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Stephanie Go, Right, Left, Back
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% figure
% stft(SA1,fs)
% title('Stephanie Go, Right, Left, Back')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Brandon Go, Right, Left, Back
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% figure
% stft(BA1,fs)
% title('Brandon Go, Right, Left, Back (1)')
%
% figure
% stft(BA2,fs)
% title('Brandon Go, Right, Left, Back (2)')
%
% figure
% stft(BA3,fs)
% title('Brandon Go, Right, Left, Back (3)')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Brandon Go's
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% figure
% stft(BG1,fs)
% title('BG1')
%
% figure
% stft(BG2,fs)
% title('BG2')
%
% figure
% stft(BG3,fs)
% title('BG3')
%
% figure
% stft(BG4,fs)
% title('BG4')
%
% figure
% stft(BG5,fs)
% title('BG5')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Brandon Right's
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% figure
% stft(BR1,fs)
% title('BR1')
%
% figure
% stft(BR2,fs)
% title('BR2')
%
% figure
% stft(BR3,fs)
% title('BR3')
%
% figure
% stft(BR4,fs)
% title('BR4')
%
% figure
% stft(BR5,fs)
% title('BR5')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Kory Go's
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% figure
% stft(KG1,fs)
% title('KG1')
%
% figure

```



```

% stft(KG2,fs)
% title('KG2')
%
% figure
% stft(KG3,fs)
% title('KG3')
%
% figure
% stft(KG4,fs)
% title('KG4')
%
% figure
% stft(KG5,fs)
% title('KG5')

%%%%%%%%%% Kory Right's
% figure
% stft(KR1,fs)
% title('KR1')

% figure
% stft(KR2,fs)
% title('KR2')

% figure
% stft(KR3,fs)
% title('KR3')

% figure
% stft(KR4,fs)
% title('KR4')

% figure
% stft(KR5,fs)
% title('KR5')

%%%%%%%%%% Lowpass Filter (fc=5kHz) Example: Brandon "Go"
% [bg4F, dF] = lowpass(BG4, 5000, fs);
% figure
% stft(bg4F,fs)
% title('BG4 LP Filtered')

```

*MATLAB Block 5 - Calculating STFTs with various parameters and audio samples*

```

close all
clc

fs = 48000;
ds = 16000;

word = BG6;
word = normalize432(word);
newDS = downsample(word,3);
[STFT1, F, T] = stft(newDS, ds);
STFT1 = abs(STFT1);
word2 = BB10;
word2 = normalize432(word2);
newDS2 = downsample(word2,3);
[STFT2, F, T] = stft(newDS2, ds);
STFT2 = abs(STFT2);

chunk1 = STFT1(72:73, 1:length(T));
sv1 = max(chunk1, [], 1); % freq=1 / time=2

chunk2 = STFT1(74:75, 1:length(T));
sv2 = max(chunk2, [], 1); % freq=1 / time=2

figure
plot(T, sv1)
title('Top')
figure
plot(T, sv2)
title('Bottom')
figure('Position', [50 50 500 300])
stft(newDS, ds)
title('Left')
figure('Position', [50 50 500 300])
stft(newDS2, ds)
title('Back')

% % word3 = BL2;
% % word3 = normalize432(word3);
% % newDS3 = downsample(word3,3);

% % word4 = BB9;
% % word4 = normalize432(word4);
% % newDS4 = downsample(word4,3);
%
% [STFT1, F1, T1] = stft(newDS, ds);
% cut1 = STFT1(72:76 , 1:length(T1));
% cut1 = abs(cut1);
% GO = sum(sum(cut1));
%
% [STFT2, F2, T2] = stft(newDS2, ds);
% cut2 = STFT2(72:76 , 1:length(T2));
% cut2 = abs(cut2);
% RIGHT = sum(sum(cut2));
%
% [STFT3, F3, T3] = stft(newDS3, ds);
% cut3 = STFT3(64:68 , 1:length(T3));
% cut3 = abs(cut3);
% LEFT = sum(sum(cut3));
%
% [STFT4, F4, T4] = stft(newDS4, ds);
% cut4 = STFT4(64:68 , 1:length(T4));
% cut4 = abs(cut4);
% BACK = sum(sum(cut4));
%
% figure('Position', [50 50 500 300])
% stft(newDS, ds)
% title('Left')
%
% figure('Position', [50 50 500 300])
% stft(newDS2, ds)
% title('Back')
%
% figure('Position', [50 50 500 300])
% stft(newDS3, ds)
% title('Left')
%
% figure('Position', [50 50 500 300])
% stft(newDS, ds)
% title('Back')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% Always Listening idek anymore
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% pause(2)
% recObj = audiorecorder(fs, 16, 1);
% disp('Talk now')
% recordblocking(recObj, 2);
% disp('Done recording...')

```

```

% data = getaudiodata(recObj);

% word = data;
% word = normalize432(word);
% newDS = decimate(word, 3);

% figure('Position', [50 50 750 450])
% stft(newDS, ds)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Looking at "Dixie"
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% [data, ~] = audioread('BG_go4.wav');
% word = data;
% word = normalize432(word);
% newDS = downsample(word,3);
% [STFT1, F1, T1] = stft(word, fs);
% STFT1 = abs(STFT1);
% SV1 = max(abs(STFT1), [], 2);
% [~, locs1] = findpeaks(SV1);
% maxes1 = F1(locs1);
% maxes1 = maxes1(maxes1>=0);
% maxes1 = maxes1(1:5);
%
% [data2, ~] = audioread('BG_right2.wav');
% word2 = data2;
% word2 = normalize432(word2);
% newDS2 = downsample(word2,3);
% [STFT2, F2, T2] = stft(word2, fs);
% STFT2 = abs(STFT2);
% SV2 = max(abs(STFT2), [], 2);
% [~, locs2] = findpeaks(SV2);
% maxes2 = F2(locs2);
% maxes2 = maxes2(maxes2>=0);
% maxes2 = maxes2(1:5);
%
% figure
% stft(newDS, ds)
% figure
% stft(newDS2, ds)

% [data3, ~] = audioread('BG_dixie3.wav');
% word3 = data3;
% word3 = normalize432(word3);
% newDS3 = downsample(word3,3);
% [STFT3, F3, T3] = stft(word3, fs);
% SV3 = max(abs(STFT3), [], 2);
% [~, locs3] = findpeaks(SV3);
% maxes3 = F3(locs3);
% maxes3 = maxes3(maxes3>=0);
% maxes3 = maxes3(1:10)
%
% figure
% stft(word, fs)
% figure
% stft(word2, fs)
% figure
% stft(word3, fs)

```

### MATLAB Block 6 - Examining Spectrograms and STFTs analytically

```

function y = ZCR(x)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Function to calculate the Zero Crossing Rate of a signal
y = sum(abs(diff(x>0)))/length(x);
end

```

### MATLAB Block 7 - Zero Crossing Rate function

```

close all
clc

%%%%%%%% Importing wav files (fs is always 48,000Hz)
fs = 48000;

%%%%%%%% Brandon Go1, Right1, Left1, Back1
% Go
Go = normalize432(BG1); %3600 28000
test1 = Go([3600:28000]);
noise1 = ZCR(test1)
[~, gS] = FindSignalStart(Go);
[~, gE] = FindSignalEnd(Go);
Go = Go( [ gS : gE ] );
figure
plot(Go)
title('Go')

% Right
Right = normalize432(BR1); % 1600 28000
test2 = Right([1600:28000]);
noise2 = ZCR(test2)
[~, rS] = FindSignalStart(Right);
[~, rE] = FindSignalEnd(Right);
Right = Right( [ rS : rE ] );
figure
plot(Right)
title('Right')

% Left
Left = normalize432(BL1); % 2000 29000
test3 = Left([2000:29000]);
noise3 = ZCR(test3)
[~, lS] = FindSignalStart(Left);
[~, lE] = FindSignalEnd(Left);
Left = Left( [ lS : lE ] );
figure
plot(Left)
title('Left')

% Back
Back = normalize432(BB1); % 500 21000
test4 = Back([500:21000]);
noise4 = ZCR(test4)
[~, bS] = FindSignalStart(Back);
[~, bE] = FindSignalEnd(Back);
Back = Back( [ bS : bE ] );
figure
plot(Back)
title('Back')

% Zero Crossing Rates of words without
% beginning and end noise
goZCR = ZCR(Go)
rightZCR = ZCR(Right)
leftZCR = ZCR(Left)
backZCR = ZCR(Back)

%%%%%%%% Results:
%%%%%%%% Go, Right, Left, Back
% goZCR =
% 0.0238
% rightZCR =
% 0.0255
% leftZCR =
% 0.0287
% backZCR =
% 0.0378

```

*MATLAB Block 8 - Using ZCR on speech vs. non-speech*

```

clc
close all

fs = 48000;

%%%%%% Compute Cepstrum and find
%%%%%% fundamental frequency from its peak
n1 = normalize432(BR3);
[~, S] = FindSignalStart(n1);
[~, E] = FindSignalEnd(n1);
M = floor((E-S)/10);
new = n1([ S-M : E+M ]);
N = length(new);

Y = fft(new.*hann(N));
C = fft( log(abs(Y) + eps));
maxim = 5000*length(Y)/fs;
f = (0:maxim)*fs/length(Y);
Y = Y(1:length(f));

ms1 = fs/1000; % 1000Hz (48)
ms20 = fs/50; % 50Hz (960)
q = (ms1:ms20)/fs;
C = abs( C(ms1:ms20) );

% Plot of the Fourier Transform
figure
plot(f, abs(Y))
title('FFT')
% Plot of the Cepstrum
figure
plot(q, C)
title('Cepstrum')

% Cepstrum max quefrequency and frequency conversion
[c, fx] = max(C)
Peak = fs / (ms1+fx-1)

```

*MATLAB Block 9 - Analyzing the Cepstrum and Quefrequencies*

```

clc
close all

fs = 48000;

%%%%%% Plots continuous wavelet transform of 4 signals
Norm1 = normalize432(BG1);
figure
cwt(Norm1,fs)

Norm2 = normalize432(BG3);
figure
cwt(Norm2,fs)

Norm3 = normalize432(BG8);
figure
cwt(Norm3,fs)

Norm4 = normalize432(BG7);
figure
cwt(Norm4,fs)

```

*MATLAB Block 10 - Analyzing the Continuous Wavelet Transform*

```

//ui8Adjust will be the number we have to change to get to 1ms, 1.5ms, and 2ms.

#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/debug.h"
#include "driverlib/pwm.h"
#include "driverlib/pin_map.h"
#include "inc/hw_gpio.h"
#include "driverlib/rom.h"
#include "inc/hw_ints.h"
#include "inc/hw_types.h"
#include "driverlib/interrupt.h"
#include "driverlib/uart.h"

// 55Hz as base frequency to control the servo
#define PWM_FREQUENCY 55

volatile uint8_t ui8Adjust;
volatile uint8_t ui8AdjustL;
volatile uint8_t ui8AdjustR;
volatile uint32_t ui32Load;
volatile uint32_t ui32PWMClock;

void EnUART()
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

    GPIOPinConfigure(GPIO_PB0_U1RX);
    GPIOPinConfigure(GPIO_PB1_U1TX);
    GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 9600,
        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
}

void motors(void)
{
    // 83 is the center position to create a 1.5mS pulse
    ui8Adjust = 83;

    // Divides the clock by 64 to run PWM clock at 625kHz
    SysCtlPWMClockSet(SYSCTL_PWMDIV_64);
    // Enable PWM 1
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
    // Enable Port D for PWM output
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    // Port D pin 0 (PD0)
    GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_1);
    GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0);
    GPIOPinConfigure(GPIO_PD1_M1PWM1);
    GPIOPinConfigure(GPIO_PD0_M1PWM0);

    /*-----*/

    ui32PWMClock = SysCtlClockGet() / 64;
    // Divide PWM clock by desired frequency to get the count
    // Subtract one because down counter counts down to 0
    ui32Load = (ui32PWMClock / PWM_FREQUENCY) - 1;
    // Configure PWM1 generator 0 as a down counter
    PWMGenConfigure(PWM1_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
    // Load the count value
    PWMGenPeriodSet(PWM1_BASE, PWM_GEN_0, ui32Load);
    // Sets the pulse width
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_1, ui8Adjust * ui32Load / 1000);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
    // PWM module 1 generator 0 enabled as an output
    PWMOutputState(PWM1_BASE, PWM_OUT_1_BIT, true);
    PWMOutputState(PWM1_BASE, PWM_OUT_0_BIT, true);
    // PWM module 1 generator 0 enabled to run
    PWMGenEnable(PWM1_BASE, PWM_GEN_0);
}

void goStraight()
{
    motors();
    ui8Adjust = 75;
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_1, ui8Adjust * ui32Load / 1000);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
}

```

```

void stop()
{
    motors();
    ui8Adjust = 83;
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_1, ui8Adjust * ui32Load / 1000);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
}

void goBack()
{
    motors();
    ui8Adjust = 90;
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_1, ui8Adjust * ui32Load / 1000);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
}

void turnRight()
{
    motors();
    ui8AdjustL = 92;
    ui8AdjustR = 74;
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_1, ui8AdjustR * ui32Load / 1000);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8AdjustL * ui32Load / 1000);
}

void turnLeft()
{
    motors();
    ui8AdjustL = 74;
    ui8AdjustR = 92;
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_1, ui8AdjustR * ui32Load / 1000);
    PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8AdjustL * ui32Load / 1000);
}

void UARTInput(void)
{
    char command = UARTCharGet(UART1_BASE);
    if (command == 'g')
    {
        goStraight();
        SysCtlDelay(20000000);
        stop();
        UARTCharPut(UART1_BASE, 'x'); // echo character on terminal
    }
    else if (command == 'r')
    {
        turnRight();
        SysCtlDelay(21000000);
        stop();
        UARTCharPut(UART1_BASE, command); // echo character
    }
    else if (command == 'l')
    {
        turnLeft();
        SysCtlDelay(21000000);
        stop();
        UARTCharPut(UART1_BASE, command); // echo character
    }
    else if (command == 'b')
    {
        goBack();
        SysCtlDelay(20000000);
        stop();
        UARTCharPut(UART1_BASE, 'y'); // echo character
    }
    else if (command == 's')
    {
        stop();
        UARTCharPut(UART1_BASE, command); // echo character
    }
}

void main(void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);
    EnUART();

    while (1)
    {
        UARTInput();

        while (UARTCharsAvail(UART1_BASE)){
            UARTCharGet(UART1_BASE);
        }
    }
}

```

*Embedded C Block 1 - Robot code for movement and XBee usage*

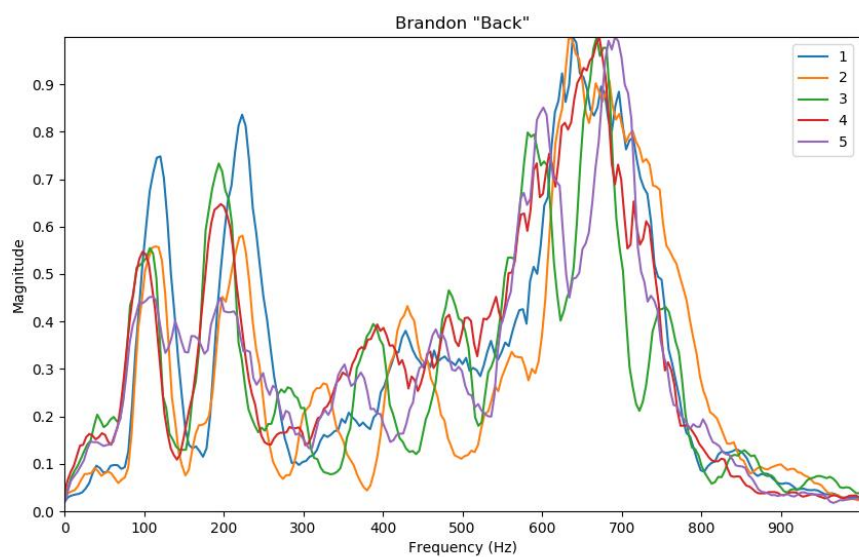


Figure 16 - Fourier Transform of Brandon saying "Back" 5x

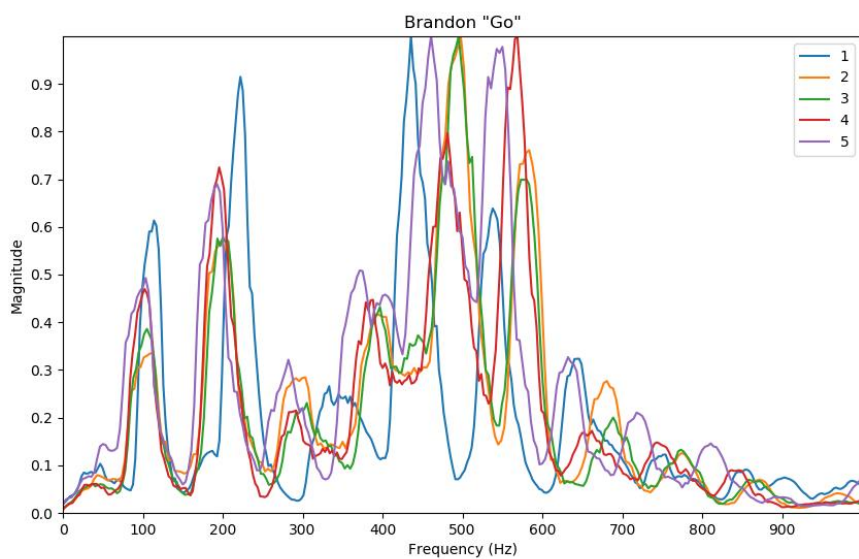


Figure 17 - Fourier Transform of Brandon saying "Go" 5x



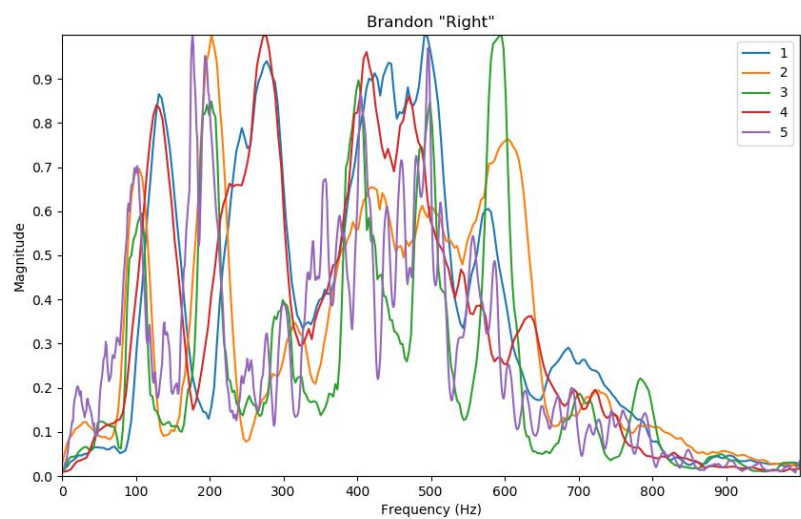


Figure 18 - Fourier Transform of Brandon saying "Right" 5x

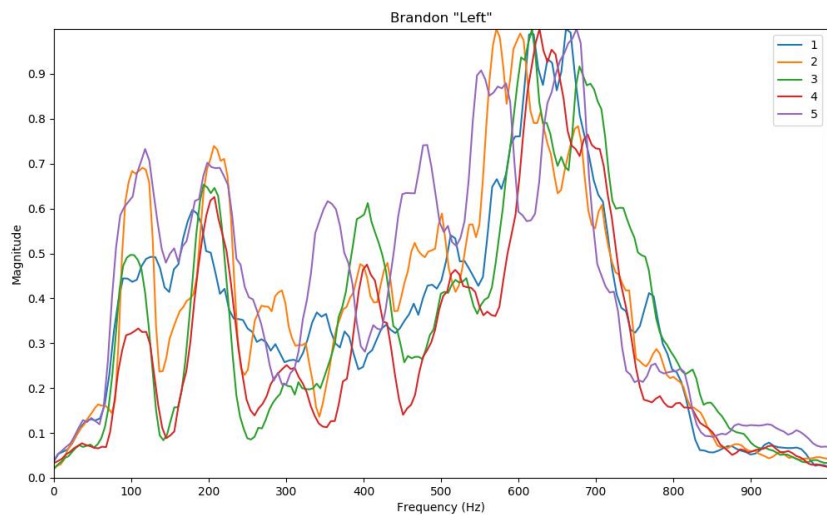


Figure 19 - Fourier Transform of Brandon saying "Left" 5x

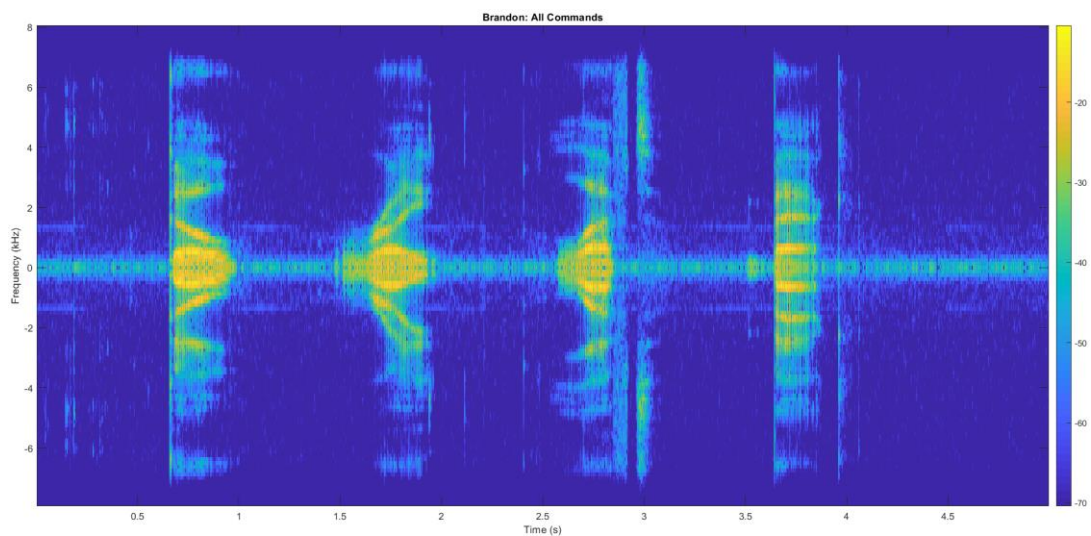


Figure 20 - Spectrogram of Brandon saying all four commands

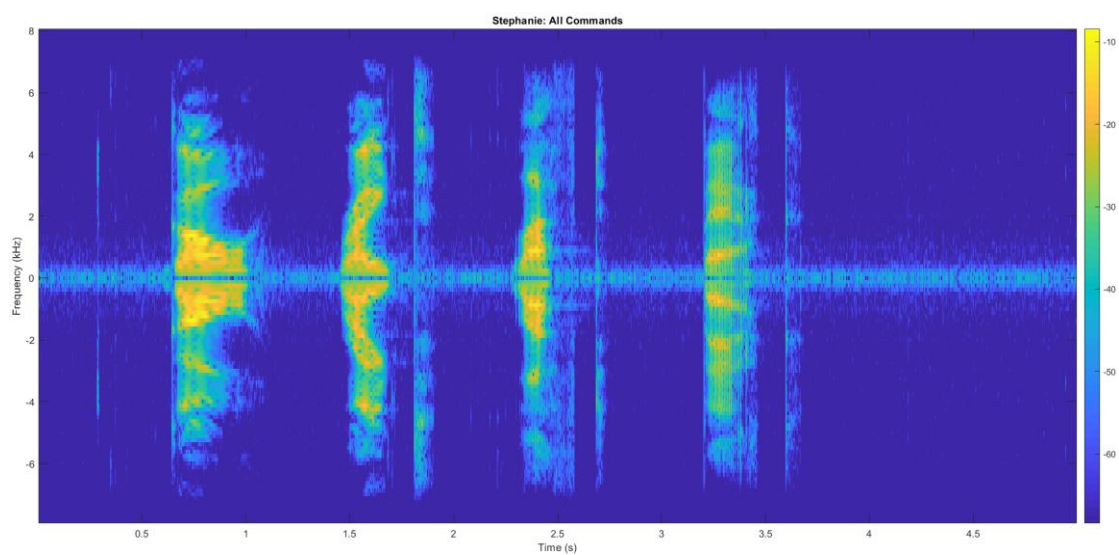


Figure 21 - Spectrogram of Stephanie saying all four commands

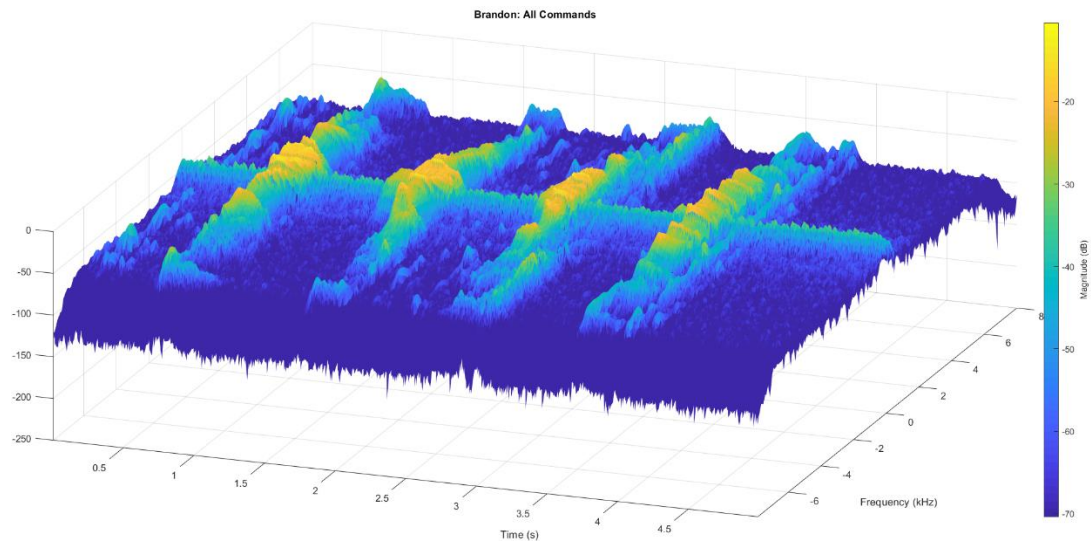


Figure 22 - 3D representation of spectrogram

## Appendix C

### *XBee Documentation*

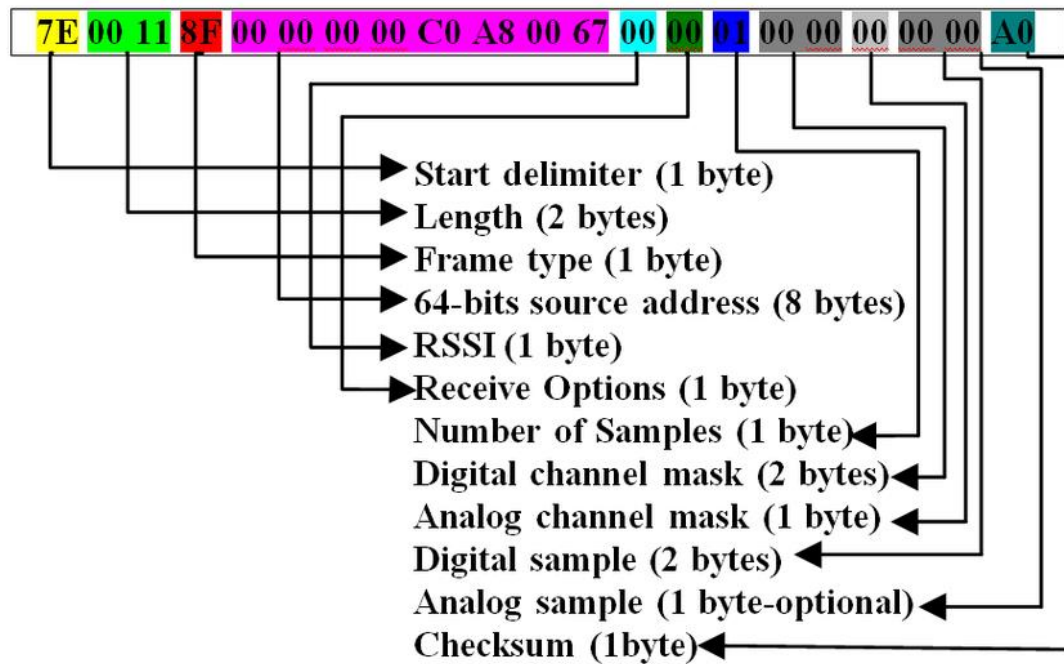


Figure 23 - Frame formatting for XBee Modules

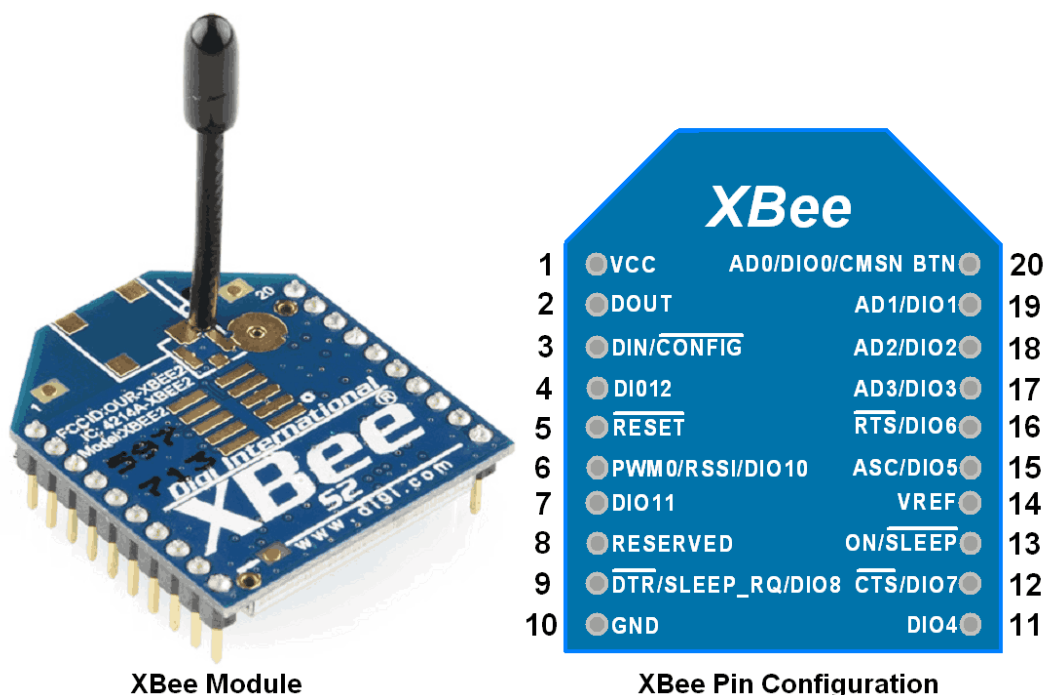


Figure 24 - XBee I/O pin layout

Table 1-02. Pin Assignments for the XBee and XBee-PRO Modules  
(Low-asserted signals are distinguished with a horizontal line above signal name.)

Pin #	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART Data Out
3	DIN / $\overline{\text{CONFIG}}$	Input	UART Data In
4	DIO8*	Output	Digital Output 8
5	$\overline{\text{RESET}}$	Input	Module Reset (reset pulse must be at least 200 ns)
6	PWM0 / RSSI	Output	PWM Output 0 / RX Signal Strength Indicator
7	PWM1	Output	PWM Output 1
8	[reserved]	-	Do not connect
9	$\overline{\text{DTR}}$ / SLEEP_RQ / DIO8	Input	Pin Sleep Control Line or Digital Input 8
10	GND	-	Ground
11	AD4 / DIO4	Either	Analog Input 4 or Digital I/O 4
12	CTS / DIO7	Either	Clear-to-Send Flow Control or Digital I/O 7
13	ON / $\overline{\text{SLEEP}}$	Output	Module Status Indicator
14	VREF	Input	Voltage Reference for A/D Inputs
15	Associate / AD5 / DIO5	Either	Associated Indicator, Analog Input 5 or Digital I/O 5
16	RTS / AD6 / DIO6	Either	Request-to-Send Flow Control, Analog Input 6 or Digital I/O 6
17	AD3 / DIO3	Either	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Either	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Either	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0	Either	Analog Input 0 or Digital I/O 0

Figure 25 - XBee I/O descriptions