UNIVERSITY OF CALIFORNIA SAN DIEGO

A Networked Solution for Robotic Sound Source Localization

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Music

by

Colin Zyskowski

Committee in charge:

Miller Puckette, Chair
Mauricio de Oliviera, Co-Chair
Tom Erbe
David Kirsh
Tamara Smyth

2018

The dissertation of Colin Zyskowski is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____
Co-Chair

_____
Chair

University of California San Diego

2018

DEDICATION

To Doug and Dianne Zyskowski, whose boundless support made this possible.

EPIGRAPH

Personally I'm not afraid of a robot uprising. The benefits far outweigh the threats

—Daniel H. Wilson

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

The greatest of thanks to Mauricio de Oliveira, who spent more time on this project than I ever could have asked. And to Miller Puckette, whose patience and support were boundless. And to Giulio Moro, who put up with my endless questions and bugs.

VITA

2000	B. A. in English Literature, The University of Michigan
2010	M. A. in Media Arts, The University of Michigan
2018	Ph. D. in Music, University of California, San Diego

PUBLICATIONS

Colin Zyskowski, J. Cantrell “The Breath Engine: Challenging Biological and Technological Boundaries through the Use of NK Complex Adaptive Systems. In: Sound and Music Computing Conference (ICMC —SMC). Athens,Greece, 2014.

Colin Zyskowski, Mauricio de Oliveira A Robotics Platform for Musical Performance. Southern California Robotics Symposium, La Jolla, CA, 2016.

Colin Zyskowski, Mauricio de Oliveira ”An Analog Audio Sensor Board for Microcontrollers.” AES, 142nd International Conference, Proceedings, Berlin, Germany 2017.

Colin Zyskowski, Mauricio de Oliveira ”A Networked System for Interactive Robotic Musical Performance.” Diffrazioni Multimedia Festival, Florence, Italy, November, 2016.

ABSTRACT OF THE DISSERTATION

A Networked Solution for Robotic Sound Source Localization

by

Colin Zyskowski

Doctor of Philosophy in Music

University of California San Diego, 2018

Miller Puckette, Chair
Mauricio de Oliviera, Co-Chair

This dissertation proposes a new method for robotic sound source localization that relies upon a series of networked robots. Common methods of sound source localization will be discussed in the first chapter. The second chapter will cover the design and construction of the robots used in this project, focusing on hardware design and implementation. Chapter three will detail the new methodology for sound source localization proposed in this project, focusing on two separate methods of locational determination and resulting in a comparison of those two methods. The final chapter will present conclusions arrived at through the process of building and testing this system as well as provide direction for future work in this area.

Introduction

The project outlined in the paper is based on a combination of various ideas that have interested me for a number of years. It is my hope that these interests have found a suitable synthesis through the completion of this project. Each idea has proved an ample field for exploration and study, posing its own difficulties and demanding unique solutions. Many of the obstacles encountered throughout the development of this project have been omitted from this writing. The problems that are presented and their solutions are ones that will hopefully prove useful to future students and researchers.

The outcome of this project is a unique solution to the problem of robotic sound source localization. In order to complete a project based on this idea, it was necessary to research several topics. The main focal points were:

- *Audio capture and analysis* - A unique system for audio capture and analysis had to be developed, a process which involved the design and construction of custom circuit boards as well as custom objects for Pure Data. These devices handle basic functions such as amplification as well as signal processing related to envelope detection and cross-correlation.

- *Microcontrollers and embedded systems design* - This project relies on a microprocessor running a real-time version of Linux. Custom changes to the OS had to be made in order to accommodate necessary hardware input/output functionality, which necessitated a deep study of Linux and embedded systems architecture.

- *Circuit and hardware design* - The custom circuit boards used for this project were designed specifically for their applications here. These include hardware interfaces for audio input and analysis as well as motor control and sensor input. Designing and building these devices required an understanding of electrical engineering as well as circuit board construction.

- *Control and feedback systems* - Control systems play an important role in this project due to its reliance on sensor input for motion control. Thus, a system a system was developed that includes feedback from microphones, encoder readers, and motion processing units. This development involved a study of various peripheral ICs, many of which found their way into the final project.

- *Wireless networking and connectivity* - One of the primary characteristics of this project is its reliance upon networked connectivity. The robots communicate over a series of networks that were built in Python. To develop multiple reliable and robust networks, various approaches to networking in Python were investigated, as were different options for networking hardware.

- *Programming for interactive applications* - This project includes output in the form of a graphical mapping system that displays the robots' positional information. The programs running on the robots themselves also allow for user input to control various aspects of the robots localization routines. The main areas of exploration for the development of these systems were Python game and interface design.

Most of these topics are ones with which I was largely unfamiliar when I began my graduate studies; some of them were wholly foreign to me. As my ideas and plans developed and problems accrued, I found myself delving into research realms that were not necessarily linearly related to common topics of computer music. However, I wandered down these paths for two reasons: 1) solving problems with low-level solutions - for example, building my own circuit

boards - was, in most cases, far less expensive than procuring pre-made products. 2) Finding low-level work-arounds to the obstacles I encountered proved a great learning experience.

Thus it was that, while in music school, I found myself learning 3D design software to build parts for autonomous robots, or in the maker studio huddled over the laser cutter as I fashioned new components for their chassis. This same pattern often found me, soldering iron in hand, building voltage divider circuitry and motor driver hardware that required a working knowledge of integrated circuits and a passing ability to read a schematic.

However, the project's original inception and intent were entirely musical; I had at first sought to create an autonomous system that would play streamed audio. The various streaming audio channels were to be individual tracks of a larger live electronic performance controlled by one musician. The effect would be a type of dynamic spatialization using a multi-speaker array. I intended to highlight the interaction that would take place between the robots and the audience as well as amongst the robots themselves. As I began working towards this goal, the performative and musical aspects were overwhelmed by the more physical endeavors related to hardware: circuit design, control systems, sensor I/O and serial communication protocols. While I had been creating and performing music for many years, these other areas were entirely new directions for me. There were so many rabbit holes and winding paths to follow. In the end, I focused on the unfamiliar territories rather than the roads I had been down before. Instead of focusing on musical output, I decided to work with the flip side of that coin and develop a listening system.

The choice to build an input- rather than output-based system was, I suppose, also influenced by my interest in audio recording and engineering and the studio work I was doing when I began this project. This was, coincidentally, the same time that a new family of ARM-based microcontrollers saw their first serious audio capabilities evolve, which would lead me down a related and even more labyrinthine path towards the development of a multi-track recording system for microcontrollers. This resulted in the creation of multichannel audio cards for the Raspberry Pi and Teensy families of microcontrollers. While they weren't used for this project,

my interests in audio recording and recording systems certainly guided my inevitable direction.

My eventual choice to focus on sound source localization also had to do with problems in need of solutions. While certainly an area I had a great deal of interest in, I did not view robotic musical performance as a research topic that presented immediate shortcomings. There has been a significant amount of work done towards creating emotive mechanical performers, notably the work of researchers such as Weinstein [Bre16] and Birgitta [BB10]. Performance quality and outcomes in such examples are subjective matters. I believed that any work I did in this regard could only be judged in that manner, and not with an objective assessment of goals met, objectives achieved, and solutions obtained. It is a matter of opinion to judge a musical performance, but localization outcomes can be assessed with objective metrics.

The pursuits undertaken throughout the course of this project have not been proven entirely unrelated to topics in computer music and music technology in general. As embedded systems and microcontrollers grow in sophistication and their use becomes more prolific, they appear in music and audio systems more commonly. The ideas I have worked on here have transferred well to audio circuit design for platforms like the Raspberry Pi and Teensy families of microcontrollers. Likewise, my study of motor control and feedback systems has allowed me to develop more sophisticated projects for interactive kinetic sound installations. My study of circuit design has enabled me to produce wearable audio-reactive devices for the hearing impaired. In short, the field of computer music is now such a broad one that seemingly disparate topics find applications in it; my hope is that the project outlined in this paper, being a synthesis of various and at times seemingly unrelated topics, has managed, among other things, to prove that.

My dissertation presents a sound source localization system consisting of three robots. Those robots, RED, GREEN and BLUE, communicate on a wireless network, along with a hub that communicates via UDP socket to all three robots. Each robot runs two programs — an audio input/analysis application and a motion control program. To enable this, there is an ARM-based embedded system as well as numerous custom circuit boards to handle audio and

control functions. There is a very specific procedure that takes place in order for the robots to successfully achieve their goal. It begins with the lead robot (RED) initiating its detection procedures, then communicating its location data to the hub and thus the other two robots. While this is happening, a graphical mapping of the robots' movements takes place. The robots' design, control systems, localization procedures, and graphical mapping are the focus of this work.

This paper begins with a discussion of current practices in robotic sound source localization, focusing on methods that have found their way into the list of commonly accepted localization routines. Also covered are methods of positional determination, as this process plays an important role in the project. The second part of this paper deals with the mechanics involved in the project as a whole, detailing the hardware and functionality of the robots themselves. The third chapter covers the control systems and networking that went into the localization routines. The final chapter is a discussion of the outcome of the project, involving successes, failures, and areas in which future work can be explored.

Chapter 1

A Survey of Common Methods for Robotic Sound Source Localization

This chapter covers work that has already been done in the field of robotic sound source localization (RSSL) in an effort to demonstrate common methods and accepted techniques. While some of these techniques have been incorporated into the project covered later in this paper, the intent of this chapter is to provide a more general background about the overall field and its current state.

1.1 General Information Regarding RSSL

Human-Robot Interaction (HRI) is a growing research field due to the prevalence of industrial-, commercial-, and entertainment-purposed robotics. Foremost in HRI research is robotic audition the process through which a robot listens to its surroundings, locates and tracks sound sources, and completes feature extraction on the input audio. Most complex embedded robotic auditory systems follow a bottom-up framework where a signal is first sensed, then analyzed to estimate source positions. Next, these positions are used to separate sounds of interest.

Finally, sound- or speech-recognition systems deal with the audio to decipher its purpose or meaning. This process shows the important role of sound source localization in robotic auditory systems [Arg15]. As humanoid robots have become more popular, there has been a trend towards creating biologically inspired robot audition systems. The audition system in humans, however, is much more complex than anything currently possible in a robotic system. Humans are able to perceive, locate, and extract sonic information at incredibly rapid rates. The attempt to replicate this quality in robotics requires two key components: embeddable systems, or systems that fit on the robots body; computationally efficient systems, or ones which can process signals effectively at high speeds while still remaining embeddable. There are also several inherent constraints:

Geometry/Embeddability Constraint: The desire to mimic human audition does not necessarily mean that a robot must only have two microphones. Larger arrays of course offer possibilities for greater accuracy in source localization, invoking a constraint in geometric positioning of the array in question. While larger arrays may be advantageous, they must also be able to fit on the robot. This introduces the issue of embeddability.

Real-time Constraint: Computational complexity is a primary concern in robotic audition. In order to achieve something close to real-time functionality, dedicated systems are often required. This again brings up the issue of embeddability. The real-time constraint also implies possible trade offs between computation time and accuracy/functionality. Frequency Constraint The signals of most use in robotic audition are broadband, meaning they exist over a wide range of frequencies. Narrow band approaches to audition imply lower computation time because they have fewer frequency bins to analyze. However, voice signals, for example, typically exist in the range from 300-3300Hz, thus necessitating higher computation times.

Environmental Constraint: Robots often operate in noisy, unpredictable environments and contribute to that noise themselves with motors, fans, and other moving parts. The spaces in which they operate are also prone to less-than-ideal physical characteristics, e.g., reverberation. The environmental constraints can also include things like multiple sound sources, obstacles that

sound bounces off of, or spaces which naturally produce a large amount of white noise. As the complexity of the environment increases, so does the computation time required to overcome such obstacles [Arg15].

The following sections describe different approaches to robotic sound source localization with special regard to the constraints mentioned above. The descriptions begin with the simplest audition systems and proceed to the more complex.

1.2 Human Auditory Systems

In 1907, Lord Rayleigh presented the duplex theory, which proposes that localization on the horizontal plane is performed through two primary cues, Interaural Time Difference (ITD) and Interaural Level Difference (ILD) although ILD is generally referred to as IID today, as in Interaural Intensity Difference. IID relates to the difference in levels as perceived by the right and left ears as a result of frequency-dependent scattering around the head. Noticeably, if a source emitted at a frequency higher than about 750Hz, then the head and any small-sized element of the face induce scattering, which significantly modifies the perceived acoustic levels, so that the ILD can exceed 30dB [Arg15]. Low frequencies, with wavelengths greater than the head diameter, undergo no scattering. The second localization cue, ITD, is a result of the difference in paths that each wave must travel to reach the ears. The maximum value involved in ITD is around 700 microseconds, or one period of a 1400Hz wave. Therefore, two frequency domains can be exhibited in human horizontal localization, each one involving a distinct acoustic cue. Frequencies under 1 kHz are azimuthally localized by means of the IPD, while frequencies above 3 kHz exploit the ILD [Arg15].

Vertical localization in humans is less reliable than horizontal or azimuthal localization. (+/- 1 for horizontal, +/- 5 for vertical [MW04]) Vertical estimation involves a more complicated and less clear combination of cues. Components of the human body such as shoulders, head,

and the outer ear act as scatterers which modify the frequency elements of acoustic waves. The combined effects of surface reflections from these obstacles introduces notches into the perceived spectrum of sound as it is introduced to the human head, all of which are greatly affected by the elevation of the source. Vertical localization is thus more a result of spectral cues and termed monaural, as it involves no signal comparison between the two ears [Arg15].

The third main component to sound source localization, distance estimation, has received the least amount of study because researchers often feel that Direction of Arrival (DOA) is sufficient for decent human-robot interaction to take place. In humans, distance estimation relies on a number of cues as well as a priori knowledge of the environment and sound source. The cues involved can be things like sound intensity, inter-aural differences ,spectral shaping, and the Direct-to-Reverberant sound energy ratio. However, Argentieri reminds us that, human performances in distance discrimination are quite poor. Even under ideal acoustic conditions, the estimated distance appears to be a biased estimate of the actual one, and listening tests have also proven that humans use to significantly overestimate the distance to sources closer than 1m, while they underestimate distances greater than 1m [Arg15].

1.3 Binaural Sound Source Localization

Humanoid (biologically inspired) robots imply a tendency towards binaural localization. Binaural localization generally relies upon Interaural Time Difference (ITD), which relates to the amount of time it takes for sound to reach two separate points, and/or Interaural Intensity Difference (IID), which relates to the difference in loudness between two points. These two phenomena are based on the differences between the inputs of two microphones. For ITD, delays can range from 0 seconds, when a sound is directly in front of the two sensors, or equidistant, to 700 microseconds when the sound is at a 90 degree azimuth from the X axis of the two sensors. In low frequencies, for example below 1,500Hz, ITD is also measured as phase delay between

the right and left inputs [Iri95].

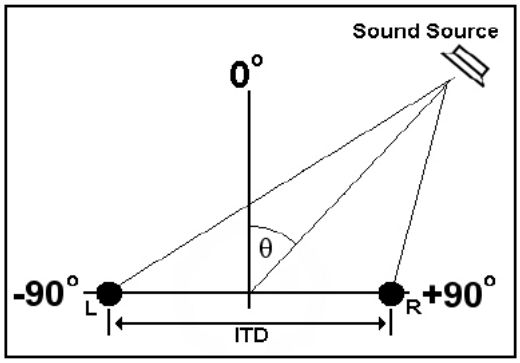


Figure 1.1: InterAural Time Difference.

As stated earlier, human auditory systems do not possess a fine enough resolution to distinguish phase differences in higher frequencies, whereas high sample rates in modern audio equipment allow for signal processing applications to distinguish phase differences on a sample-by-sample basis. Murray et al. used a sample-by-sample cross correlation technique to exploit this capability. Their process involves finding the point at which independently received signals most closely match, then calculating a TDOA based on the input times of those signals.

> In order to determine the angle of incidence of the received wave form, we have to be able to detect the Interaural Phase Difference (IPD), i.e. the lag of the wave at a specific point received at both microphones. When the first microphone detects the sound, we need to ensure that when we are calculating the TDOA, that we compute it between two identical points along the waveform in order to ensure we get an accurate measure of the ITD. To carry out this task on our robot the system records a sample of sound in a time interval (initially this was a one second slice). The stereo signal recorded at the microphones is then split into its left and right components. This is passed to the cross-correlation function which is used to compare the left and right channels for similarity i.e. where the signals are most matched [MW04].

Cross-correlation is thus used to compare two vectors, A and B, for similarity. As the signals are slid across each other, they produce a product vector as shown by, length(C) = (length(A) + length(B)) 1 [MW04].

The human head has the ability to move and compare differences in ITD at various angles, enabling greater accuracy in source determination. However, with regards to a stationary or

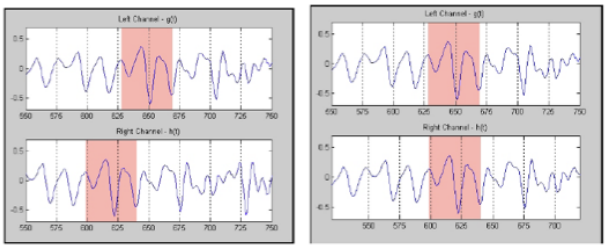


Figure 1.2: Cross-Correlation of Signals.

slow-moving robot, these fast paced motions and calculations are less than second nature. This presents a problem when searching for sound sources, especially in determining whether the sound is in front of or behind the sensor, referred to as the “mirror effect” [MW04].

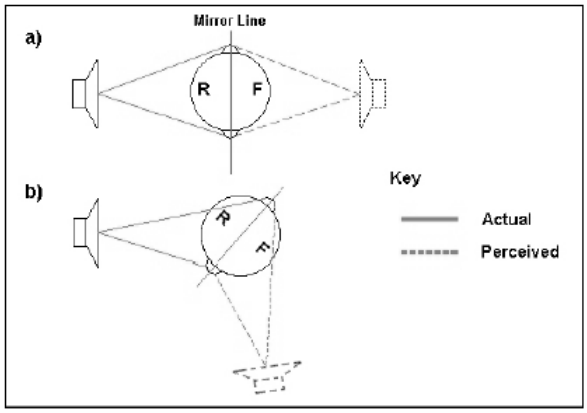


Figure 1.3: The mirror effect is responsible for confusion regarding forward or backward placement of sound sources.

Another pitfall inherent in binaural sound source localization lies in perceiving location along the Z axis. ITD can only be of use on one axis at a time. In other words,it is of no use in determining the height of a sound source. With a microphone pair that is static along the Z axis, bypassing this handicap can be extremely difficult. Some authors have proposed creating artificial pinnae that model the ones in the inner human ear. One model, proposed by Hebrank and Wright [JH74], superimposes the incident wave with a single wave reflected by the pinnae and enables some prediction of source elevation based upon notches that occur in the resulting spectrum analysis. This method is hindered by the fact that these notches are difficult to detect and may be blurred by destructive interference from obstacles [Arg15].

The issue of distance estimation has so far been dealt with through triangulation of the sound source position through two distinct ITD measurements taken at separate intervals. This, of course, is not a real-time process. Rodemann proposed a solution that compared several auditory cues, such as inter-aural differences, amplitudes, and spectral characteristics, but the processing power required was great and the results still had an error of 1 meter for a 6 meter sound source [Arg15]. Thus, binaural sound source localization techniques can achieve decent results in determining DOA in regards to the horizontal azimuth. Attempts to garner significant information regarding vertical axis localization or distance estimation have proven to be less successful and far more complicated. Numerous efforts have been made, though, to extend the robots ability in these areas by adding more microphones, which is where the next section takes us.

1.4 Localization with Arrays

As robots are not restricted to the sensory inputs of humans, it is a seemingly simple matter to increase the number of sensors involved in sound source determination. Adding microphones can increase the processing power necessary for such determination, but with the benefit of greater accuracy and often speed; a higher number of sensors means negation of the "mirror effect" and thus less processing time is needed to account for this. An array also offers the possibility, depending on microphone position, to determine source location upon the Z axis [VL07]

With an array of microphones, given that the space between inputs is great enough, time delay of arrival (TDOA) can be used to asses a sounds source location. TDOA is similar to ITD in that it measures the amount of time necessary for sound to travel from one sensor to the next [VL07].

Lim et al. dealt with the "mirror effect" issue in the simplest way possible adding one microphone. In their three-microphone array, they used a triangular arrangement that not only

provided details as to the front-or-back question, but also gave some insight as to the vertical placement of sound sources[LK07].

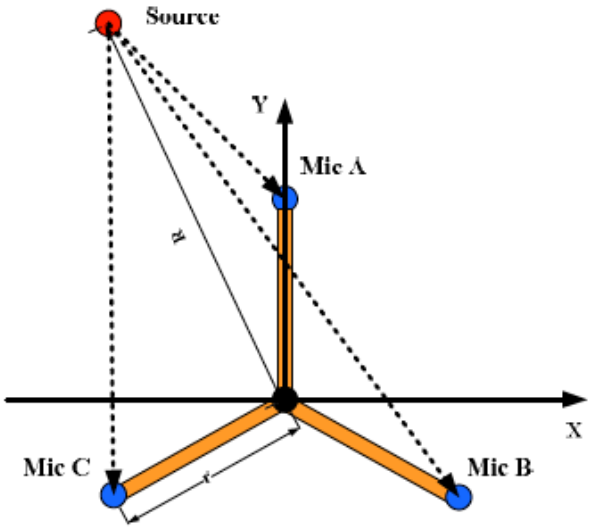


Figure 1.4: Proposed solution to the ”mirror effect” using three microphones.

With this arrangement, they used basic TDOA values between all three microphones to obtain reliable predictions about sound source location [LK07].

1.5 Beamforming

“The basic idea behind the steered beamformer approach to source localization is to direct a beamformer in all possible directions and look for maximal output” [VL07]. Beamforming is a far field localization technique, which means that it best used for situations in which the distance between the sound and the array is greater than the distance between the microphones in the array. It is sometimes called “sum and delay” because it compares the relative delay of sound waves as they reach the microphones in the array. As a rule, beamforming has the two downsides that it cannot be used for frequencies below 1000Hz and it cannot be used to calculate sound power. Various array configurations are possible in beamforming, but there is generally a trade off between dynamic range and accuracy of the array based on the configuration. The

beamforming technique also has the disadvantage that it requires large numbers of channels — often around forty [LJ10].

1.6 General Methods for Position Localization in Robotics

The robotics project that is introduced in this writing employs a method of sound source localization this relies heavily on knowing each robots position in space. This brings about the problem of maintaining accurate and up-to-date location information. Current methods for localizing robots vary depending on resources at hand. These include external detection devices such as global positioning satellites (GPS), infrared light emitters, and cameras [GS]. However, the intention of this project is to allow for each robot to self-localize, that is, to determine its own position in space without the use of external devices.

Odometry, or dead reckoning, is the method of calculating position based on the rotation of a vehicles wheels and is a widely used technique in position localization. Odometry converts a wheels rotational motion to positional motion through the use of an encoder, thus providing an inexpensive (computationally and financially) method for position determination. However, there are several errors inherent in using solely odometry. These errors fall into two classes: systematic and non-systematic. Systematic errors include faults in the robots physical system, such as misalignment of wheels, limited encoder resolution, limited encoder sampling rate, undefined wheelbase due to intermittent contact with the floor, or a difference in wheel diameters. Non-systematic errors can be caused by things like uneven travel surfaces, travel over unexpected objects, or wheel slippage [AH11]. These errors also have the tendency to accumulate over time and distance [SB01].

Several methods for correcting systematic and non-systematic errors have been introduced. In addressing systematic errors, one popular method, called UMBmark, was developed by Borenstein and Feng in 1995. This method attempts to take into account wheelbase and distance-

related errors by running a robot in clockwise and counter clockwise paths, then comparing the expected location and orientation with the actual location/orientation readings. This serves to model the average error of the given system, which can then be compensated for. Improvements were made to the UMBmark method by Chong and Kleeman by adding external an unloaded wheel independently installed on the bearings of each wheel in order to reduce slippage. The results of this technique reduced errors to a zero-mean level, but they limited the robots motion considerably [AH11].

Houshangi and Azizi extended the UMBmark system further by incorporating an inertial gyroscopic system. They used a single axis fiber optic gyroscope, measuring rotational rate of each wheel, to determine the robots orientation. They then incorporated this information with odometry data using an unscented Kalman Filter. The result was a far greater accuracy in location determination than odometry, inertial systems, or the UMBmark methods alone [HA05].

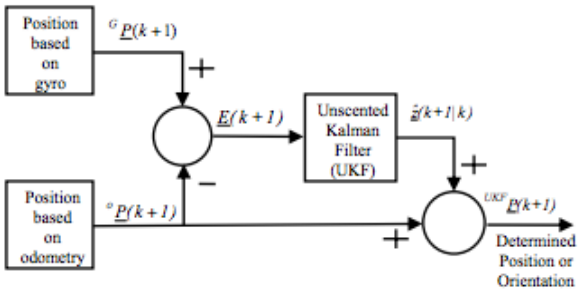


Figure 1.5: The Kalman filtering process used in the UMBmark algorithm.

Another method of positional localization that should be discussed is relative positioning and orientation using binaural sensors between multiple robots. Shoval and Borenstein equipped robots with ultrasonic sensors (sonar) and radio transmitters for communication between two robots and were able to obtain a decent accuracy at a low cost and with fairly simple operation. This technique uses the ultrasonic transmitters to send a pulsed signal away from itself, which then bounces back and is received. The time of flight (TOF) for the signal is then calculated. As the TOF changes amongst the sensors, the robots positions relative to each other can be determined [SB01]. This method is mentioned in this context because the project at hand requires multiple

robots whose relative position will be of great importance.

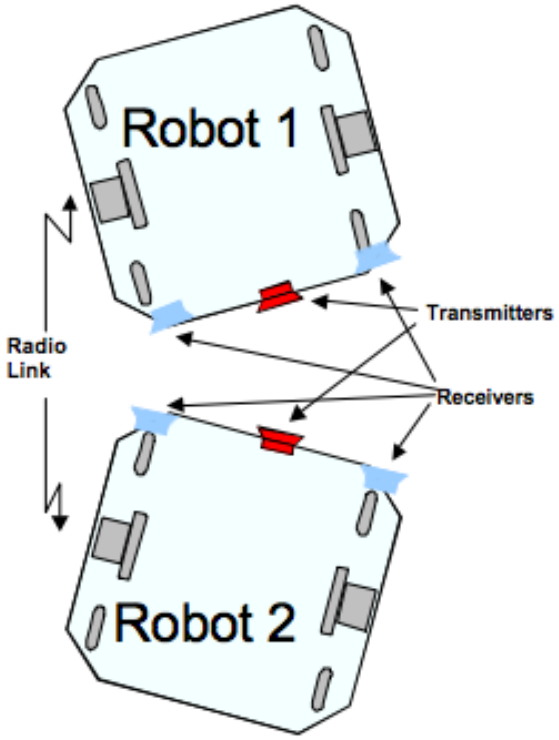


Figure 1.6: Measuring the ”time of flight” of signals between multiple robots.

1.7 A Networked Method

This concludes the discussion of previous robotic sound source localization methods. The remainder of this paper will cover attempts to develop a new method of robotic sound source localization that uses various and combined processes from some of the previously discussed procedures.

Chapter 2

Robotic Hardware and Software Design

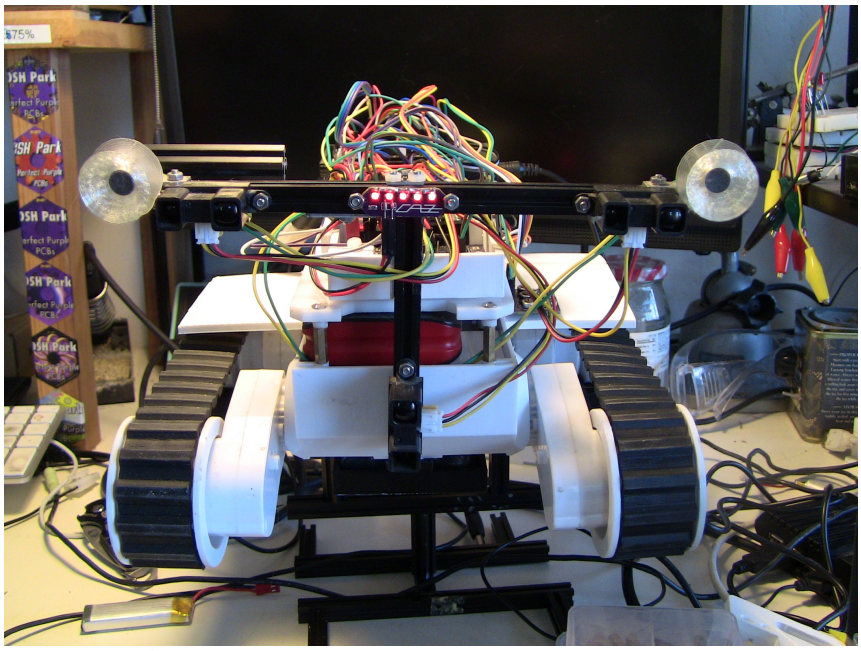


Figure 2.1: An image of the completed RED robot.

In order to implement a new approach to RSSL, and to accomplish this task on a budget, a unique system was developed. This system had to meet several requirements:

- Mobility - As the idea behind this project relies on a triangulation routine in which robots locate and surround a sound source, the platform had to be mobile. This requirement necessitated developing a system of motor control as well as sensory input.

- Networked Communication - The triangulation routine used in this project demands that each of the robots share its location and heading information with the others as well as a central hub. This called for a wireless communication system that would also be mobile and low power.

- Multiple Power Requirements - The various sensors, ICs, motors, and microcontrollers require differing levels of power input. A system had to be designed where the various voltages could be housed and distributed reliably.

- Embedded Functionality - As the project relies on mobility, it had to be fairly small. The computational demands required a system that could handle a variety of signal processing functions while still maintaining a small form factor.

- Low Cost - This project was supported almost entirely by the income of a graduate student in a music department.

This chapter describes the design and build processes involved in creating the robotic network as well as the hardware and circuitry involved in the final product. It starts with an overview of the primary microcontroller and cape, and then delves into the circuit boards that were custom designed specifically for this project. Next, the various sensors used in the project are covered, followed by a description of the bodies and special construction of the robotic frames. Finally, the power requirements and distribution system are explained.

2.1 Beaglebone Black

The requirements for this project regarding hardware are fairly specific. As each robot needed to function autonomously and with a relatively high-level of processing power, only a limited number of microcontrollers could be considered. There were a few that met the requirements of processing speed, such as the Raspberry Pi 3, the ESP32, and the STM32F4

series microcontrollers, and the Dragon Board, but the only one that offered the processing speed as well as advantageous hardware I/O was the Beaglebone Black (BBB). The Beaglebone Black is an embedded processor that features a wealth of programmable GPIO pins offering multiple functions. The processor is an AM335x 1GHz ARM Cortex, which provides: 512MB DDR3 RAM, 4GB 8-bit eMMC on-board flash storage, 3D graphics accelerator, NEON floating-point accelerator, and two PRU 32-bit microcontrollers.

Figure 2.2: The Beaglebone Black Microcontroller.

The Beaglebone Black is capable of running embedded versions of the Linux operating system, specifically the Debian release that is used for this project. It offers a variety of interconnection options, including tethered Ethernet and WiFi, which provide means for interacting with the board via SSH from another computer. Another important consideration in choosing the Beaglebone Black was its online community of users and support; with the exception of the Raspberry Pi, the (BBB) has the largest community of support of any the other microcontrollers, through which much needed troubleshooting and assistant was often sought and found.

The hardware I/O on the BBB was the most important qualification leading to its selection. The BBB offers 92 GPIO pins, many of which are reconfigurable. 2.3 shows some of the numerous options available as standard GPIO functions on the BBB.

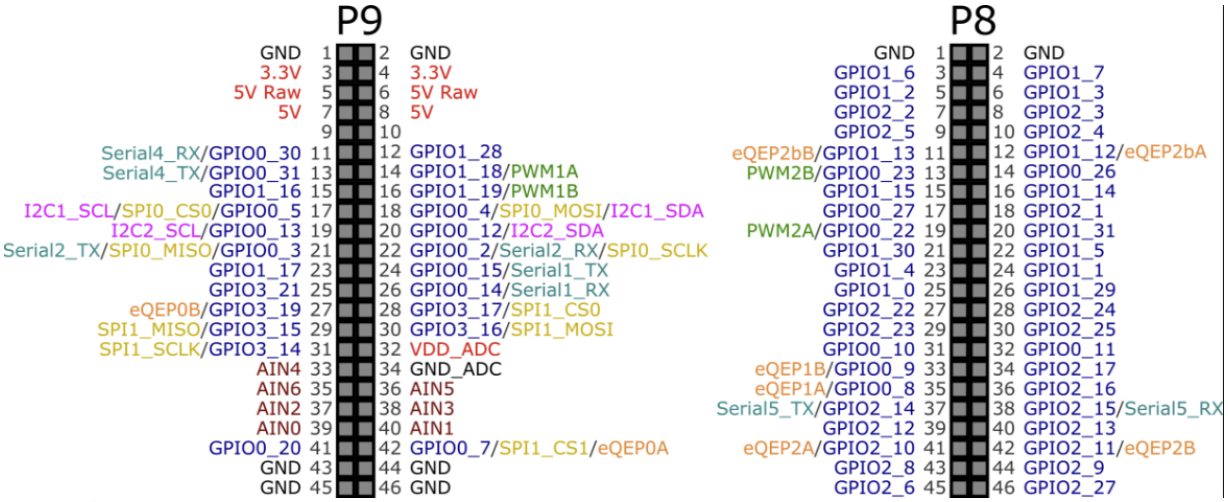


Figure 2.3: Pinouts for the Beaglebone Black Microcontroller showing various hardware I/O capabilities.

Usage

The BBB is the brain of the robots. The programs necessary for localization and interaction are all stored on and run from the BBB. The BBB functions primarily as a headless unit; working with it requires interaction via SSH, programming from the console. This was done with the PUTTY program in Windows and Terminal on a Macintosh.

The BBB manages most of the hardware functions required by the robots that are not related to audio. This includes motor control, analog signal input from sensors, and I2C communication. The BBB also handles the networking hardware and setup, as well as some of the power distribution. While the BBB technically is responsible for audio processing as well, it is really the Bela Board that makes this possible. For this reason, these functions will be covered in the next section.

2.2 Bela Board

The original concept for this project had several different approaches in mind for the hardware that would accomplish the necessary tasks of audio processing and systems control. In the end, the BBB was chosen because of its robust hardware capabilities, but also because of the recent release of an audio cape made specifically for the BBB called the Bela Board.

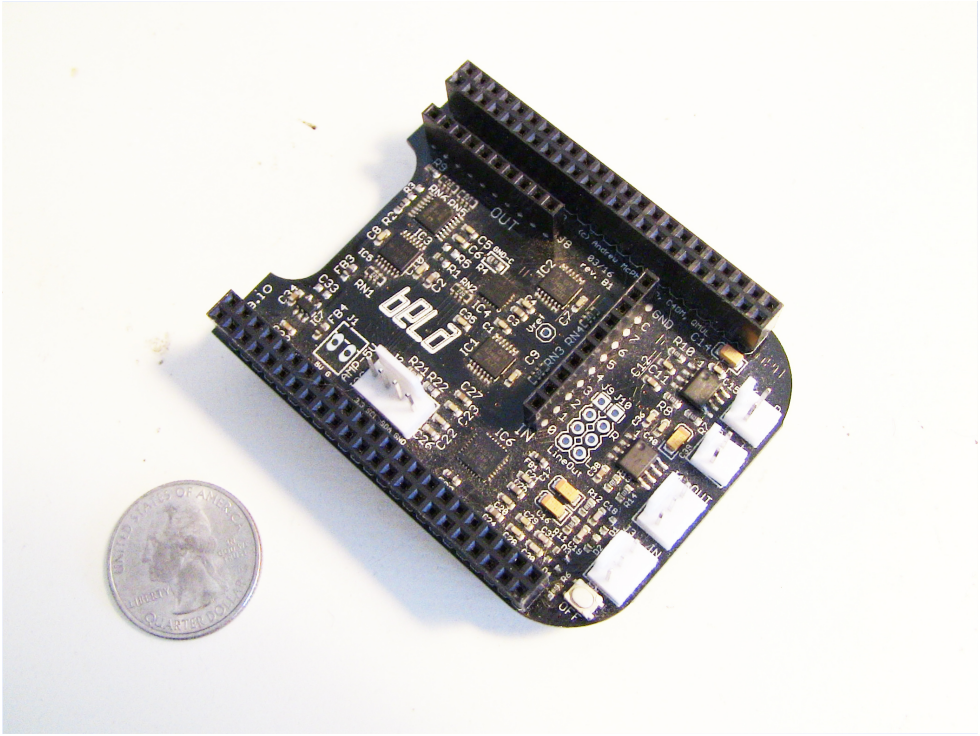


Figure 2.4: The Bela Board audio cape for the Beaglebone Black.

The Bela Board was designed by the Augmented Instruments Laboratory at C4DM, Queen Mary University of London. The Bela Board provides extremely low-latency audio processing (¡.5ms) due to its use of a real-time Xenomai Linux kernel. For audio and control hardware, it boasts:

- 16-bit stereo audio I/O at 44.1kHz
- 2X 1W 8ohm speaker amplifiers
- 8X 16-bit analog input at 22.05kHz

- 8X 16-bit analog output at 22.05kHz
- a6X digital GPIO at 44.1 or 88.2kHz

The Bela Board also embeds libPd, SuperCollider, and its own C++ audio library (see section 2.7, while still allowing access to the BBB's other features such as the embedded Linux functionality and numerous GPIO pins. The Bela Board, however, does take control of some GPIO pins for its own digital I/O and PRU capabilities. This fact made it necessary to make changes to the Bela's device tree and cape manager - the files that determine which hardware is setup at boot time and how that hardware functions.

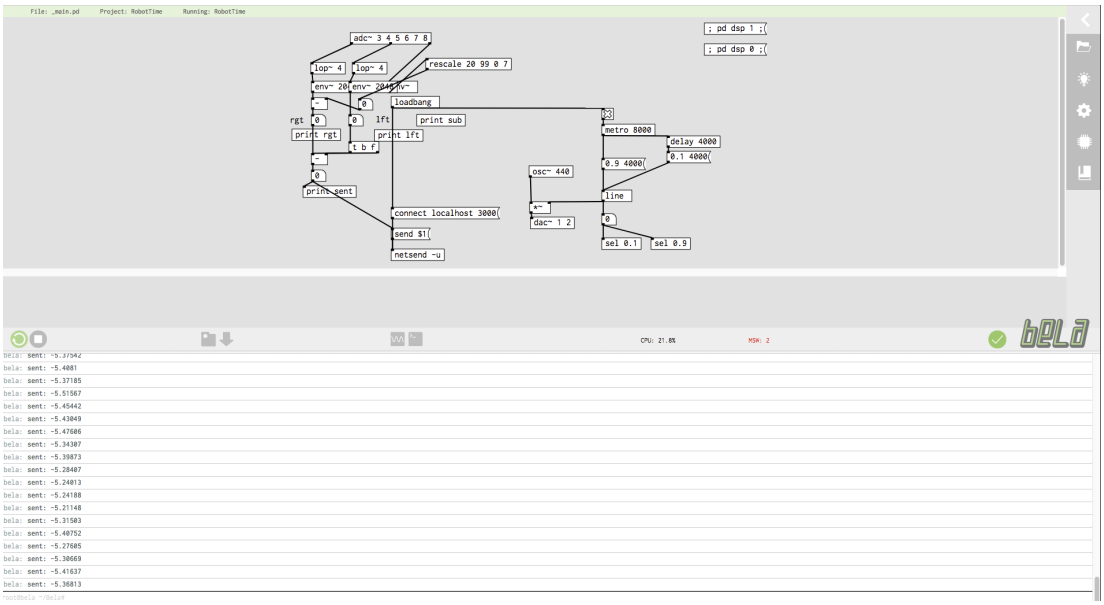


Figure 2.5: The Bela IDE, showing a running version of the Pure Data patch that determines envelope detection difference.

The Bela Board IDE provides easy access to the file system and projects contained on the device. It is accessed by a web browser that connects through the BBB's IP address. Once connected, the user has the ability to make changes to projects by uploading files to specific project folders. The file system used by the Pd programs functions the same as it would on any computer program, where subpatches can be called as long as they are present in the project folder. The work flow for updating files and projects requires that patches be updated offline, then uploaded, overwriting the old files.

Usage

The Bela Board is used primarily for audio signal processing. Depending on the method of DOA detection used (see 3.3), the Bela board runs one of two Pd patches each of which uses an [adc~] object to connect with audio codecs used in obtaining analog signals from the Audio Sensor Board. The Bela Board also receives analog input from potentiometers that control microphone calibration. These signals use the Bela Boards additional analog ports. These ports exist aside from the analog input pins of the BBB, and are accessed by Pd through hardware analog-to-digital converters on the Bela Board. Finally, the Bela Board handles audio output through its amplified speaker outputs.

2.3 Custom Circuit Boards

This projects' reliance upon embedded processing and small-package microcontrollers necessitated the use of special circuits designed for the audio and control systems signal processing. The following sections will discuss the functionality and design of these circuit boards.

2.3.1 Audio Sensor Board

The ASB was designed as a tool for microcontrollers to interpret audio signals through analog and digital GPIO pins. It does not transmit full audio streams, but rather converts audio signals into data that is understandable via simple digital or analog voltage values. In this way, it acts as a sensor like any other digital or analog sensor, with sound being the environmental characteristic that it measures. For a full schematic of the ASB, see Appendix A, section 4.2.

Stereo Preamplifier: The first stage, which is necessary for proper level input, is the preamp. A circuit schematic used for simulation of this stage is shown in Fig. 2. The preamp is designed to be used with typical electret condenser microphones, providing the necessary gain and bias voltage required by such devices. The stereo preamp circuit (in fact all analog circuits in

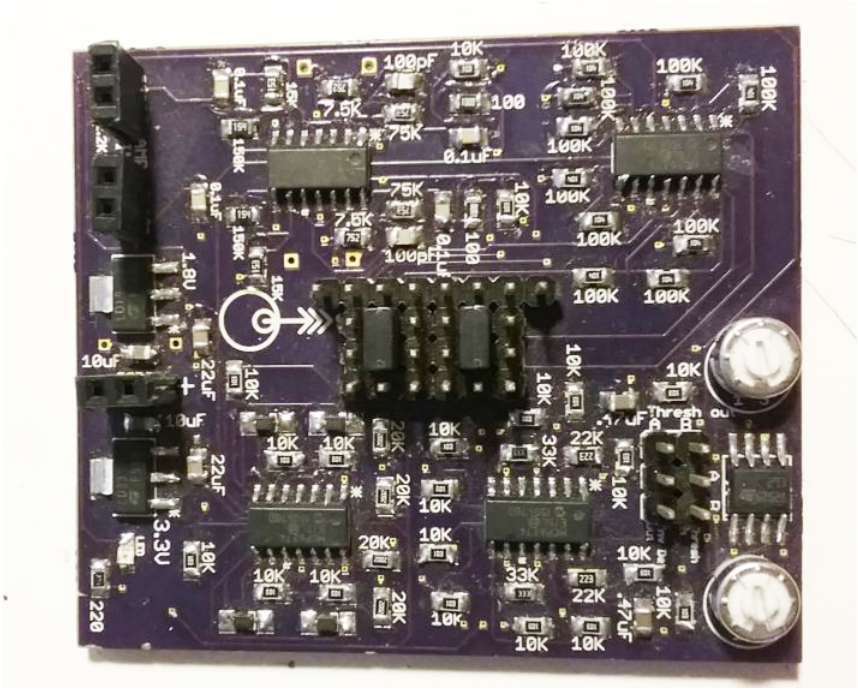


Figure 2.6: The Audio Sensor Board.

the board) is built using MCP6274 rail-to-rail operational amplifiers with a single 3:3V supply source and is based on two cascaded inverting amplifier circuits. The stereo output signal is available in two forms: with a 1V DC-bias, if used to connect to further stages, or AC-coupled if routed directly to a power amplifier. The feedback resistors in the second inverting amplifier are exposed, which can be used to change the built-in gains by associating resistors in parallel.

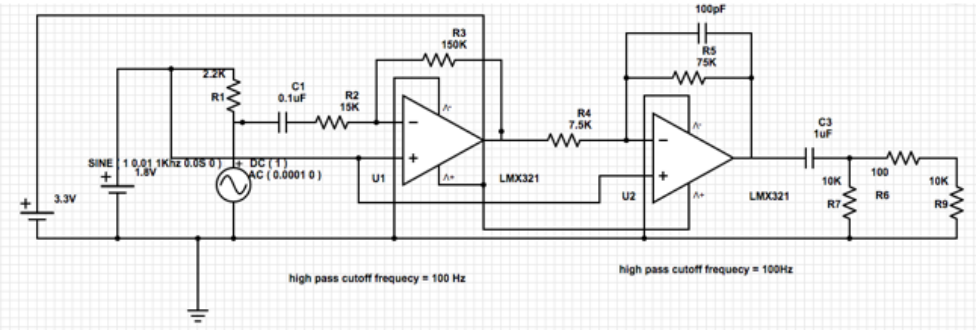


Figure 2.7: Schematic for the preamp section of the ASB.

Simple Signal Operations: Following the preamp the ASB provides a signal summer and signal subtractor, allowing for addition or subtraction of the right and left channels. The

summer can be used, for example, in applications in which the combined power of the audio signal is the quantity of interest. The subtractor can be used, for example, in applications in which the direction of the audio source is the quantity of interest. The difference between two signals emanating from a single source picked up by two spaced omnidirectional microphones is correlated to the angle of incidence of the source.

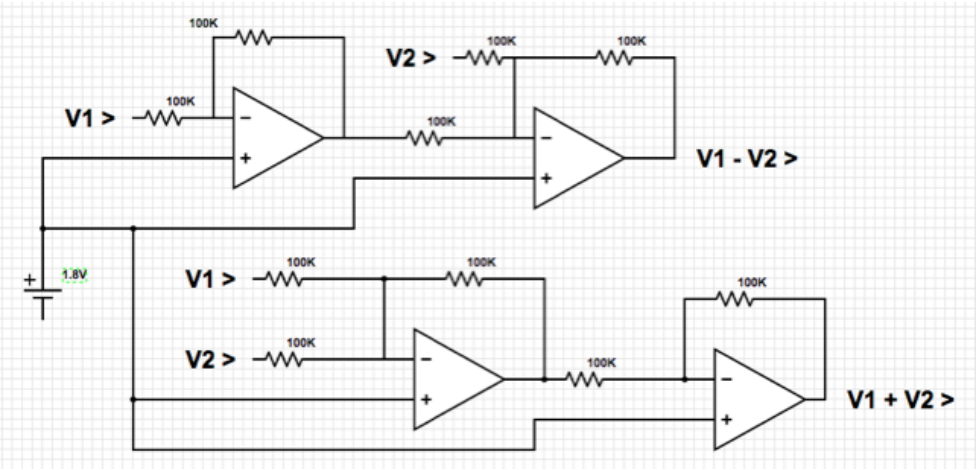


Figure 2.8: Schematic for the summer/subtractor section of the ASB.

Envelope Detection: The next stage is en envelop detector, which receives an audio level input from the preamp and outputs a scaled voltage in the range of 0-3.3v corresponding to the audio amplitude. The circuit is a combination of a precision full-wave rectifier followed by a level-shifter and low-pass filter. This feature is particularly useful for tasks that require analog feedback, such as dimming lights or controlling vibrating motors.

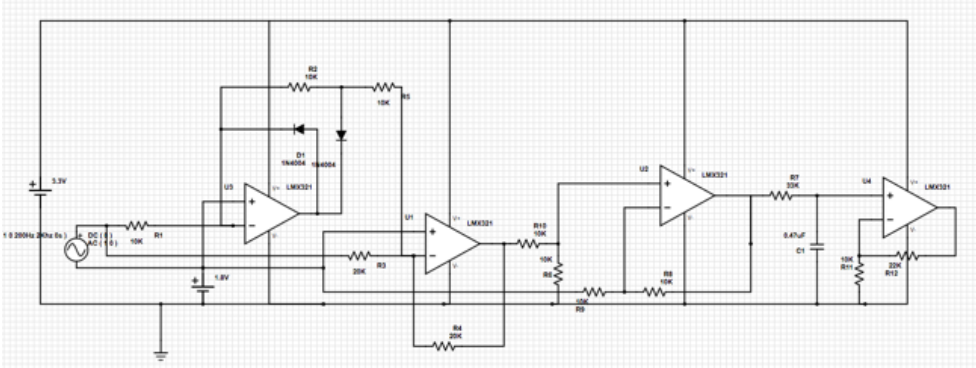


Figure 2.9: Schematic for the envelope detector on the ASB.

Threshold Detection: A scalable threshold detector, using an LM293 comparator, acts as a digital on off switch that can be tuned to a corresponding voltage coming from the envelope detector. The threshold voltage can be varied using a potentiometer mounted on the ASB board. The user can therefore trigger on/off messages that correspond directly to a chosen signal amplitude. For example, a light could be turned on when the noise level in a room reaches a chosen amplitude.

Patching section: Finally, the ASB board is equipped with a versatile patching section that allows users to route signals from one feature to another or multiple section(s) by using jumpers. The patching section can be seen in the center of the board shown in the Patcher Section Figure. The terminals correspond the diagram in Fig.6. This patching section provides a great deal of variability to the boards possible uses. The patching section would allow, for example, the preamp to be routed to the summer, then to the envelope detector, thus providing a voltage measure for the combined signals from the microphones. Another option could be to route the signal from the preamp, to the envelope detector, to the threshold detector, while taking direct outputs from each of those sections, thus providing access to the direct audio, the analog voltage signal, and the digital output of the threshold detector simultaneously.

Figure 2.10: Map for the Patcher layout on the ASB.

Usage

For this project, multiple functions of the ASB are used. First stereo signals are amplified with the preamplifier, then routed to other locations for various purposes. Then, the original amplified audio signal is used for the Cross-Correlation method of Direction of Arrival detection (see 3.3.1). The signal is also routed to the envelope detector and used in the Envelope-Detection method of DOA (see 3.3.2). Lastly, the threshold detectors send digital on/off signals to the BBB and Python application to signify audio event detection, and thus trigger reactions in the form of audio or motorized output.

2.3.2 Robotic Sensor Board

As the design and build process for the robots progressed, more sensors were added, each requiring its own power circuitry and external components. Initially a solder-less breadboard was used to connect the sensors to their respective pins on the BBB, but this practice soon proved cumbersome, messy, and confusing, with wires intersecting and having no labels or designations. For this reason, it was decided that a dedicated circuit board for sensor input and power distribution would be beneficial.

The Robot Sensor Board (RSB) handles routing for most of the signals used on each robot. It also functions as a power distribution center that connects to both the BBB `SYS_VDD` and DGND, as well as external 5V power and GND. It serves as the housing for the robots gyroscopic sensor as well, providing signal routing for the sensors I2C, interrupt, and power pins to the BBB.

The RSB contains three three-pin inputs for the proximity sensors. These are male header pins that fit directly with the three-wire cables coming from each of the sensors. There is then one male output pin per sensor which can be connected to the BBB analog input pins with a single jumper wire. The 5V and GND signals are provided to the sensors through the header pins, thus making the connection a simple plug-and-play operation.

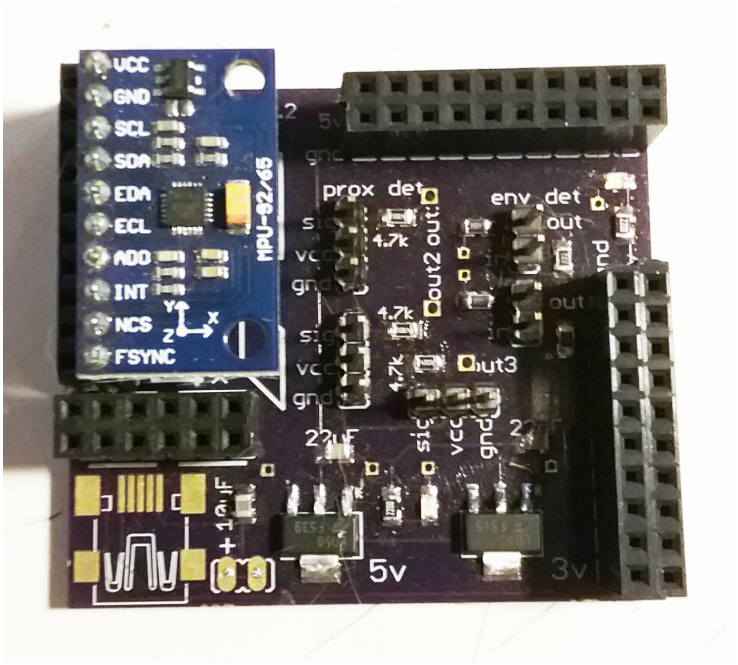


Figure 2.11: The Robotic Sensor Board.

The RSB also handles input from the envelope detectors on the Audio Sensor Board. The output from the envelope detectors requires a voltage divider to create a safe input level (1.8v) for the BBBs analog input pins. On the original bread board, through-hole resistors were used in conjunction with additional jumper wires, which took a good deal of space and added to the general clutter of the signal routing. On the RSB, this is all accomplished with surface-mount components in a very small space. The signal enters through one male input pin, goes through the voltage divider, then is output through a single male output pin. The gyroscope has its own seat on the RSB as well, with power supplied from the VDD pins. Since the gyro uses the two-wire I2C protocol, along with VDD and GND, a simple four-wire connection from the gyro to the BBB is provided.

The RSB also has multiple sections for power distribution. It receives 5v from the BBB's `SYS_VDD` pins and distributes this through a 3.3v regulator. This provides the necessary 5v or 3.3v signal to the sensors depending on their requirements. There are also two header sections that provide additional power supply - ten outputs for 5v + GND, and ten outputs for 3.3v + GND.

These outputs are then used to power other parts of the robots, for example external LEDs and the Mototrol Board.

Usage

The RSB is primarily used as a signal router and input/output hub. The power supply that it receives from the BBB is routed to numerous locations on the robot, including the analog proximity sensors (5v), the indicator LEDs (3.3v), the gyro (3.3v), and the LCD display (3.3v). The jumper section provides a means for the BBB's I2C ports to be shared by the LCD screen and the gyro. The three-pin analog inputs enable easy signal routing of the proximity sensor outputs to the BBB pins. Lastly, the inputs for the ASB's envelope detector, with a built-in voltage divider, allow for ready-made transfer of the analog signals from the ASB to the BBB.

2.3.3 Mototrol Board

The third board developed for this project was the Mototrol Board. The Mototrol Board enables locomotion and odometry for the robots. Early work had been done using a pre-made motor controller cape for the BBB, but that cape provided limited functionality and necessitated the use of specific pins on the BBB which in the end were required for other functions. It also did not provide quadrature encoder readers for odometry measurement. Lastly, it consisted of a large-footprint cape that was very space consuming. Due to these factors, it was determined that it was best to design and fabricate a separate control platform for the motors and encoders on the robots.

The motors are driven using the LM298 dual motor driver IC. This chip can function with power input of 9-25 volts, making it ideal for various-power requirements and situations. The LM298 requires the use of an additional voltage translator IC, which takes the low-voltage PWM output of the BBB and translates it into values usable by the LM298

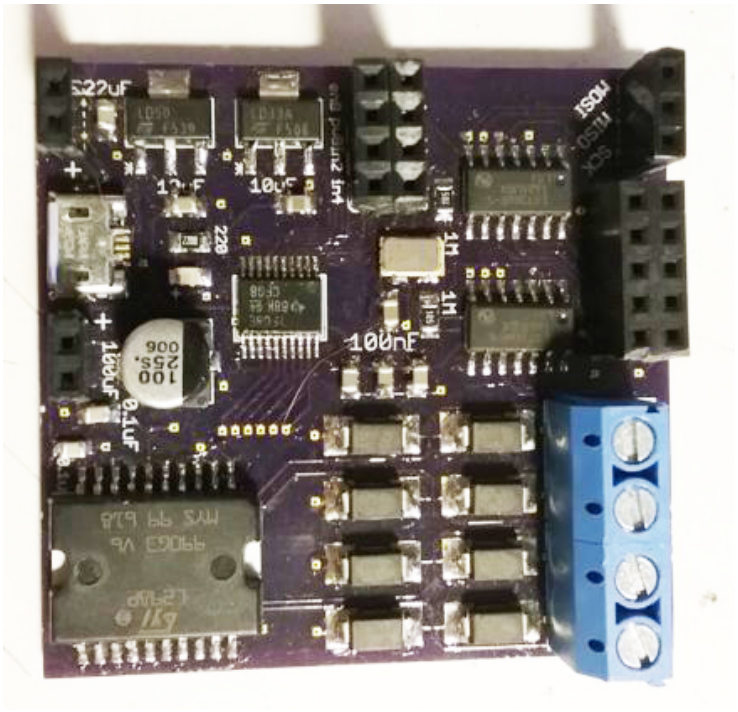


Figure 2.12: The Mototrol Board.

The motors each require three GPIO pins in order to operate: one PWM pin to determine speed, and two digital pins to determine direction (low/high for forward, high/low for backward). As mentioned previously, pre-made capes for motor control necessitate the use of specific pins. For example, the Seeed Studios Moto-Cape, which had been used initially, required the use of GPIO pin P927. Unfortunately, the Bela Board also uses P927 as a power switch. When early testing was done with the Moto-Cape, the BBB would turn off every time the robots left track went backward. (It took a few days to figure this problem out.) The Mototrol Board thus uses header pins for input and output that do not sit directly on top of the BBB —bh jumper wires can be used to connect any of the BBBs GPIO pins to any input or output on the Mototrol Board. In this way, the Mototrol Board is not a cape, but rather a separate control systems board that can also be used with other microcontroller platforms.

Another issue that necessitated the creation of the Mototrol Board was odometry. One of the attractive aspects of the BBB initially was the fact that it has on-board eQEPs (enhanced quadrature encoder pulse detectors). The Rover 5 chassis that is used by the robots contains two

encoders — one per motor, or track. These encoder readers enable the user to determine the amount and direction that each track has moved, thus enabling accurate odometry. However, once the Bela Board was added to the project, the eQEPs stopped working. It again took a number of days to figure out why, but eventually it was determined that the kernel used by the Bela Board had not included initialization of the eQEP hardware in its configuration. A good deal of time was then spent attempting to recompile the Bela kernel with the necessary configuration, but with limited success. Thus it was determined that it would be easier to add external hardware encoder ICs to the project than to recompile and reconfigure the Bela kernel. The encoder ICs used on the Mototrol Board are the LS7366 chips by LSI. The chips communicate with the BBB (or any microcontroller) using the SPI protocol.

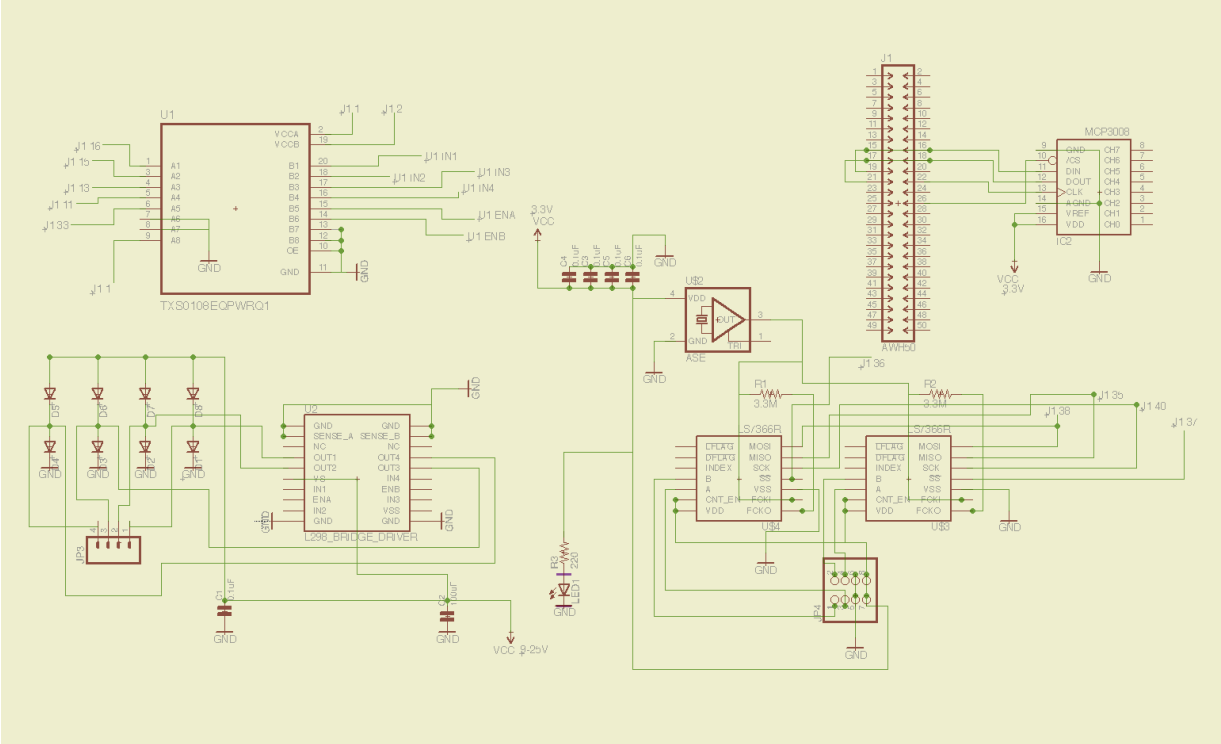


Figure 2.13: shows the schematic for the Mototrol Board.

Usage

The Mototrol Board is used as the robots' motor driver. For the final implementation of the project, the eQEP inputs on the BBB are functional (the result of custom changes to the device tree and pinmux configuration of the BBB), so the encoder readers on the Mototrol Board are not needed. The version of the board that is currently on the robots does not have the encoders or their power supply circuitry in place.

2.3.4 Analog Calibration Board

The final custom circuit board on the robots was designed to serve as a source of analog inputs for the Bela board and the Python programs. The Board consists of six potentiometers, each acting as an analog voltage divider. The board receives power and ground from the 3.3v supply on the BBB, which is then divided to provide analog values to be read by the Bela Board. These values are then used by the Pd patch for calibrating the microphones and assigning frequency and volume to the audio output.

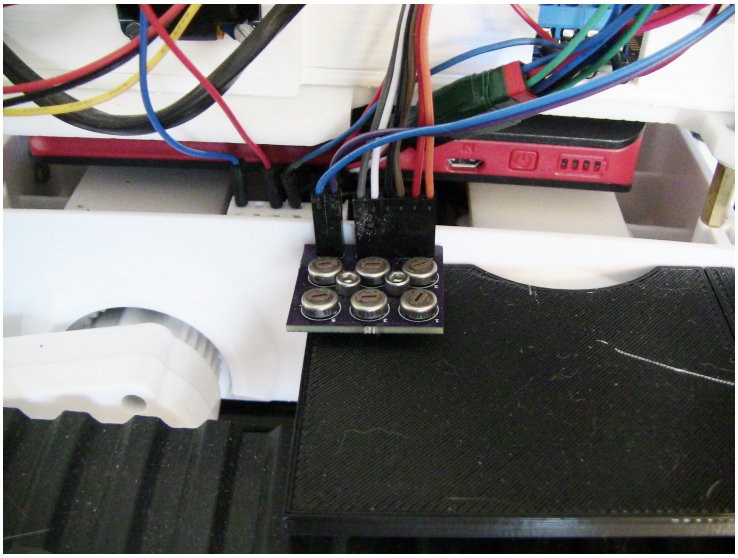


Figure 2.14: The Anoalog Calibration Board.

2.4 Sensors

Each robot contains a total of seven sensors. The sensors are necessary for the basic functionality of this project, providing input relating to obstacle avoidance, sound detection, and odometry. The following is a description of those sensors and their purposes.

2.4.1 IR Proximity Detectors

Each robot contains three Sharp GP2Y0A02YK0F Long-range IR Proximity Detectors. These are analog distance-measuring sensors with a range of 20 to 150 cm. that use a combination of PSD (position sensitive detector), IRED (infrared emitting diode) and signal processing circuitry. The operation of these sensors is not influenced by object reflectivity, environmental temperature or duration of operation, making them well-suited to the applications of this project. They require an optimal 5v power supply, and output an analog voltage in the range of 0-3.3v relative to the proximity of the objects they detect. They work with a basic three-wire connection and, unlike many common digital proximity sensors, require no external timing program for accurate distance measurement. These sensors are used for obstacle detection and avoidance [Sha06].

2.4.2 Microphones

Two CUI Electret Condenser Microphones. These are small omni-directional microphones with a signal to noise ratio (S/N) f = 1 kHz, 1 Pa, A-weighted 58dBA.

While the CUI are not the highest-quality microphones available, they were chosen to meet the requirements of the project; they are low-cost, and require simple power supply circuitry. Also, as the microphones do not record audio, but merely sense amplitude and frequency, it was determined that they are more than adequate for these tasks. The frequency response (see fig2.16) is flat enough that any signals from environmental sound or from the robots will not be

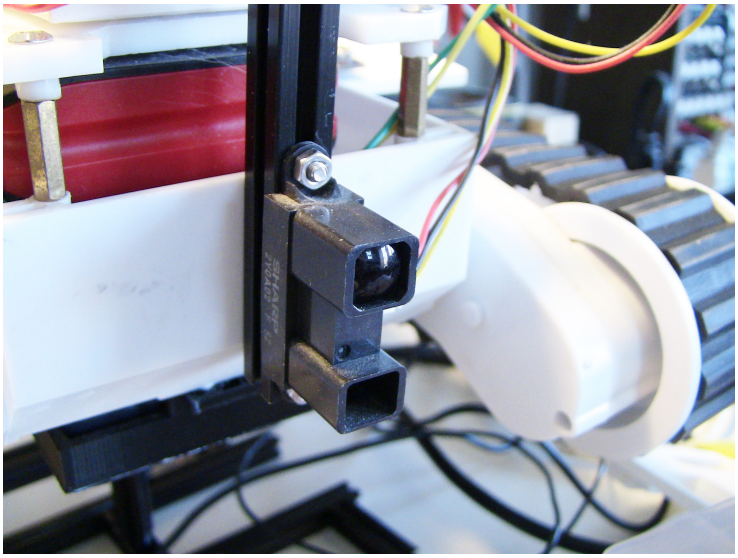


Figure 2.15: One of the Sharp IR Proximity Sensors.

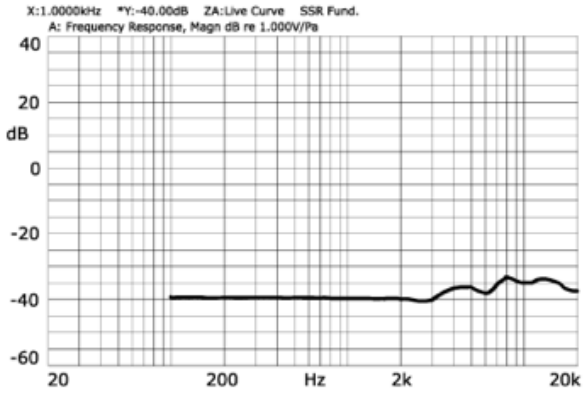


Figure 2.16: Frequency response of the CUI Electret Condenser Microphone.

negatively effected. The microphones are connected to the Audio Sensor Board (see 2.3.1) in order to provide amplitude measurements for sound in the environment. They are also used as audio input devices running to the Bela Board for later frequency detection, which is then used for the robots locational determination of each other [CUI16].

2.4.3 Gyroscopes

Each robot has two magnetometers available for use at any time. There is one HMC5883L 3-Axis Digital Compass IC on each robot. This is a magnetometer that features a 12-bit ADC,

enabling 1-2 positional accuracy. The HMC5883L was chosen because of its low cost and ease of use; it functions with a two-wire I2C connection to the BBB. It is also beneficial due to its low power consumption at 3.3v. The small packaging and outline, even on its breakout board, allow it to fit nicely on the Robot Sensor Board. The HMC5883L provides compass location as one half of the odometry system that is used for positional determination as its output is integrated in to motor control system [Hon13].

The second gyroscope is part of a larger inertial measurement unit (IMU). The unit chosen for this project is the MPU9250. The MPU9250 contains an accelerometer, a gyroscope, and a magnetometer all in one package. It communicates with the BBB over I2c, like the HMC5883L, which means they can share the same port yet be activated and used separately. The MPU9250 is a much more sophisticated motion processing unit than the HMC5883L, as it can provide data from any of its components individually or in combination, supplying therefore more complex output than its magnetometer-only companion. The MPU9250 relies on a Python library that supplies Tait-Bryan angles, compass data, and quaternion angles as output.

Usage

Both the HMC5883L and the MPU9250 were tested and used extensively throughout this project. In the end, the MPU9250 proved more reliable due to the HMC5883L's susceptibility to magnetic interference. The MPU9250 also provided output that was more stable and less prone to error. However, the MPU9250's reliance upon a complex Python library for operation caused its own set of difficulties - I2C communication problems, device initialization issues, and the like. The final result was that both IMUs were employed as backup checking systems rather than primary instruments of position or heading.

2.4.4 Encoders

There are two 333 CPR Quadrature Encoders. The 333 CPR (counts per revolution) encoders come built into the Rover 5 chassis that was used for this project. The encoders determine revolutions of the wheel shaft to which they are attached by sensing a rising or falling edge pulse. They connect to the encoder readers on the BBB through a four-wire connection: VDD, GND, Ch. A, Ch. B (for rising and falling edges). These sensors are used to measure odometry the primary means for positional determination used by this project. However, the encoders themselves are only one half of that measurement system. The other half consists of the eQEP readers on the BBB [Tex08].

2.5 Robotic Chassis and Parts

The physical construction of the robots presented numerous challenges. It was important to find components that offered mobility, accuracy and modularity. A number of prototypes were created using various modular building systems, but it was eventually determined that a pre-existing chassis with built-in motors best suited the needs of the project.

2.5.1 Rover 5 Chassis

It was ultimately decided that the Rover 5 chassis by Dagu provided all the functionality the project required, and was sufficiently inexpensive. The Rover 5 contains two motors that require only 7.2v of power, yet have a decently high torque at 10Kg/cm. The optimal speed of 1Km/hr makes the robots slow enough that they can be easily controlled, yet fast enough that they can accomplish their given task in large spaces quickly. The chassis also provides four noise suppression coils around the motors, which help to separate motor noise from the microphones. The most important facet of these chassis is the fact that they contain built in encoders on the wheel shafts.

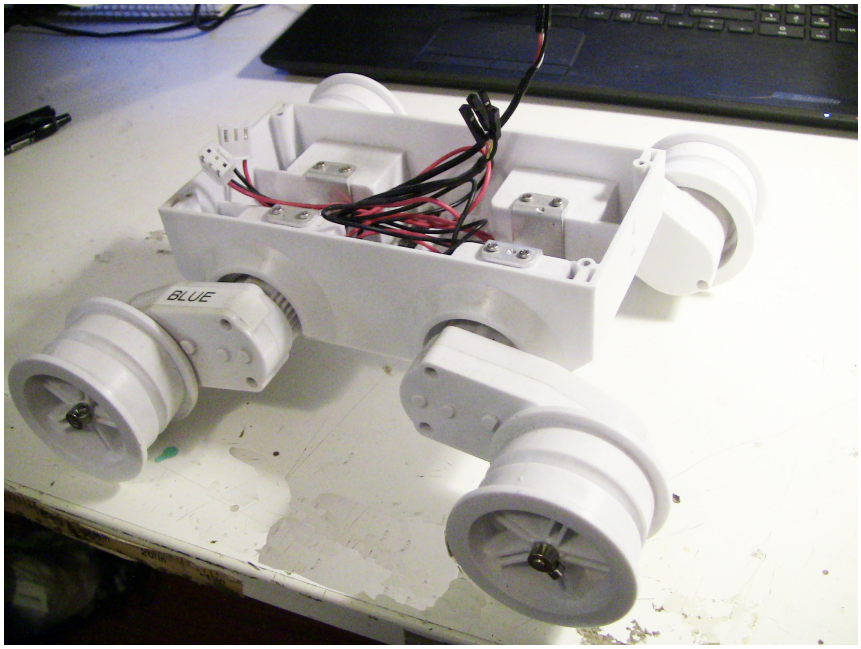


Figure 2.17: The Rover 5 chassis used for the robots.

2.5.2 3D Printed Components

As the robots use a number of non-standard components, it was necessary to create housings that the chassis didn't already provide. The various power supply requirements of the boards and sensors also made it necessary to have multiple batteries on board, which also needed storage space. Therefore, there are five parts of the robots' bodies that are custom designed and fabricated using 3D printed materials and laser-cut acrylic:

- LiPo battery compartment for the underside of the robot.
- Compartment for the three custom circuit boards.
- Case for the BBB/Bela.
- Microphone cones printed with flexible filament to dampen sound from the motors and rear noise.
- Track/motor covers to shield the microphones from motor noise.
- Acrylic shelf for the circuit boards and controller.

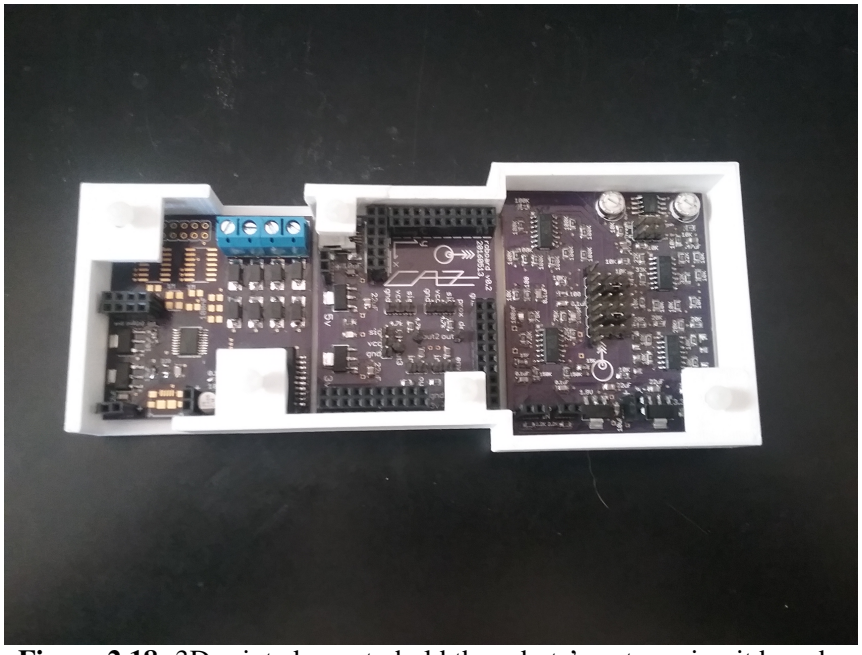


Figure 2.18: 3D printed case to hold the robots' custom circuit boards.

These components not only allow for all of the hardware and circuitry to fit on the otherwise insufficient chassis, but also keep everything well organized and in place. Early iterations of the robots featured a wealth of wires and cables running to and from various places, which created difficulties in accessing and changing wiring. The final design places all wire access points within easy reach.

2.6 Power Supply

The robots have very specific power requirements due to the number of sensors and motors used, as well as the specific power requirements of the BBB and Bela Board. Therefore, powering the system requires more than one supply source, rated at different voltages and amps. Each robot uses a total of three batteries, and disseminates power from those batteries in a very specific manner using GPIO pins on the BBB as well as power supply pins from the Robot Sensor Board.

The largest voltage battery is the one for the motors. This is a Tenergy 9.6V 2000mAh rechargeable battery that sits in a 3D printed support compartment on the underside of the robot.

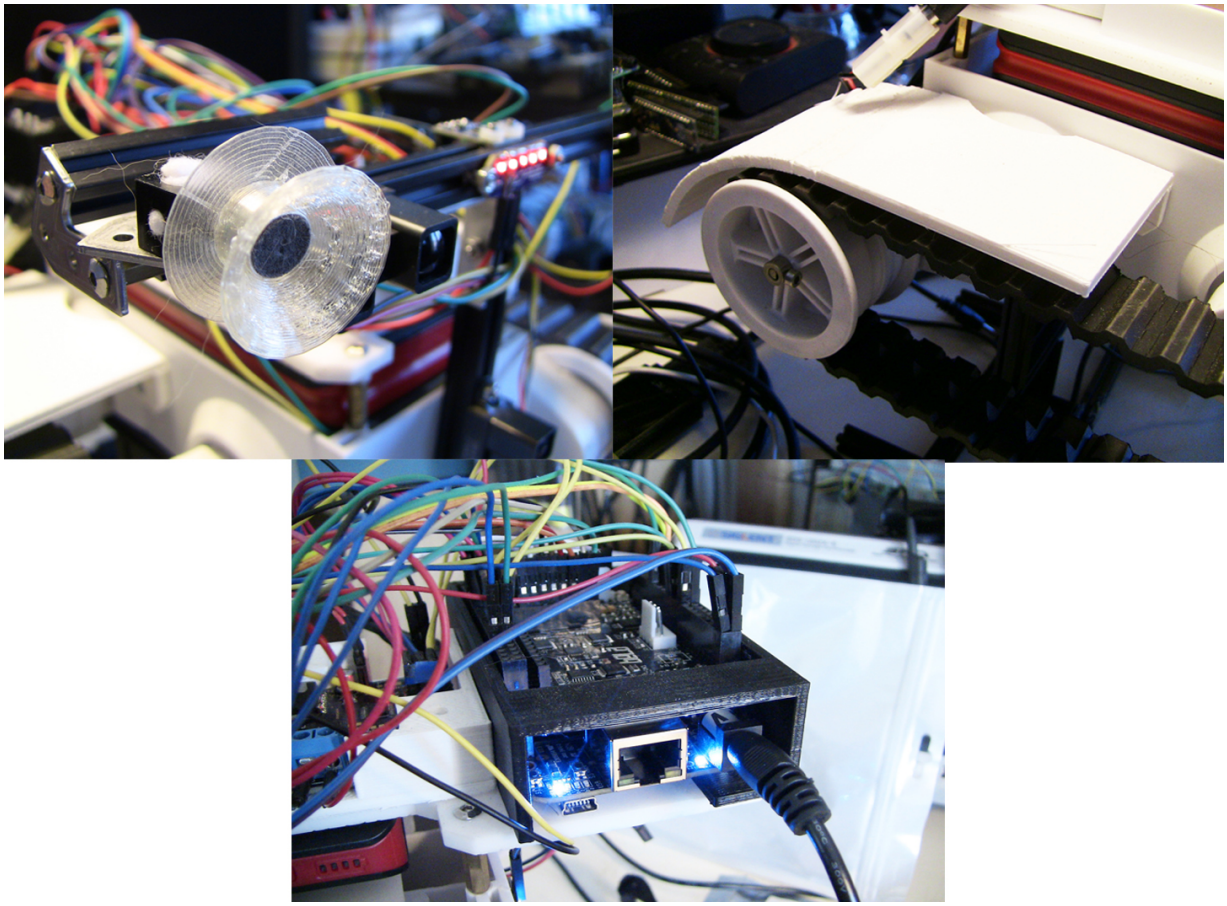


Figure 2.19: 3D printed parts on the robot bodies.

This battery is only used to supply voltage to the motors, as its output is large enough to damage the other circuitry. The output goes directly to the VDD and GND pins of the LM298 H-Bridge on the Mototrol Board.

The BBB is powered by a Kmashi USB battery pack that puts out 5v at either 1 or 2 amps depending on the output used. This connects to the BBB with a USB-to-barrel jack cable, going to the BBB's 5v input jack. This is necessary for two reasons: 1) The WiFi dongle on the BBB tends to lose functionality due to low power if the board is only powered via the USB input, and 2) The Bela Board requires that the 5v power input be used in order for the powered speaker outputs to function. The power supply from this battery is then also used to power the Mototrol Board and the Robot Sensor Board via the BBB's `SYS_VDD` pinouts. The `SYS_VDD` pins are connected by jumpers to the 5v inputs of the RSB, which then provides power to the proximity sensors, the

eQEPs, the gyroscope, and the LEDs.

The third battery is a Tenergy 3.7v rechargeable battery that is used for the Audio Sensor Board. This connects to the 1.8v and 3.3v regulators on the ASB, providing adequate power for the entire board. All of the batteries on the robots are rechargeable.

The above sections detail specific parts of the robots. When combined, the parts create a unique system that is capable of autonomy, networked communication, sound perception and analysis, and positional tracking. The overall system can be represented by figure 2.20.

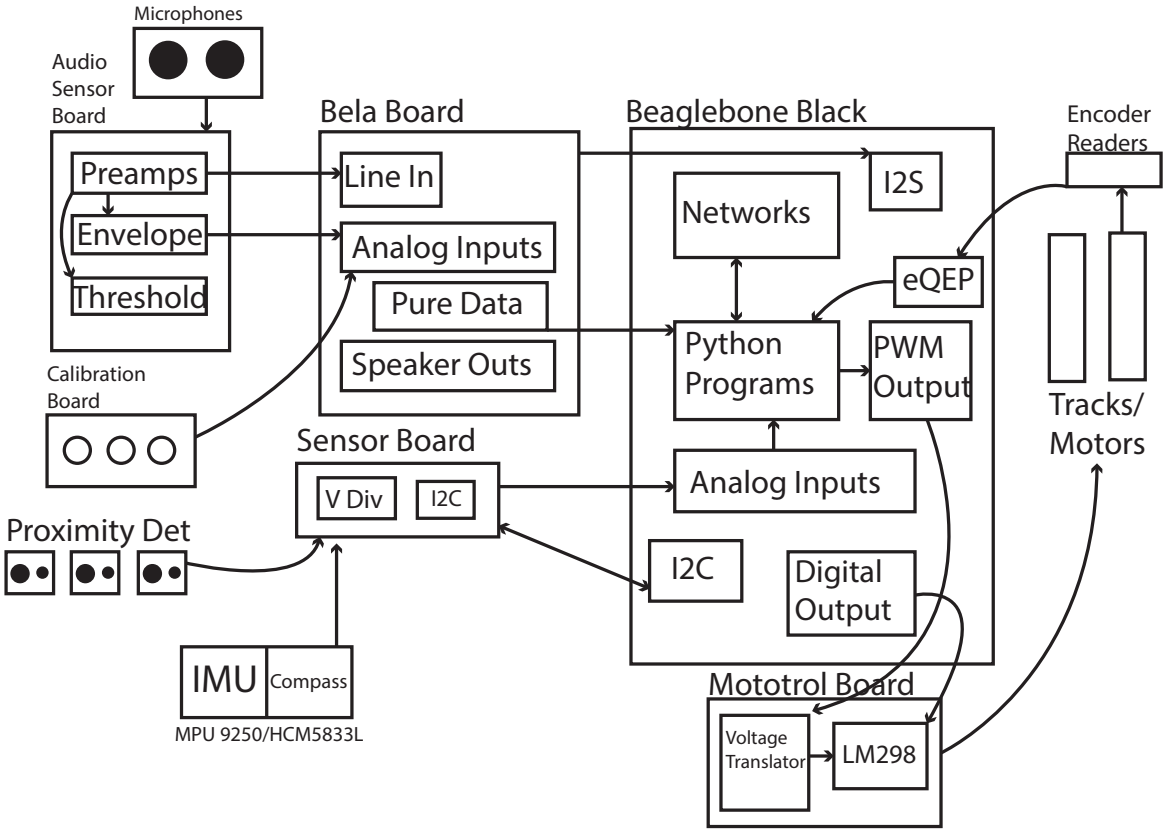


Figure 2.20: A representation of the robotic system and the connection/interaction of separate components.

2.7 Pure Data

Pure Data is an open source visual programming environment that runs on anything from personal computers to embedded devices (ie Raspberry Pi) and smartphones

(via libpd, DroidParty (Android), and PdParty (iOS). It is a major branch of the family of patcher programming languages known as Max (Max/FTS, ISPW Max, Max/MSP, etc), originally developed by Miller Puckette at IRCAM.

Pd enables musicians, visual artists, performers, researchers, and developers to create software graphically without writing lines of code. Pd can be used to process and generate sound, video, 2D/3D graphics, and interface sensors, input devices, and MIDI. Pd can easily work over local and remote networks to integrate wearable technology, motor systems, lighting rigs, and other equipment. It is suitable for learning basic multimedia processing and visual programming methods as well as for realizing complex systems for large-scale projects.

Algorithmic functions are represented in Pd by visual boxes called objects placed within a patching window called a canvas. Data flow between objects is achieved through visual connections called patch cords. Each object performs a specific task, which can vary in complexity from very low-level mathematical operations to complicated audio or video functions such as reverberation, FFT transformations, or video decoding. Objects include core Pd vanilla objects, external objects or externals (Pd objects compiled from C or C++), and abstractions (Pd patches loaded as objects).[Pur99]

2.8 Python and Python Libraries

Python is a high-level programming language that supports multiple paradigms such as procedural and object-oriented coding. Python is designed to be easily readable, with structures dependent upon whitespace. The language provides dynamic memory management, meaning things such as variables and objects are automatically assigned to computer memory, rather than requiring the programmer to allocate the necessary memory themselves. Most importantly, Python is widely supported on Linux and embedded-Linux systems such as the Raspberry Pi and Beaglebone Black. Python is also open-source, with a large and robust community of users, which makes for easy problem solving.

This project relies heavily on Python programs running on the BBB. These programs handle all robotic functions that are not specifically related to audio signals. This includes motor control, sensor reading, and networking, among other low-level tasks. The program running on the hub computer is also written in Python. The specific programs running on the BBB and hub

will discussed in greater detail in sections 3.2.1, 3.2.2, and 3.2.3.

The hardware I/O upon which this project relies depends upon a number of Python libraries - code modules that contain methods for handling specific tasks, which can be called as functions from the main program by importing them. There are three Adafruit libraries which come from Adafruit_BBIO (Beaglebone I/0). These are the ADC, GPIO, and eQEP libraries. The ADC library handles both PWM output to the motors and analog voltage input from the proximity sensors. The GPIO library handles all digital I/O functions, such as HIGH/LOW signals for determining motor direction. The eQEP library is a new library that enables easy monitoring of the quadrature encoder readers [Coo]. The magnetometer and IMU also rely on their own libraries, namely the rc_mpu_py library written by Mauricio de Oliveira and James Strawson [dO18]. The socket library is also necessary for creating all of the UDP network connections that exist on the robots and hub computer. On the hub, for the graphical mapping (discussed in section 3.2.4) the PyGame library is used for all drawing functions. Aside from these libraries, standard libraries such as math and sys are used.

2.9 Custom Changes to the Beaglebone and Bela Board

The Beaglebone platform is is very much oriented towards control systems related to GPIO functionality, but by itself, lacks easily accessed audio capabilities. Conversely, the Bela Board, because of its reliance upon some of the Beaglebone's high-speed processing power and certain GPIO pins, as well as its requirement of a real-time kernel, relinquishes some of the Beaglebone's hardware performance in its out-of-the-box configuration. This project presents unique challenges to the Beaglebone and Bela Board platforms due to the fact that it requires extensive use of the Bela Board's audio capabilities as well as the Beaglebone's hardware functions. Thus, some fairly critical and low-level changes had to be made to the operating system[1]. This section will detail

[1] It should be noted that the changes made to the Bela OS were made possible in large part by Giulio Moro, one of the researchers at Queen Mary London responsible for the Bela Board. Moro often directed the author through the

the changes that were made and briefly discuss the reasons for each adjustment.

2.9.1 Changes to LibPd

The Bela Board has two methods by which one can make use of Pure Data patches. From the Bela website:

> There are two ways you can run Pd patches on Bela: using Enzien Audio's Heavy Audio Tools or using libpd. Libpd is a GUI-less version of Pd which allows to embed Pd patches into other programs, whereas Heavy is an online service that generates highly-optimized C code from a Pd patch. Libpd runs all of Pd vanilla objects, almost all of of which are supported on Bela. Pd patches compiled with Heavy can only contain a subset of Pd vanilla objects. Using libpd, a patch can be run immediately, as soon as it is copied over to the Beaglebone. When using Heavy, you have to re-compile the patch every time you modify it, which may take up to one minute and requires an internet connection.[Mor18]

The development stage of this project necessitated continual changes and updates to the Pd patches used for cross-correlation and envelope detection, often dozens of times each day. Therefore, it was decided that libpd was the best route to take for embedded Pd use. However, the version of libpd that ships with the Bela Board presented certain obstacles.

The first obstacle to be overcome was the limited number of Pd objects available for use with the Bela Board. While most of the standard objects that come with the Vanilla distribution of Pd were present, some had to be altered or otherwise substituted. Early networking tests proved difficult due to the functionality of the [netsend] object. The object initializes to a state that uses the TCP protocol, but for reasons unknown the TCP stream would not connect through any networking system tested. The [netsend -u] object, however, employs the UDP protocol and functioned properly with initial testing. Unfortunately, the object as written posted a continuous stream of error messages to Bela IDE console which caused the CPU to choke under the strain. After discussing this issue with Giulio Moro on the Bela forum, Moro compiled a new version of the object that ignored these error messages and made it available for use in this project.

---

step-by-step processes necessary to make custom changes to the Bela kernel and OS

The second obstacle was the available block size in Pd applications running on the Bela Board. The custom [xcorr∼] object (see section 3.3.1) relies on a block size that is larger than the standard black size of 16 that the Bela uses. During initial testing of the object, it returned values that did not correspond to expected outputs. It was therefore necessary to recompile a version of libpd with a larger black size - 64 for this project. In order to do this, Moro made available Bela's active libpd repository. Once cloned, it was possible to alter libpd's s_stuff.h file by changing #define DEFDACBLKSIZE from 16 to 64. With this change, the [xcorr∼] object functioned perfectly.

2.9.2 eQEP Requirements

A recurring source of frustration throughout the implementation of this project was the functionality of the BBB's quadrature encoder readers. The eQEPs (see section 2.4.4) in early releases of the Bela Board were not accessible. While the eQEPs were one of the main factors in the BBB's appeal, it was decided that the BBB/Bela combination could still be used, but with a significant hardware workaround. To accomplish this, the Mototrol Board was designed with external hardware encoder reader ICs. For several months while the early distributions of the Bela kernel were still in use, this proved to be a functional way to deal with this issue; it merely required implementation of additional hardware libraries and code.

After a number of requests on the part of the author to the Bela's designers, updated versions of the kernel made the eQEPs accessible. However, the standard pins used by the eQEPs had been allocated to other functions related to the Bela digital inputs. A number of changes therefore had to be made to the standard pin configurations of the BBB/Bela. To do this, the Beaglebone device tree binary (dtb) [2] had to be rebuilt and recompiled. The Bela's dtb file mainly specifies which pins are allocated to which functions. The changes made to the dtb file thus

[2]The dtb file specifies which peripherals are loaded, their order, and their functionality. On the BBB this file handles the GPIO functions specific to digital IO, PWM output, and the ADC.

dealt with deallocating pins that had been assigned as digital IO. Once this was done and the dtb file recompiled, the pins could be properly configured to eQEP functionality with the BBB's "config-pin" command line tool.

2.9.3 General GPIO Changes

When this project began, the BBB had loaded capes using the device tree overlay. Standard capes [3] could be loaded or unloaded using the uEnv.txt file. However, during the building phase of this project, the BBB and Bela kernels were upgraded and changes were made to the manner in which such devices were loaded and accessed. Along with this change, access to functions such as the eQEPs were no longer loaded by default. Instead, one of "universal capes" had to be loaded at boot. However not all universal capes contained settings for all of the pins functions required by this project. Upon searching through the universal cape overlay files, one was located that contained the needed settings. This is now loaded at boot using the Crontab application, which specifies the cape name and settings.

It is significant that the development of this project assisted in turn with the development of the Bela platform. The Bela Board came out as this project was beginning, and as such was very much in a beta stage. By working with the board in a detailed manner and stretching its intended usage, this project served as an impetus for various improvements and bug reports regarding the Bela operating system. For example, an early discovery made through experiments and testing was the Bela's incompatibility with the wireless version of the Beaglebone. This was due to a a pinmuxing conflict with the Bela's analog inputs and the Beaglebone wireless GPIO configuration. The bug report on this issue resulted in the release of a separate Bela image developed specifically for the wireless Beaglebone. Likewise, there were multiple Pd objects whose functionality was less than perfect on the Bela. The [netsend] object, for example, was

[3] The BBB refers to add-on boards such as sound cards or motor drivers, etc. as "capes". This is an analog to the Arduino's "shields", or the Raspberry Pi's "hats".

at first nearly unusable due to the way it interacted with the Bela console. This precipitated Giulio Moro's improvement to the object in Bela's libPd version, which was later made a standard feature in the Bela release. These changes, similar to the Bela's inclusion of eQEP functionality mentioned above, were direct results of the work done throughout the course of this project.

2.10 Summary

This concludes the chapter detailing the design and construction process involved in this project. The project presented unique challenges from a hardware design perspective. The robots are essentially mobile signal processing units, which means they demand hardware related to both locomotion and audio input/output and analysis. For this reason, custom circuitry that handles motion control and low-level audio signal processing had to be incorporated into the design.

The robots' hardware that is not custom is still very new, in some cases still in development. Working with new hardware presented the challenge of discovering, reporting, and fixing bugs. Due also to the projects extensive demands on hardware and software, changes had to be made down to the kernel level of the BBB, and up to the development of customized compilations of libpd and device trees.

The robots' physical design manufacture also required a good of customization in order to include and facilitate all necessary components. In order to house every on-board device, including microphones, multiple power supplies, numerous circuit boards and the like, unique components were fabricated using 3D printing and laser cutting facilities as well as multiple CAD software suites.

Chapter 3

The Networked Method of Robotic Sound Source Localization

This chapter will discuss the implementation and use of networked system of robots in order to locate sound sources in an environment. While the hardware discussed in the previous chapter comprises the physical aspect of this project, the algorithms and programs covered in this chapter are the actual process upon which this method relies. The discussion of Networked Sound Source Localization (NSSL) begins with an overview of the Python programs that run on the robots. Next, the two methods used for DOA determination, cross-correlation and envelope detection, will be covered. Lastly, there will be a description of the interaction that takes place between the robots, focusing on the networking involved in their design.

3.1 Networking

The networked system between the robots and the hubs is multi-faceted. A network consisting of various parts is necessary due to the flow of information that is required not only among the robots and the hub, but also between the different programs running on the robots.

This section provides an overview of the network system design for this project.

A choice between TCP and UDP sockets had to be made early in the development of this project. TCP communication offers error checking and negates the possibility of dropped packets over the network, waiting for transmission completion and status acknowledgement before proceeding to the next packet. While this method offers greater reliability of data transfer, it sacrifices speed for security. UDP, on the other hand, sends out data packets in a constant stream, neglecting error checking and transmission/reception reports. While this can easily result in lost information and dropped packets, the UDP protocol was chosen as speed of transfer is deemed the priority over continuity of data. It was decided that lost packets would almost immediately be replaced with whatever data was next in the transmission queue. As this project relies on reaction that is as close to real-time as possible, UDP made more sense.

As stated, there a number of UDP sockets in operation at all times. These are as follows:

- Local connection between Pd and Python
- Wireless connection from each robot to the hub
- Three wireless sockets on the hub, each transmitting to a specific robot

The various sockets have to be setup in a specific order for the robots to successfully bind to one another. First, the local socket between Pd and feedback.py is established. Once data flows from Pd to feedback.py, the robots begin their routines of moving and listening. The hub program, robutServer.py, starts running as soon as it gets data from the robots. At this point, only RED is moving, while BLUE and GREEN are waiting for RED to acquire the sound signal before they approach.

The robotic network uses an Apple Airport as its wireless router. The Airport creates a private network that has no connection to the Internet. Upon boot, each of the robots automatically connects to the network. The wpa_supplicant·conf file on each of the robots assigns a static IP address to that robot. The same thing happens on the hub when robutServer.py is started, also with its own static IP. When the Python programs are run on the robots and the hub, the UDP

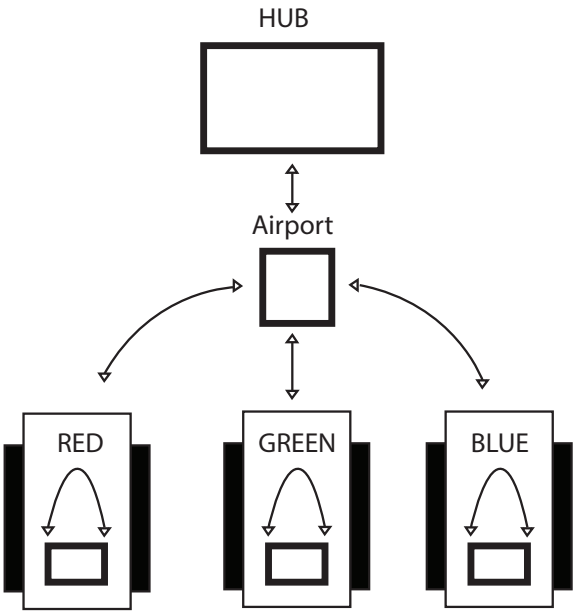


Figure 3.1: Configuration of the robotic network.

sockets listen for a connection then bind to each other, maintaining open socket connections for the duration that the programs are running.

The networks are responsible for transferring all of the data, post signal acquisition, on the Bela Board. In other words, this accounts for all the information that is to be used after the audio signals and envelope detection have been input. The network is thus instrumental in the information dissemination that the project relies on. It is only because of the network that the robots have any awareness of each others' locations, and thus can work together to triangulate the location of sound sources.

3.1.1 Individual Networks

Localhost Network: The first network that is enacted in the robots' startup process is the local network that runs on the BBB. This is a UDP socket that connects the Pd patch to the Python program. This is a one-way stream from Pd to Python, where the [netsend] object transfers data

from the microphones to the Python program. The messages are formatted in Pd to stream as a string of floats separated by an underscore. The Python program then splits the list into usable values and places them in an array where their values can be called and used by the motor control sections of the program.

Robots to Hub: Each robot establishes a connection to the hub computer. The Python program sunning in the hub sets up a UDP socket that listens for connections from any IP address on a given port. The robots broadcast to the hub's IP specifically. The data stream from the robots to the hub consists of each robot's encoder and gyro information. This allows the hub to maintain the heading and X/Y coordinate values for each robot, resulting in the knowledge of their exact locations.

Hub to Robots: The hub, as it receives and updates positional information from each of the robots, sends corresponding data back to each of the robots. To achieve this, the hub establishes a separate socket for each robot, with that robot's IP address and port. The data transferred over these sockets consists of instructions for the robots based on each of their positions. For example, once the lead robot acquires the DOA of a sound source and finds its general location, the hub tells the other two robots to proceed towards the sound at relative angles to the first robot's coordinates.

Thus, in summation, there is one network connection running in the Pd patch that sends data to feedback.py. There are two connections in each robot's Python program: one receiving and one sending. There are then four network connections on the hub: one receiving and three sending.

3.2 Python Programs

There are two Python programs that are necessary for networked sound source localization to function successfully. One runs on the BBB, and the other on a "hub" computer that facilitates communication between the robots, disseminates data, and carries out the dynamic mapping of the

robots' environment. The following sections discuss these programs and their basic composition and functionality.

3.2.1 feedback.py

Each robot runs similar versions of a program that controls its motion, interaction, and portions of the hardware interfacing. The program feedback.py handles data coming from the proximity sensors, encoders and magnetometer, and controls the motors in reaction to that data. For example, if the proximity sensor 1 detects an object, feedback.py tells the motors to stop, turn right, then go forward again. The data it receives from the encoders is used to keep track of "dead reckoning" positional information. feedback.py receives data from the encoders and uses that information to store a running tally of the robot's position. The encoder and positional data is also sent over the network to the hub, which also maintains its own positional information for each robot. feedback.py also receives data from the Pd patch running via the Bela Board. Pd sends information relating to differences in envelope detection and cross-correlation over a UDP socket, which feedback.py then uses as control information to manage the motor system and direct the robot's motion toward the DOA.

feedback.py relies heavily on the `Adafruit_BBIO` (Beaglebone I/O) libraries for both sensor input and motor control. The BBIO libraries handle all of the digital and analog sensor input and PWM output that isn't handled by the Bela Board. This includes the proximity sensors and the digital signals from the ASB's threshold detectors. These libraries are also responsible for setting up the digital outputs of the GPIO pins that control motor direction.

The robots have slightly different functions. There is one lead robot which is responsible for approaching and obtaining the sound's DOA before the other two. This task was assigned to RED. Once this is accomplished, RED sends a message to the hub denoting successful acquisition of the signal. At this point, the hub sends messages to the BLUE and GREEN telling them to approach RED at opposing angles. The routines for the approach are part of the programs running

on BLUE and GREEN, which run a slightly different program called position.py.

3.2.2 position.py

The lead robot is able transmit its position and heading once it has achieved its goal of obtaining a DOA and has honed in on the source's location. Once this data is received by BLUE and GREEN, the program position.py is able to use the data to move the robots to specific positions at a given heading. Position.py uses a feedback motor control system like that of the feedback.py program, translating errors in PWM and encoder data back into the PWM output. However, position.py begins by using the RED robot's position and heading data as it position and angular reference. Position.py uses a variable called 'forwardPosRef', which specifies a certain number of revolutions for the wheels. As an correlation between wheel rotation and forward motion (see equation 3.1e below), the forward position reference can then be set to send the robots to a location corresponding to RED's position, at which point the angular reference for BLUE and GREEN will change to a heading that opposes that of RED.

3.2.3 robutServer.py

The second half of the Python interface necessary for the robots' interaction and localization routines is the server that is run on the hub computer. This program, robutServer.py, receives data from each of the robots over three separate UDP sockets. Each data stream is denoted with a tag referencing the senders identity - GREEN, RED, or BLUE. The data that comes into robutServer.py is information from the robots' encoders and gyroscopes. RobutServer.py then uses this data to maintain positional information pertaining to each of the robots and send it back out to each of them. This way, the robots are each aware of the others' locations, which becomes important for the triangulation routines that occur later in the process. The second function of robutServer.py is to build a dynamic, or evolving, map of the robots' environment. This process

is covered in the next section.

3.2.4 Graphical Mapping

One of the more utilitarian functions of the robotic network is to create a dynamic map of its environment. In order to accomplish this, networking is essential. As the robots move throughout a space, they encounter obstacles which they sense via one of their three on-board proximity sensors. When an obstacle is perceived, the robot that sensed it sends a message to the hub. The hub maintains constant knowledge of each robot's position on space, and therefore knows the robot's position and orientation when the obstacle was encountered. The hub then draws a point on a dynamic map corresponding to that robot's heading.As the robots continue to move, the map is filled in with a basic representation of the objects in the space.

Once the lead robot senses and hones in on the sound source, a representation of the source appears on the map. The initial drawing of the sound source is merely placed in a general area in front of the lead robot. As the other two robots run their routines and arrive at their DOAs, the location of the sound source is drawn in a more exact location. When all of the robots have finished running their routines, the precise sound source location is pinpointed on the map and reported with its X/Y coordinates. The X/Y coordinates of the graphical map are, of course, only relative positions; the location in real space is determined through the relationship between the X/Y coordinates and real space.

The graphical map is created using the Pygame module in Python. Pygame has a number of functions that enable easy drawing with basic shapes. In this instance, each robot is represented by a circle with a line pointing out from it. The line denotes the direction that the robot is heading. As the encoder and gyro information is updated by robutServer.py, corresponding changes are made to the robots' graphical representations. The map also keeps track of obstacles encountered in the environment. Each time a robot's proximity detector is triggered, an obstacle warning is sent to the server. The server adds the location of the obstacle to an array of points and then draws

those points on the screen as squares oriented in the direction of the robot that sensed it.

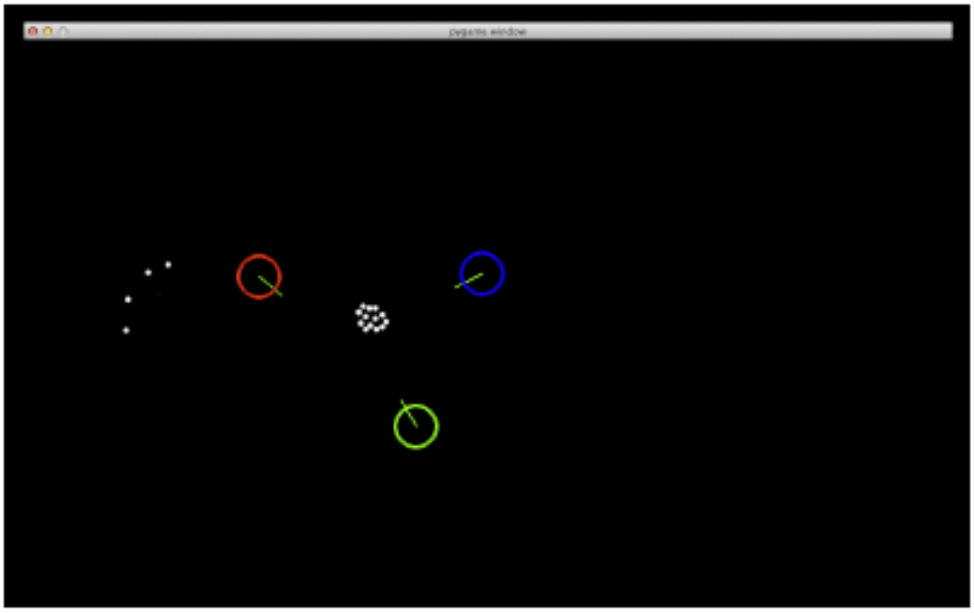


Figure 3.2: An early version of the dynamic graphical mapping that takes place on the hub computer during localization.

3.3 Determining Direction of Arrival

The first step in localizing sound is determining the sound's direction of arrival (DOA). Before the precise triangulation of the sound source can be accomplished, each robot must determine its own DOA for the source. To perform this task, two separate methods were explored throughout the course of this project. The first method, here referred to as cross-correlation difference (CCD), uses a process of sample analysis and correlation of onsets between left and right microphones. The second method, here referred to as envelope detection difference (EDD), uses the Audio Sensor Board's envelope detectors to determine differences in amplitude between the left and right channels. The following sections will detail the processes for each of these methods.

3.3.1 Cross-Correlation Difference

Cross-correlation Difference (CCD) is achieved by sampling the audio signals from the microphones and comparing the peak onsets between the two channels. It most closely resembles the TDOA method of DOA determination discussed in section 1.3. The process runs an FFT analysis on the incoming signal in order to obtain peaks in amplitude. The output signals are then stored in an array. The peak onsets are then compared to determine derivation from a central point in that array. Figures 3.3 and 3.4 show the output of two cross-correlated signals using Matlab's crosscor function, where the center line represents an equal onset for both signals, a positive value represents a signal closer to the right channel, and negative values are closer to the left channel.

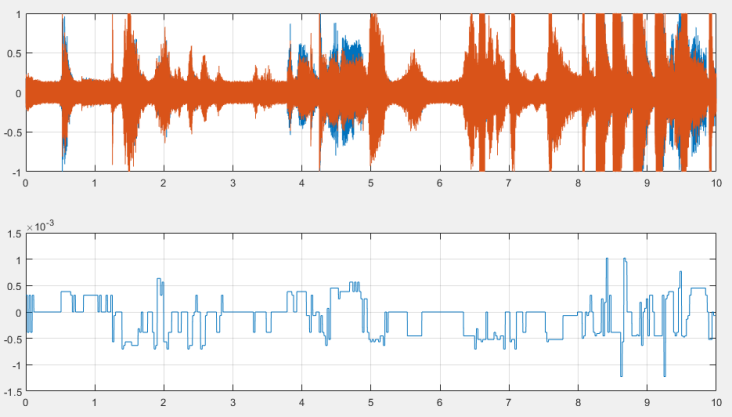


Figure 3.3: The cross-correlated signals between left and right channels.

In this project, the cross-correlation process takes place in a Pd patch running on the Bela Board. The audio signals are provided by the Audio Sensor Board's preamps, which supply line level input to the Bela Board's inputs. The Pd patch used for cross correlation is based on the use of a custom object called [xcorr~]. The [xcorr~] object takes in two inputs - the left and right microphone channels - and performs the cross-correlation routine by running the following processes on the signals:

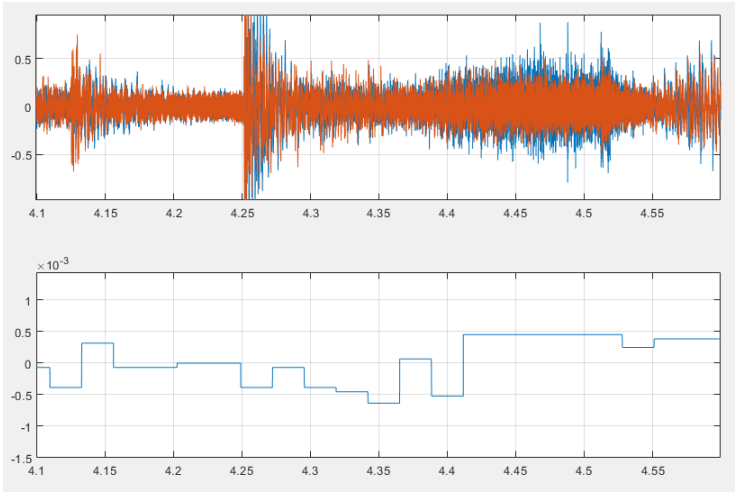


Figure 3.4: Close up image of the cross-correlated signals between left and right channels.

$$Y_1 = Y_R^1 + jY_I^1 \tag{3.1a}$$
$$Y_2 = Y_R^2 + jY_I^2 \tag{3.1b}$$
$$Y_1 * \bar{Y}_2 \tag{3.1c}$$
$$= (Y_R^1 + jY_I^1)(Y_R^2 - jY_I^2) \tag{3.1d}$$
$$= (Y_R^1 * Y_R^2 + Y_I^1 * Y_I^2) + j(Y_I^1 * Y_R^2 - Y_R^1 * Y_I^2) \tag{3.1e}$$
$$\tag{3.1f}$$

In equation 3.1b, Y_1 and Y_2 represent the object's inputs and the resulting processed signal using Pd's mayer_realfft function. The outputs of this function are then cross-correlated using the equation in 3.1e. First, though, the signals are gated by comparing them to a value that comes in through the object's third input. If the signal input is greater than the third inlet value, it is evaluated, otherwise it returns a zero value. This has the effect of gating the input and thus negating extraneous noise. The [xcorr~] object also returns the peak value of the correlated output, as well as the index of that value. The index is returned as an amount of deviation from the center point of the array - in this case, for a 64-point array, the value is the offset from position 32.

Once the signals have been processed by the [xcorr~] object, they can be used by the motor control system. As with the EDD process, the values are translated to a corresponding difference in degrees of offset from the center of the robot. At this point the value can be sent via [netsend] and used by the Python program as the angular velocity error that is fed back into the motor control system to achieve the appropriate angular velocity (see section 3.3.4).

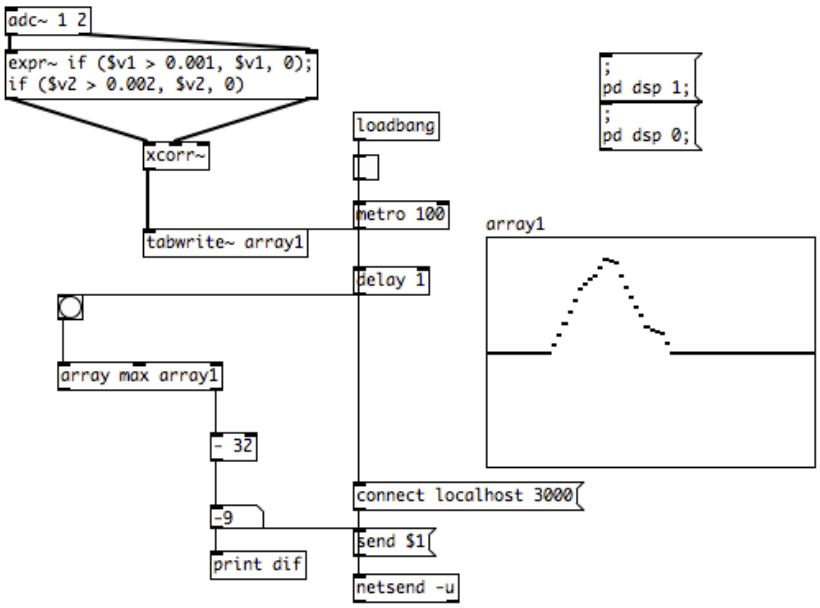


Figure 3.5: shows the [xcorr~] object and its output as it functions in the Pd patch. As shown here, the output of [array max] shows a value of -9, corresponding to a signal that is significantly closer to the right microphone.

3.3.2 Envelope Detection Difference

Envelope Detection Difference (EDD) most closely resembles the Interaural Intensity Difference process discussed in section 1.3. Of the two systems explored in this project, EDD is the less complex, requiring only basic signal analysis. The patch for envelope detection is made fairly simple by the incorporation of the Audio Sensor Board outputs. Since the ASB provides voltage output from each of the channels corresponding to that channel's amplitude, the Pd patch merely reads the voltage with the [adc~] object. The signal then goes through a [lop~]

filter object set to a frequency of 4Hz in order to slow the rate of data transmission - otherwise the Bela Board can become overwhelmed by the influx of data. At this point, the data needs to be calibrated since, with early testing, it was apparent that the microphones had a tendency to produce different outputs even with no sound in the environment. The calibration is done by using a potentiometer on the Analog Calibration Board 2.3.4 to subtract or add an amount to the left microphone until the values are even. Once this done, the values for each channel go through a subpatch that produces a moving average with a window of ten inputs in order to smooth the values over time. At this point, the calibrated and averaged signals are compared by subtracting the left signal from the right. This provides a positive or negative value depending on which microphone has the closer proximity to the sound source. Once this value is determined, it is routed to the [netsend] object and sent to the Python program.

3.3.3 Using the Data

Both the EDD and CCD methods return a value that corresponds to the difference between the left and right channels, providing amplitude and onset differences respectively. Once this value is received, it is used to determine DOA. The left/right channel difference-to-DOA relationship is obtained by creating correlations between the values and actual sound source positions in space. In order to gain an understanding of the relationship between the angle of the audio signal's arrival and the differences in the left and right channels of the microphones, a series of tests were administered. For this process, a sound source consisting of a single sine tone at 440Hz was placed at varying degrees of offset from the center of the two microphones. As the tone played, measurements were taken of the resulting difference in the left and right channels. A 0° onset would account for a 0 sum difference between the two signals. An onset of 90° would account for a difference value of +/- 15. These values were then used as inputs for the angular error variables in the feedback.py and position.py programs that will be discussed in the next section.

3.3.4 Feedback Motor Control System

There are two main parts to the odometry system used in this project: encoder data and magnetometer data. The information gathered from these sensors is combined in the feedback.py program to produce a precise calculation of the robots' positions. As the robots move, the encoder systems keep a running count of wheel revolutions as determined by the PWM duty cycle set in the motor control section of the code. The counts for each of the wheels are different as the robots turn and move towards sound sources or away from obstacles. For example, if a robot turns to the right, the left encoder count will increase slower than the right. Maintaining this count gives a general idea of the robots' positions, but this data can be fraught with errors to due to wheel slippage and other factors (see section 1.6). To correct for these errors, a feedback system is used where accurate counts are maintained and any errors in the counts incorporated back into the system. This system can be describe as follows:

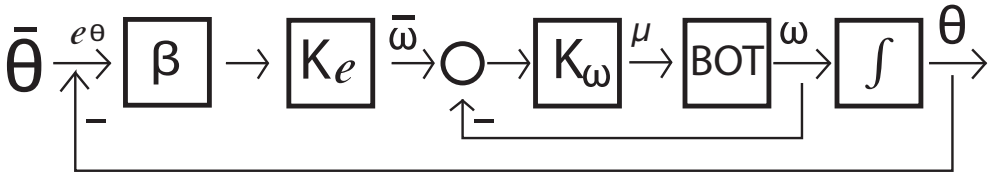


Figure 3.6: is a representation of the feedback system used for motor control, where eθ is the error due to difference in the amplitude of the microphones and $\bar{\theta}$ is the intended heading due to that difference. ω is the current heading

There are multiple steps to determining accurate velocity and heading information from the robots. The first step is obtaining velocity. For this, The encoder values for each wheel are received from the encoders, which are then translated to real measurements in centimeters per second. This value is achieved by measuring the size of the wheels, multiplying by 2π, then multiplying that value by the change in encoder values over time. In order to maintain a constant velocity, changes in the output of this system are then incorporated back into the system as errors and corrected by adding the error value into back into the PWM output (see figure 3.6).

The next step is to achieve a measurement for angular velocity, measured in radians per second. Angular velocity is obtained by checking the difference in rotation between the wheels as output by the encoder values.

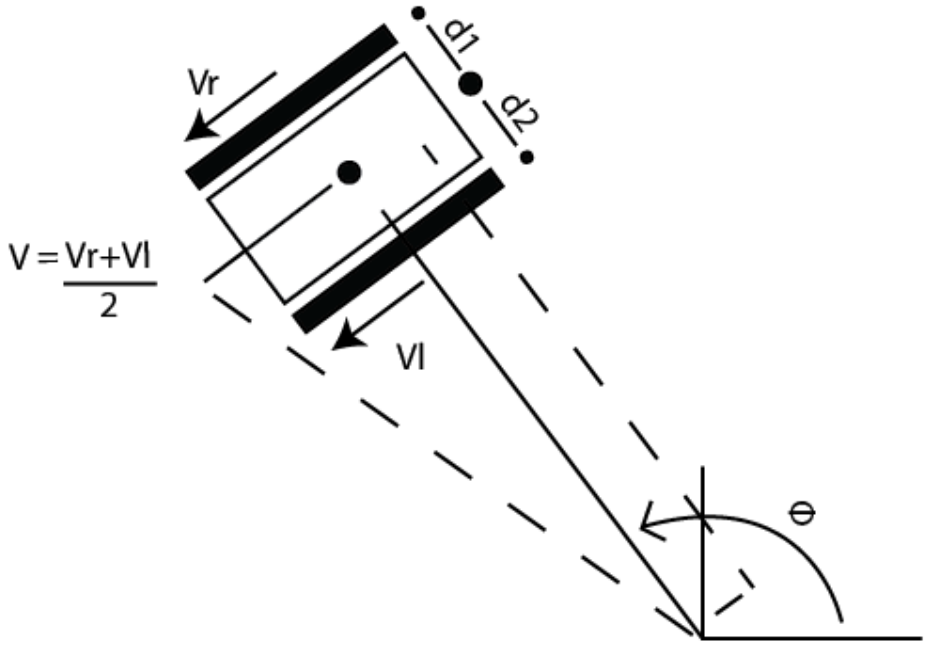


Figure 3.7: is a representation of the velocity and angular velocity calculation.

To check against these errors, the use of a gyroscope is incorporated. The gyroscope, or magnetometer, maintains a running vector for each robot's heading in degrees and minutes. The gyroscope value is also used as a reference for the angular velocity once error is obtained in the form of left/right microphone channel difference. The microphone difference is mapped to a corresponding difference in degrees of derivation from the center of the robot. This value is referred to as the angular velocity error and incorporated into the motor control system so that the angular velocity is altered to match the angular velocity reference.

The motor control system generates a PWM signal with values between 60 and 100, with values below 60 being insufficient to drive the motors. The inputs for the motor control system rely on several inputs. The first input is the microphone difference sent from the Pd patch. This

value is $e\theta$ and is used to determine $\bar{\theta}$, which becomes the robot's desired heading. Once this value is obtained, the robots are able to proceed along a path to the sound source. In so doing, an accurate odometry measurement must be maintained. For this, a feedback system consisting of input from the encoders and the gyro is used.

To calculate velocity, it is necessary to maintain an encoder count for each track, and compare that against the time stamp. Once these velocities are obtained, they are used to calculate forward velocity and angular velocity. Forward and angular velocity are maintained by incorporating errors into the PWM output. The velocities are calculated by:

$$v_r = w_r r \tag{3.2a}$$

$$v_\ell = w_\ell r \tag{3.2b}$$

$$v = \frac{v_r + v_\ell}{2}, w = \frac{v_r - v_\ell}{d} \tag{3.2c}$$

$$v = wR \tag{3.2d}$$

$$v_r = w(R + \frac{d}{2}), v_\ell = w(R - \frac{d}{2}) \tag{3.2e}$$

$$u_1 = v_r + v_\ell = \alpha P_r + \alpha P_\ell \tag{3.2f}$$

$$u_2 = v_r - v_\ell = \alpha P_r + \alpha P_\ell \tag{3.2g}$$

$$v_r = \frac{u_1 + u_2}{2}, v_\ell = \frac{u_1 - u_2}{2} \tag{3.2h}$$

$$P_r = \frac{u_1 + u_2}{2\alpha}, P_\ell = \frac{u_1 - u_2}{2\alpha} \tag{3.2i}$$

Forward velocity and angular velocity are determined by 3.2c. Error correction is shown in 3.2f where forward velocity is maintained as a result of forward velocity reference $\alpha P_r + \alpha P_\ell$ multiplied by velocity error and 3.2g where angular velocity is a result of current angle w multiplied by the angular error given by the difference in values between the left and right microphone channels.

Thus, the PWM calculation is represented by:

$$PWM = \begin{cases} (1 + \frac{p1}{p2} - \frac{p1}{100})u,\ 0 < u \leq p1 \\ p1 + (1 - \frac{p1}{100})u,\ p2 < u \leq 100 \\ -p1 + (1 - \frac{p1}{100})u,\ -100 \leq u < -p \\ 100,\ u > 100 \end{cases}$$

3.4 Triangulation

The triangulation routines for NSSL are based on the outcome of the RED robot's DOA determination. Once RED has obtained its DOA, it sends a message to the hub. When the hub receives this message, it sends a command to BLUE and GREEN that includes RED's coordinates and heading. BLUE and GREEN are then able to use thus information in the position.py program. To make use of this data, the two robots must calculate their distance from RED, BLUE, and GREEN do this by taking RED's total encoder count and subtracting their own count from that value. This value is then fed into the forward position reference variable in position.py, which is the determiner of BLUE and GREEN's forward movement. Added to this value is a +/- correction amount to the left or right of each robot's relative position to RED so that these robots approach the sound source at an angle diametric to RED rather than analogous or parallel to it. This process ensures that the three robots surround the sound source rather than approach it all in a

line along the same heading. As BLUE and GREEN proceed forward, they perform the same cross-correlation or envelope detection method used by RED. However, the feedback system they use allows for errors to the left and right of the sound source in order to correct for the previously mentioned opposing approach vector.

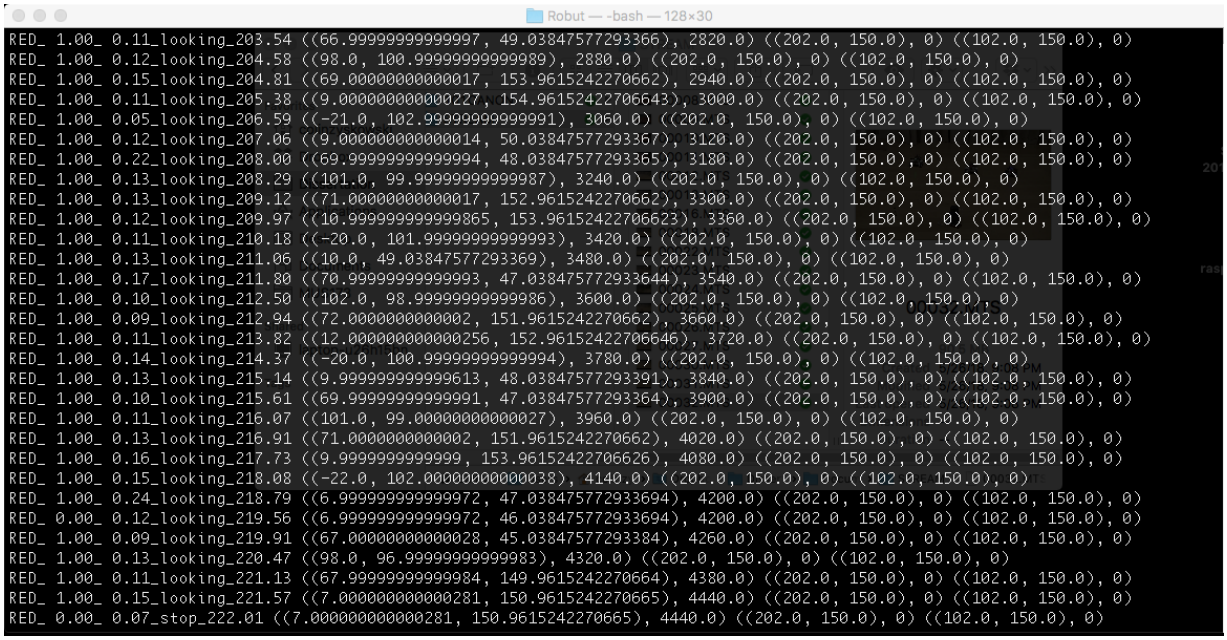


Figure 3.8: An example of the data stream from RED to the hub. This contains information including the robot's name, heading, current task ("looking" or "stop"), and encoder values. Also represented is the coordinate system used by Pygame for the drawing of each robot's position and angle on the graphical map.

The three robots determine they have located the sound source once they share similar positions yet contradictory headings. At this point, all the robots perform a final correction to face exactly toward the source. When the exact DOAs are achieved the robots send their position and heading vectors to the hub. The hub processes this information, allowing the user to calculate the point at which each robot's heading meets the others or the vector in the center of the triangle formed by the robots.

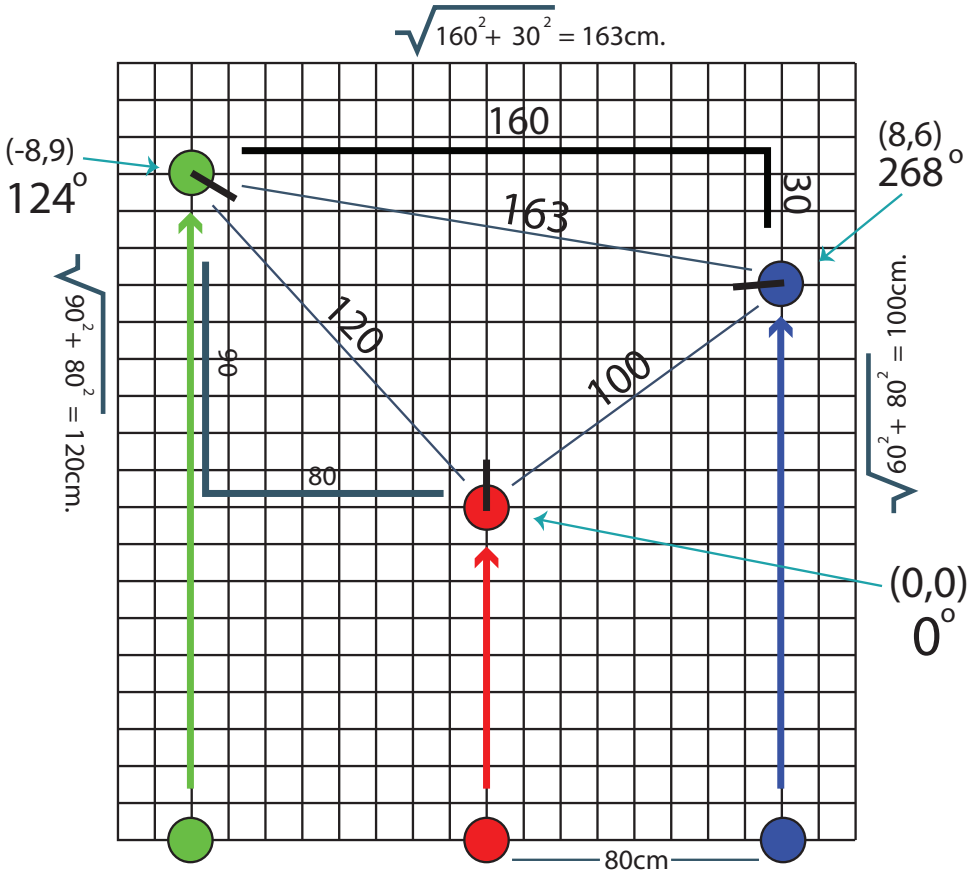


Figure 3.9: Once the robots have each found their DOA and stopped in their final locations, triangulating the sound source becomes a matter of simple geometry.

3.5 User Interaction

Both feedback.py and position.py allow for user interaction through keyboard commands. These commands control variable values that effect the motion of the robots. This practice was used during testing procedures and for calibration of the robots' responses to audio input. For example, in matching the each robot's feedback system as audio pertained to motor control, a specific amount of turn radius had to accompany specific differences in amplitude/correlation between the microphones. This process was made much simpler with the incorporation of user input controls, which allowed various values for both angular and forward velocity to be entered as audio was received. This interaction also greatly simplified the process of matching the robots' actual movements with their represented motion in the graphical map; the robotic motion could

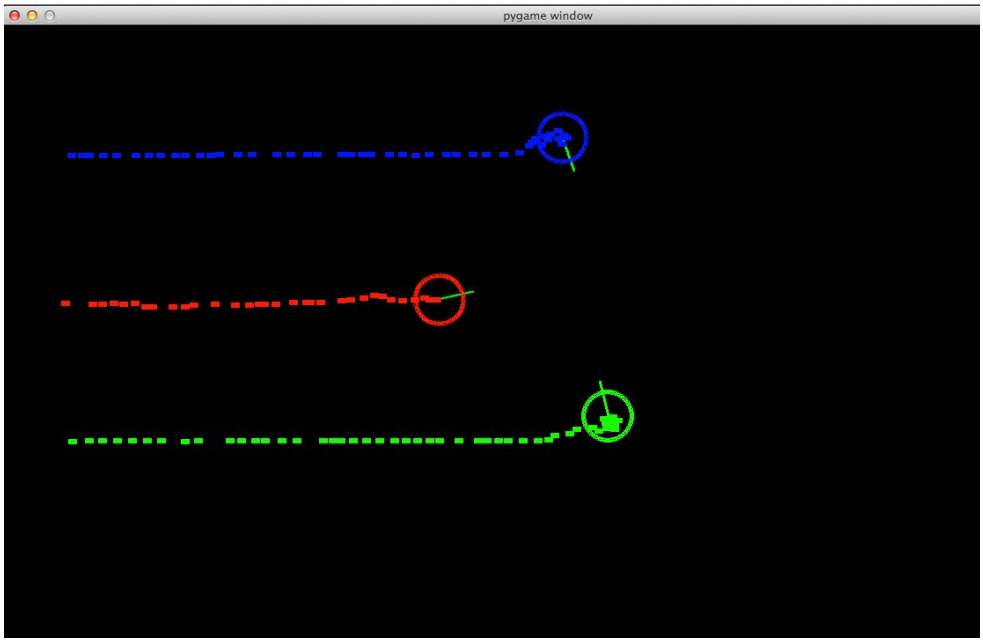


Figure 3.10: The graphical map shows the final positions of the robots as well as the paths they traveled. Note that the RED robot's path wavers as it searches for the sound source, while the GREEN and BLUE paths are much straighter due to their prior knowledge of RED's location.

be controlled as different values were input for variables such as turn radius and forward motion. This allowed for a much closer relationship between the graphical map and actual robotic motion.

3.6 Results

Numerous tests were conducted to assess the efficacy of the sound source localization routines used in this project. This section will discuss the testing procedures as well as the results of the tests.

3.6.1 Testing Procedures

The first tests were done with one robot in a stationary position and a single speaker. The two localization routines would each be run on the robot while its forward reference velocity was set to zero, meaning it could could turn in place, but not move forward at all. This allowed for the angular velocity parameters of the feedback.py program to be set properly. Surprisingly, the

values for these variables could be quite low while maintaining functionality. This allowed for very precise calibration of the angular velocity-to-microphone error correlation which in turn diminished the amount of error in the final localization process, which will be discussed further on.

For further testing, a variety of scenarios were developed. The general setup consisted of a speaker in the middle of an open space, with the robots beginning their processes facing in varying degrees of difference away from that speaker. Tests were first conducted in which the robots all had similar starting points, then other tests in which the robots proceeded from opposing directions towards the speaker in the center of the room. The tests in which the robots all started from the same location proved most successful in terms of the graphical map's output. In this scenario, it was a simple matter to correlate starting points and distances from the real world to the map, as everything essentially began from the same reference point.

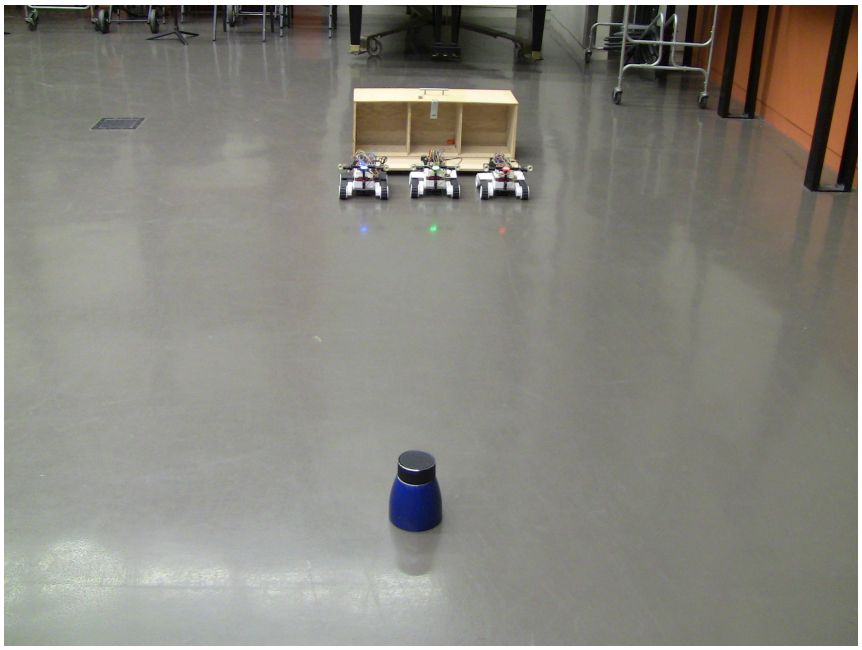


Figure 3.11: Initial tests conducted with all robots at the same starting location.

Because the algorithms running on the hub call for encoder data from the RED robot to be transferred to and used by GREEN and BLUE as their initial traveling distances, the robots would have to begin with similar initial distances from the source for their routines to work

well; if the robots proceeded from vastly different distance from the sound source, they were prone to travel too short or too long a distance for their own localization routines to take effect. Since the idea behind this project was for the robots to locate a sound source with an unknown location, placing the robots at opposing yet equidistant locations seemed a bit like cheating. For practical applications of this project, it also made much more sense to start the robots at the same location, since there are few, if any, situations in which it would be necessary to locate the robots at dissimilar starting points.

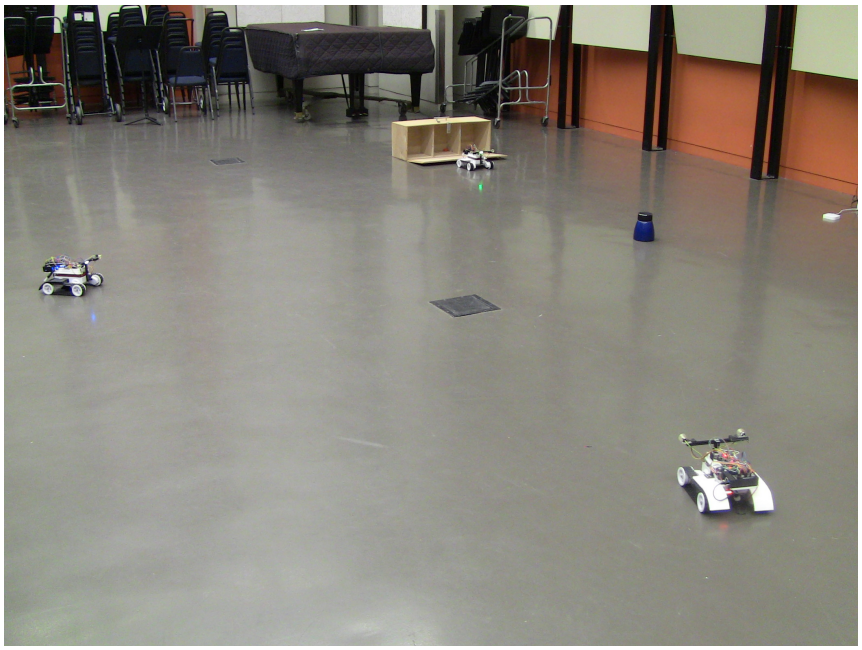


Figure 3.12: Secondary testing included shifting the starting locations of the robots.

A solution to the problem of different starting points presented itself through another series of tests. In another scenario, all three robots ran the feedback.py program. This negated GREEN and BLUE's dependency on encoder data from RED. Instead, all three robots now could localize completely independently of the others. This still presented difficulties in generating precise correlations in the mapping routines, as the graphical representations of the robots depend on awareness of their starting points, but for the localization routines, this proved very successful. All three robots still transmitted encoder and heading data to the hub, which then triangulated the final source location.

A primary limiting factor of the tests is that they were done under very controlled circumstances. The rooms were setup to have no obstacles between the robots and the sound source. The floor was also hardwood or concrete, clean and flat. While the robots were designed to travel over semi-rough terrain (hence the tracks instead of wheels) and up and down inclines and declines, as well as to avoid obstacles, their abilities in these areas were not the focus of the tests. Similarly, the initial point of using proximity detectors was for obstacle avoidance, but they later became useful for determining proximity to the sound source. As the tests conducted here were done to assess the quality of the localization routines, and not obstacle avoidance, it seemed more important to carry them out free of foreign objects because of the effect external objects can have on sound reflection While this situation does not, of course, mimic real-world scenarios, they did provide a good starting, or reference point from which the project can develop. In going forward, a primary point of focus will be to put more energy into building the robustness of the system so it can handle interference such as that created by obstacles and variations in setting.

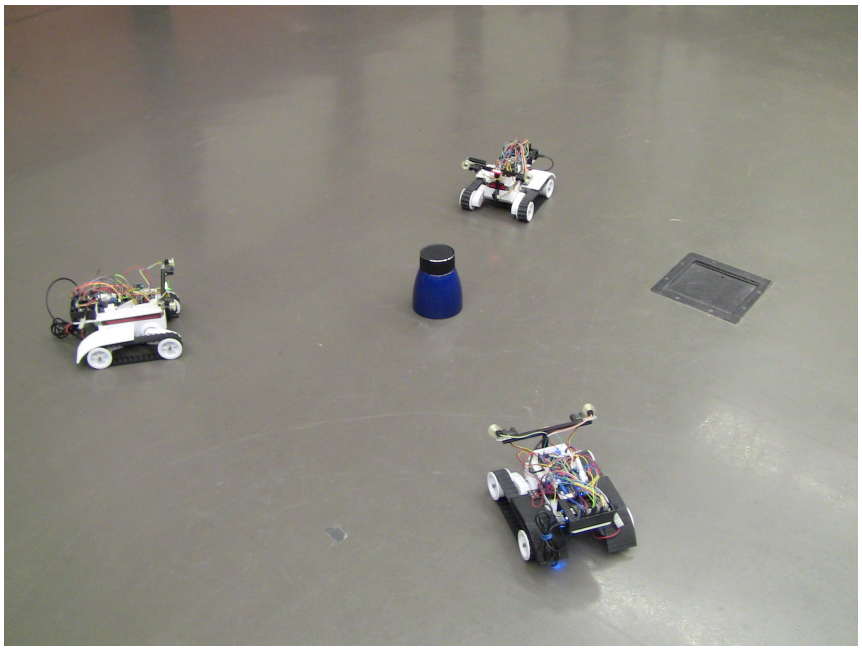


Figure 3.13: Final positions of the robots after initiating procedures from different starting locations, each using the feedback.py program.

3.6.2 Envelope Difference Detection Results

The two sound source localization methods used for this project - EDD and CCD - each had strong points and weak points. Envelope detection proved much more successful when presented with sustained tones rather than intermittent sounds. When attempting to process non-constant noise, the EDD routines were easily overwhelmed. This was in large part because of interference from excess noise from the motors. Even with the incorporation of sound dampening above the wheel wells and around the microphones, motor noise still proved an issue. The robots would often pick up motor noise in between sounds from the speakers and produce microphone error results related to that noise. This had the effect of sending the robots off target from the sound source, sometimes irreparably. However, when it came to constant sounds, the EDD routine worked exceedingly well. Constant tones allowed the robots to ignore motor noise and focus specifically on the speaker. For this to happen, the speaker had to be at sufficient volume, which would necessarily increase with the robots' distance from the source.

Another drawback of the EDD routine was its dependence on very precise calibration of the microphones. This was the initial reason for incorporating the analog calibration board - to enable real-time manipulation of the signals going into the Bela Board in order to ensure they were exactly matched. When system tests were first started, the left and right microphone channels often gave vastly different output values with no discernible sound present, though the reason for this is not known. Rather than redesign or rebuild the Audio Sensor Boards, it was decided to calibrate the inputs. Without precisely matched microphone input values in a zero sound environment, the end result of the EDD routine is inexact.

3.6.3 Cross-Correlation Difference Results

The CCD routine had a couple advantages over the EDD routine, as well as one disadvantage. CCD benefited from the fact that it did not rely on precise microphone calibration.

The values taken in by the [xcorr~] object simply compare peak values between two channels, regardless of the difference in level between those two channels. This meant that the process could be run without calibration needing to be done regularly. The second advantage of this system was its speed - since the [xcorr~] object depended on a relatively small block size (128), it output values several times per second. This actually proved to be too fast for the Python sockets to handle, and thus had to be slowed down with the Pd patch itself. In the end, though, the speed of this method meant that the robots could react to changes in environmental sound quite quickly. This also had the effect of making the CCD method much better suited towards intermittent sound, as their response time while running this process was ~5ms. This fast response made it possible to enable higher forward velocities for the robots, which in turn resulted in faster completion of the task.

The one downside of the CCD method was that the output from Pd could be a bit noisy, meaning the values could be varying slightly from one to the next. The reason for this was probably that the method was incredibly susceptible to differences in sound. Since microphone volume levels, for the most part, did not effect the outcome of the CCD routine, this seemed an unavoidable side effect of a sensitive system. However, a moving average sub-patch was incorporated to even out the values, which helped provide a significantly more level output.

3.6.4 Overall

The localization routines in general worked well. They provided a degree of accuracy that exceeds that of other autonomous robotic systems. The robots were able to locate sound sources to within .5 inches with a negligible error in degrees offset in any direction. This far surpasses common methods of RSSL that do not account for multiple axis determination.

There were, however numerous issues that presented themselves during the testing and documentation process. The first had to do with the network. The numerous UDP sockets, with various servers and clients between the hub and the three robots, did not always function as well

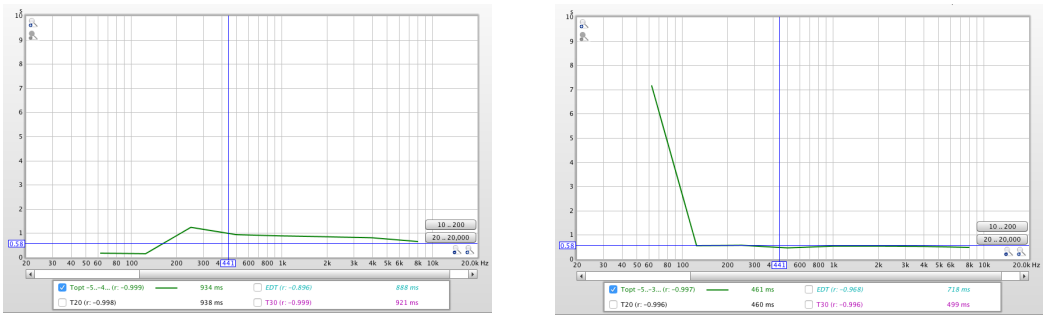
from one network to the next. The home network on which the system was originally created was robust and had no problem handling the flow of data in multiple directions at high speeds. However, the airport that was eventually used as the wireless router and network connection, was not as well-suited to the task. The airport, however, was a necessary feature of the system design; the rooms on campus that were used for final testing and documentation only supplied campus WiFi, which allocated different IP addresses to the robots and hub with each boot up. The robots of course relied on static IP addresses in order to communicate over the network.

The networking issues were not easily surmounted, and often caused large delays in data transmission, which in turn caused long lags in the robots' response time. This made the robots miss cues from the proximity sensors and slowed the feedback process in which the motors reacted to microphone error. In order to negate this adverse side effect of a weak network, a more sturdy and reliable router could be incorporated for a slightly higher cost. Another solution would be to use only networks that allow for static IP address assignment, but this option limits the locations in which the system could work.

Another issue was surface reflection. There were three separate and very different environments in which the system was tested, all with fairly different results. The first was a small room with wood floors and multiple objects such as rugs to prevent reflection. The initial calibration of the feedback.py and position.py programs was accomplished in this room. When the robots were taken to a second, much larger room with concrete floors and no sound dampening surfaces, the robots had a much harder time focusing on the single speaker in the room, as they were also confronted with reverberations and reflections off of the floor and walls. The smooth concrete floor also presented the issue of track slippage. Both of these problems were in large part solved by re-calibrating the input variables in the feedback.py and position.py programs.

Decreasing the values of the angular and forward velocity multipliers had the effect of slowing down response times for the robots, and therefore making it easier for them to, first, move slower and therefore not slip on the concrete, and second, spend a greater amount of time

Figure 3.14: Graphs showing the T60 times of impulse responses in the two main testing areas.
(a) Figure 1 (b) Figure 2



listening before moving. The third room that was used was somewhere between the first two - large and open, but with wooden floors and a good deal of sound dampening material on the walls. In this room, again re-calibration was done and successful implementation of the process accomplished. In the end, the robots seemed to do best, as could be assumed, on non-slippery surfaces in non-reflective environments.

The calibration process for accommodating different room reverberations relied essentially on configuring the response times and magnitudes of the robots' feedback systems as they relate to microphone errors. The T60 times[1] of each room were measured. The higher the T60 time, the slower the response time and magnitude of angular correction had to be. In the larger room, with a T60 of 932ms, the angular velocity had to be set to angVelRef = 1.2, forwardVelRef = .9cm/s. For the smaller room, with curtains drawn across the walls for sound dampening, a T60 of 432ms related to feedback values as high as angVelRef = 2.6, forwardVelRef = 1.5cm/s.

3.6.5 Procedure

The following is a representation of the procedure taken by the robots as they initiate and complete their localization routines.

[1] T60 times refer to the time in which an impulse's reverberation takes to drop by 60dB. For this measurement, impulses were recorded in each testing room, and then measured using the Room EQ Wizard software.

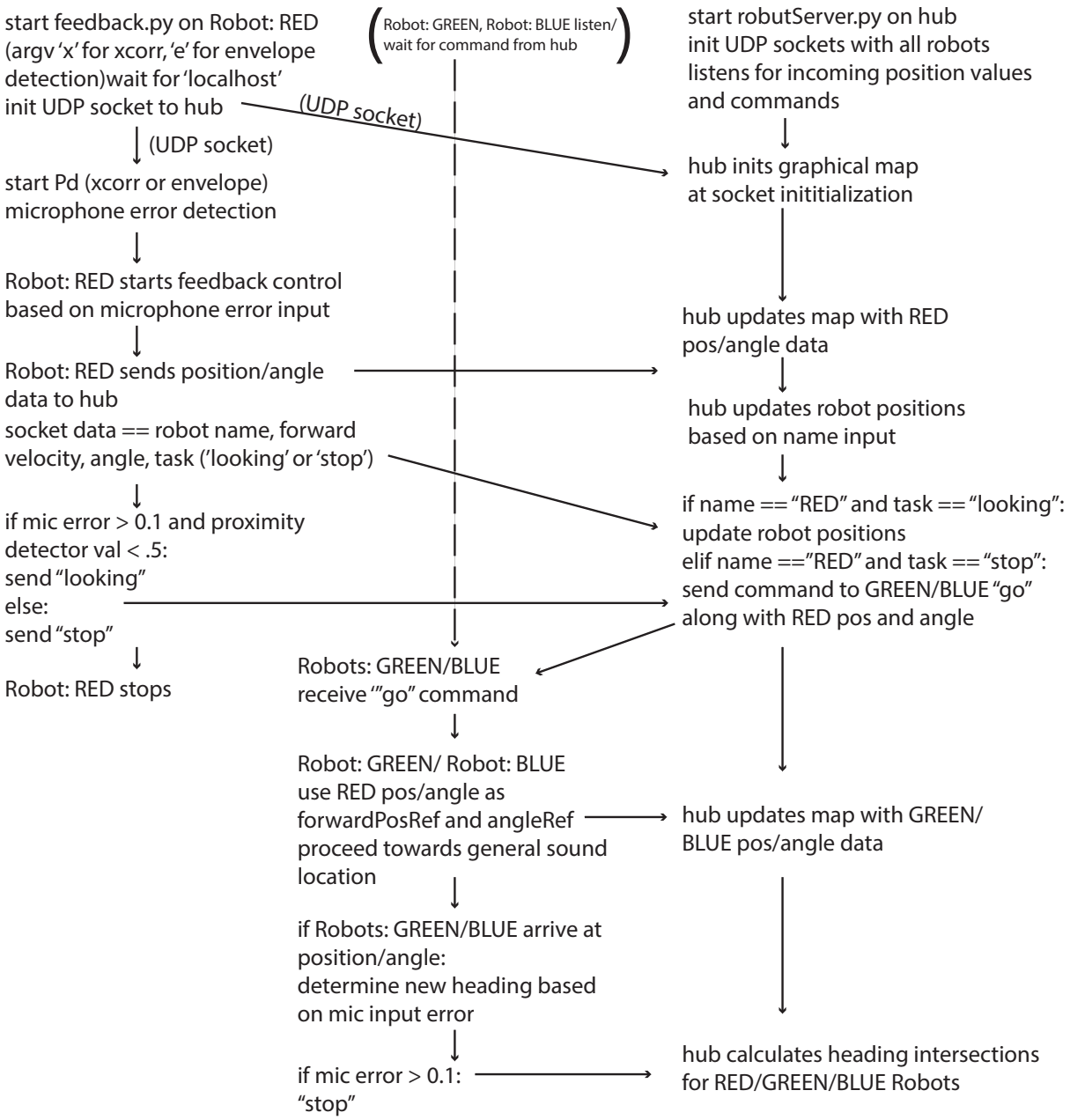


Figure 3.15: A procedural diagram of the actions that occur throughout the course of the RSSL task.

Chapter 4

Conclusions and Future Work

4.1 Future Work

While the outcome of this project was successful, there is room for improvement. Algorithms in the areas of control systems and sound recognition can always be improved by updates in hardware and software. It is certainly conceivable that faster reaction times and more precise localization could be achieved through code optimization. This will, of course, be a goal moving forward. However, there are broader concepts whose inclusion in this project were limited by time and resource constraints. Some of the areas in which future development could occur are:

Allowances for multiple sound sources: The experiments conducted throughout the course of this research were done in fairly contained and controlled environments. This means particularly that the robots were tasked with locating a single sound in a space that was not complex; there were few obstacles for the robots to work around. Allowances were written into the robots' code for navigating obstacles, but in order to account for sound sources placed within a field of numerous obstacles, more complex motion routines would have been necessary. As the general motion and obstacle avoidance routines that run on the robots now make up the simpler parts of the programming, this most likely will not prove a too difficult task.

With some fairly simple additions to the Pd patch and an addition to the Python console interface, it would be possible to select specific frequencies for the robots to locate. This would have the affect of negating sound sources that are not of interest. Similarly, it would be possible to apply narrow filters to input signals in order to focus on only sound of interest. While this would not necessarily solve the issue of multiple sound sources of similar frequencies, it would be useful for tasks such as removing background noise.

Enhanced mapping system: The mapping system in place now is a general representation of the space that the robots inhabit. It relies on relative distance estimations that correspond to real space. In order for the mapping system to prove useful in practical applications, it would be necessary to create exact correlations between mapped and real space. Fortunately, this project does not rely on the mapping system for localization; the mapping system exists as a visual means of determining the progress of the robots as they perform their task. It was thus not necessary to work towards precision in this area. However, it is an area that could benefit from further work.

Increased interactivity: Much could be done to expand the interactive capabilities of the robots, both in regards to each other and to human observers. The robots now sense each other's sound output and are thus able to determine relative positioning. The sound could, though, be used to trigger more intricate actions and reactions. For example, choreographed motion routines such as flocking algorithms could be incorporated. More practically, a complex set of audio triggers could be used to cause reactions. In this sense, it could be possible to develop a system of communication that relies more heavily, or even solely, on audio transmission.

There is also room for improvement in the area of human-robot interaction. At this point, the robots are able to sense humans as obstacles or as sound output devices. It is conceivable however that more input from humans would increase the practicality and usefulness of the project. This could be done in a variety of ways, including a networked user input application, or a speech detection algorithm that enables the robots to receive and react to spoken commands.

More complex musical output: This initial idea for this project - a much different one

than that which is represented in this writing - was for a dynamic sound installation. Over time and because of varying research interests, the project became focused on what is seen as a more practical application of sound in robotics. The robots currently output sound and rely on it for interaction. However, for the output to be more than a functional part of the process and for it to be appropriate for audience enjoyment, it would need to be much more complex. The Pd patch running on the robots now already has methods for controlling and manipulating the audio output. To give it more complexity would be a fairly simple matter involving changes to the sound output section of the patch. There are already controls in place via the analog calibration board that would enable users to alter the output previous to or even during the robots' motion routines. This could easily be expanded to a point where the project could also function equally well as an art installation.

Automation of calibration processes: Another area that could be improved is automation of calibration procedures. The amount of time it took to adjust the feedback system's variables in response to room reverberation was considerable and had to be done for every testing procedure. This process could theoretically be automated and accomplished by the robots themselves. This would be done by having the robots produce impulses, measure that T60 times, then alter the angVelRef and forwardVelRef variables correspondingly.

4.2 Conclusions

This project met its goals and proved a successful proof of concept. The project was able to establish a new and reliable method for robotic sound source localization based on communication and cooperation amongst a small network. In comparison to other robotic sound source localization methods, it was far more accurate in precisely triangulating source locations. The system that was developed for this purpose is a unique one that involves multiple custom circuit boards for both audio and control systems processing. The project also proved

the possibility of incorporating embedded devices in a unique manner. The inclusion of the Beaglebone Black, the Bela Board, and the various sensors, circuits, and power supplies is distinct from all other localization systems, and proves the possibility of accomplishing this task in a low-cost, effective way with fairly minimal equipment. While numerous issues appeared that negatively effected the outcome of the project, these could be solved by switching existing components with slightly more robust and slightly more expensive components, as in the case of the wireless router.

The documentation for this project will hopefully provide guidance for similar projects. All code used on the robots and the hub, including the Python programs and Pure Data patches, as well as 3D CAD designs and circuit board schematics, can be found at `https://github.com/czyskows/NetworkedRobots`.

Appendix A

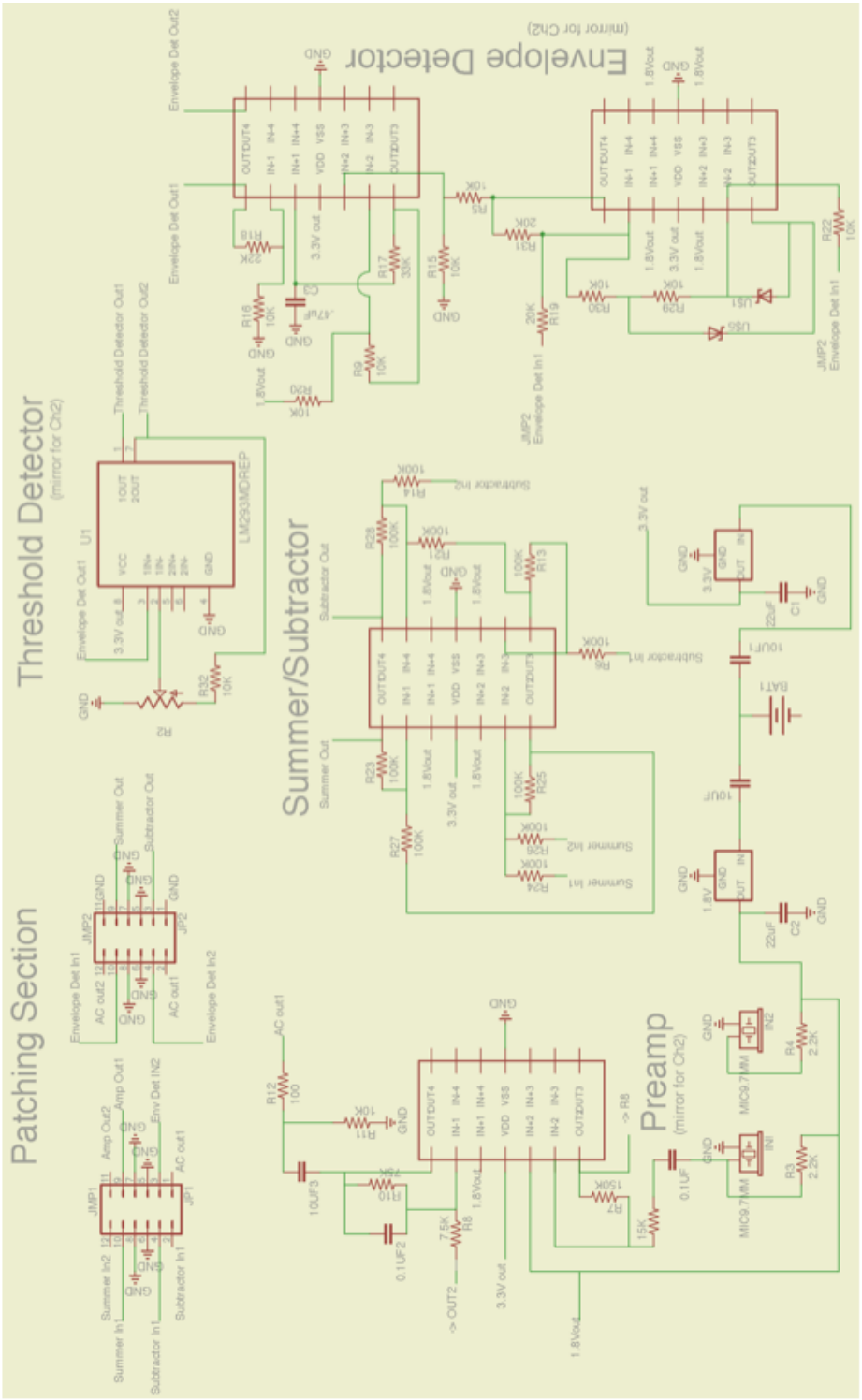


Figure A.1: Full Audio Circuit Board Schematic

Bibliography

[AH11] Farouk Azizi and Nasser Houshangi. Mobile robot position determination. *Recent Advances in Mobile Robotics*, 2011.

[Arg15] P. Danes P. Soueres Argentieri, S. A survey on sound source localization in robotics: From binaural to array processing methods. *Journal of Computer Speech and Language*, 34:87–112, 2015.

[BB10] Roberto Bresin Birgitta Burger. Communication of musical expression by means of mobile robot gestures. *Journal on Multimodal User Interfaces*, Volume 3(Issue 12, pp 109118), mar 2010.

[Bre16] Birgitta BurgerRoberto Bresin. A survey of robotic musicianship. *Communications of the ACM*, 59(Issue 12):109–118, apr 2016.

[Coo] Justin Cooper. Adafruit beaglebone i/o python api. `https://github.com/adafruit/adafruit-beaglebone-io-python`.

[CUI16] CUI Inc. *MODEL: CMC-4015-40T ELECTRET CONDENSER MICROPHONE*, apr 2016.

[dO18] Mauricio de Oliveira. Rcmpupy. `https://github.com/mcdeoliveira/rcmpupy`, apr 2018.

[GS] Stergios I. Roumeliotis Goel, Puneet and Gaurav S. Sukhatme. *Robot Localization Using Relative and Absolute Position Estimates*.

[HA05] Nasser Houshangi and Farouk Azizi. Accurate mobile robot position determination using unscented kalman filter. *Sound and Vibration Journal*, 2005.

[Hon13] Honeywell Internation Inc. *3-Axis Digital Compass IC HMC5883L*, 900405 rev e edition, feb 2013.

[Iri95] Robert Eiichi Irie. *Robust Sound Localization: An Application of an Auditory Perception System for a Humanoid Robot*. Masters thesis, Massachusetts Institute of Technology, June 1995.

[JH74] D. Wright J. Hebrank. Spectral cues used in the localization of sound sources on the median plane. *Journal of the Acoustical Society of America*, 6, 1974.

[LJ10] Filip Deblauwe Lanslots, Jeroen and Karl Janssens. Selecting sound source localization techniques for industrial applications. *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, 2010.

[LK07] Jong Suk Choi Lim, Yoon Seob and Mun-Sang Kim. Probabilistic sound source localization. *Proceedings of the Internation Conference on Control, Automation and Systems*, pages 20–28, 2007.

[Mor18] Giulio Moro. Running puredata patches on bela. `https://github.com/BelaPlatform/Bela/wiki/Running-Puredata-patches-on-Bela`, Jan 2018.

[MW04] Harry Erwin Murray, John C. and Stefan Wermter. Robotic sound-source localization and tracking using interaural time difference and cross- correlation. *Proceedings of the NeuroBotics Workshop*, pages 30–34, 2004.

[Pur99] About pure data. `https://puredata.info/`, 1999.

[SB01] Shagra Shoval and Johann Borenstein. Measuring the relative position and orientation between two mobile robots with binaural sonar. *Proceedings for the 9th International Topical Meeting on Robotics and Remote Systems*, pages 1–4, 2001.

[Sha06] Sharp Corporation. *GP2Y0A02YK0F Distance Measuring Sensor Unit*, e4-a00101en edition, jan 2006.

[Tex08] Texas Instruments. *TMS320x2833x, 2823x Enhanced Quadrature Encoder Pulse (eQEP) Module*, sprug05a edition, aug 2008.

[VL07] Francois Michaud Jean Rouat Valin, Jean-Mark and Dominic Letourneau. Robust localization and tracking of simulataneous moving sound sources using beamforming and particle filtering. *Proceedings of the Conference on Robotics and Autonomous Systems*, pages 1–4, 2007.