Stereo Voice Detection and Direction Estimation in Background Music or Noise for Robot Control

Undergraduate Thesis

Submitted in partial fulfillment of the requirements of
BITS F421 Thesis

By

Samyukta Ramnath
ID No. 2012A3B5038G

Under the supervision of:

Prof. Dr-Ing. Gerald Schuller
&
Dr. Amalin Prince

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, GOA CAMPUS
December 2016

Declaration of Authorship

I, Samyukta RAMNATH, declare that this Undergraduate Thesis titled, 'Stereo Voice Detection and Direction Estimation in Background Music or Noise for Robot Control' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

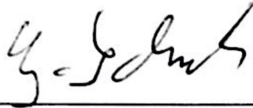
Signed: S Ramnath (SAMYUKTA RAMNATH)

Date: 8.12.2016

Scanned by CamScanner

Certificate

This is to certify that the thesis entitled, "*Stereo Voice Detection and Direction Estimation in Background Music or Noise for Robot Control*" and submitted by Samyukta RAMNATH ID No. 2012A3B5038G in partial fulfillment of the requirements of BITS F421 Thesis embodies the work done by her under my supervision.



Supervisor	*Co-Supervisor*
Dr. Gerald SCHULLER	Dr. Amalin PRINCE
Faculty,	Professor,
Ilmenau University of Technology	BITS-Pilani Goa Campus
Date:	Date:

Scanned by CamScanner

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, GOA CAMPUS

Abstract

Bachelor of Engineering (Hons.)

Stereo Voice Detection and Direction Estimation in Background Music or Noise for Robot Control

by Samyukta RAMNATH

The study aims at implementing Stereo Voice Detection and Direction Estimation in Background Noise or Music for Robot Control. It has two parts - Voice Detection (in noise or in Music), and Acoustic Source Localization. The problem of Acoustic Source Localization is to identify the location of a single source. This was accomplished using a vacuum cleaner robot (Roomba), which communicated with a Raspberry Pi via a serial interface. The time difference between the left and right channel was calculated to find the angle of arrival. The problem of localization in 360^0 was solved by rotating the Roomba by a small angle and listening again, assuming that the source is in the same place. The goal of voice detection in this project was to distinguish voice from other kinds of sound such as noise or music, and have the Roomba move towards the voice when other kinds of sound are present. The task of distinguishing tonal sounds from noise was accomplished by using a Spectral Flatness Measure. The task of separating voice from music is attempted by using frequency cut-off in the form of a low-pass filter, on the left and right channels of the stereo microphone. Feature extraction and classification using a Support Vector Machine was found to be too slow to be a viable method to run on the Raspberry Pi. Future work could consider using more robust rule-based methods that are computationally more viable than a simple frequency cutoff.

Acknowledgements

The author is grateful to Dr. Gerald Schuller, Ilmenau University of Technology, for providing the logistical support to undertake the following study, the Academic Research Division of BITS Pilani KK Birla Goa campus, and Dr. Amalin Prince for their support throughout the process.

Contents

Abstract iii

Acknowledgements iv

Contents v

List of Figures vii

Abbreviations viii

1 Introduction 1
- 1.1 Problem Statement . . . 1
- 1.2 Constraints and Requirements . . . 2

2 Literature Review 4

3 Theory 6
- 3.1 Short Term Fourier Transform . . . 6
- 3.2 Spectral Flatness (Tonality Coefficient) . . . 9
- 3.3 Cepstral Analysis . . . 10
  - 3.3.1 Mel-Frequency Cepstrum . . . 10
- 3.4 Acoustic Source Localization . . . 11
  - 3.4.1 Frequency limitations of two microphone array . . . 12
  - 3.4.2 Finding delay between two signals . . . 12
- 3.5 Speech-Music separation . . . 13
  - 3.5.1 Speech-Music Discrimination using Support Vector Machine and Feature extraction . . . 14
  - 3.5.2 Speech-Music Separation . . . 15

4 Methodology 19
- 4.1 Setting up the Raspberry Pi . . . 19
  - 4.1.1 Enable SSH . . . 19
  - 4.1.2 Connecting to the Pi via SSH, using Ethernet cable . . . 20
  - 4.1.3 Setting up WiFi on the Pi . . . 21
  - 4.1.4 Sending a Mail with the wlan0 IP address from the RPi at start-up . . . 22
  - 4.1.5 Syncing the Raspberry Pi with a home PC . . . 23

4.2 Interfacing the Roomba with the Raspberry Pi 23
4.2.1 Moving the Roomba 24
4.2.2 Setup 24
4.3 Audio Source Localization 24
4.4 Speech-Music Separation 25
4.4.1 Speech-Music Discrimination 26
4.4.2 Speech-Music Separation 28

5 Results 30

6 Conclusion 31

7 References 32

8 Appendix 35
8.1 Voice Detection and Separation 35
8.1.1 Frame-Wise Fundamental Freqency 35
8.1.2 Useful Functions used in Feature extraction 36
8.2 Acoustic Source Localization 43
8.2.1 Moving the Roomba 43
8.2.2 Functions to Move the Roomba 47

List of Figures

3.1 Diagram to calculate angle of arrival of acoustic source. Image source : Direction of Arrival Estimation and Localization Using Acoustic Sensor Arrays,Vitaliy Kunin, Marcos Turqueti, Jafar Saniie, Erdal Oruklu 11

4.1 The Setup 25
4.2 Top View of Roomba - before rotation 26
4.3 Top View of Roomba - after rotation 27
4.4 Signal classification by feature extraction with the given features, using the multi-class classifier from Python, on a saved wave file, on a computer with higher processing power. 28

Abbreviations

GCC **G**eneralized **C**ross **C**orrelation
SRP **S**teered **P**ower **R**esponse
FFT **F**ast **F**ourier **T**ransform
STFT **S**hort **T**erm **F**ourier **T**ransform
DFT **D**iscrete **F**ourier **T**ransform
SFM **S**pectral **F**latness **C**oefficient
MFC **M**el **F**requency **C**epstrum
MFC **M**el **F**requency **C**epstral **C**oefficient
DCT **D**iscrete **C**osine **T**ransform
ZCR **Z**ero **C**rossing **R**ate
SVM **S**upport **V**ector **M**achine
NMF **N**on-negative **M**atrix **F**actorization
RPi **R**aspberry **Pi**

I would like to dedicate this thesis to my family and close friends for their unconditional support, and Dr. Gerald Schuller, who presented me the opportunity to work under his supervision.

Chapter 1

Introduction

1.1 Problem Statement

The problem statement to solve is :Stereo Voice Detection and Direction Estimation in Background Noise or Music for Robot Control. It has two parts - Voice Detection (in noise or in Music), and Acoustic Source Localization.

The problem of Acoustic Source Localizaton is to identify the location of a single source. This will be accomplished using a vacuum cleaner robot (Roomba), which communicates with a Raspberry Pi via a serial interface. A stereo microphone set (two microphones, with left and right channel), is connected to the Raspberry Pi and takes continous audio input into its right and left channels. The time difference between the left and right microphone channel is calculated, and the angle will be found in that way. The problem of localization in 360° can be solved by rotating the Roomba by a small angle and listening again, assuming that the source is in the same place.

The goal of voice detection in this project is to distinguish voice from other kinds of sound such as noise or music, and have the Roomba move towards the voice when other kinds of sound are present. The task of distinguishing tonal sounds from noise will be accomplished by using a Spectral Flatness Measure, and the task of distinguishing voice from music will be accomplished by using a low-pass filter which would suppress the higher frequencies of music (assuming that music of relatively higher frequencies is used, such as violin music).

There are certain difficulties faced in the real world, which must be considered while performing acoustic source localization in general, and latency and computational aspects which become all the more important when using a platform with limited computational capacity, such as the

Raspberry Pi.

The basic goal of this thesis is to have a system that iteratively changes its direction towards the source. Precision is not as important as the ability of the robot to eventually reach the source of sound, in a real-world environment which would entail noise and reverberations. Low-complexity is a requirement, since the platform used is a Raspberry Pi.

1.2 Constraints and Requirements

There are some constraints and points that should be considered while approaching the problem of acoustic source localization with robots. These can be listed as referred to [1]:

- *Echoes and Reverberation* The effect of reverberation is to confuse the localization algorithm, as the sound reflected off a surface from a frame could reach the microphone and interfere with the next frame of audio, and thus cause an inaccurate calculation of time delay between the audio reaching each microphone. Thus, in very reflective environments, the algorithm would be expected to have low performance. Literature mentions the use of reverberation filters ([2]); however, in this thesis, a simple power threshold has been used, where there is a check that the power in the left and right channel is comparable. This is using the fact that there will be some attenuation in the signal post-reflection.
- *Noise* High noise levels reduce the Signal-to-Noise Ratio of the signal and makes the algorithm less accurate. Noise suppression algorithms are available in literature, ([3]), but in this thesis, a simple power threshold has been used. The power of each frame has been computed. The minimum power amongst all frames so far has been computed, and this has been assumed as the background noise power level. A ratio of the current power to the background power gives a way to distinguish between stray noise and an acoustic event.
- *Source Specificity* It would be desirable to have the robot move towards a specific acoustic source, and thus, have the power to distinguish between and separate different acoustic sources. Existing methods to do so have referenced methods such as Non-negative Matrix Factorization, which are fairly computationally expensive. The approach used in this thesis is to have a simple frequency cutoff, assuming that each source occupies a specific part of the spectrum of the frame.
- *Latency* is defined as the time difference between stimulus and the response to the stimulus. It is essential for the system to have a low latency for any practical application, else the algorithm would take too long and thus be too discontinuous to be of any practical use.

- *Computational Expense* The amount of computational expense acceptable in the system depends on the platform used. For a robot, limited hardware capabilities mean that algorithms should try and reduce the computations performed. This is particularly true on the Raspberry Pi, which has limited real-time computational capabilities. There is a trade-off between computational power and accuracy, and thus, a suitable algorithm should be used, which ensures acceptable accuracy.

Chapter 2

Literature Review

Previous approaches to sound source localization have extensively studied algorithms for localization and separation, both on platforms with vast computational resources and those with limited computational power. Some approaches have taken inspiration from the human hearing system, which is binaural and uses pinnae to give additional cues about direction [4]. Further approaches have studied the effect of the shape of the human head on localization, and have attempted localization with a humanoid shape [5]. Some literature has gone beyond binaural source localization, and have used multiple microphones to localize a source in 3 dimensions with high accuracy [6]. The application of source localization in ground robots generally consider localization in up-to 2 dimensions - that is, localization in one plane is enough for most robot applications, since most of the robots in literature on sound source localization move on the ground. Two microphones can only localize a sound in one plane, with an inherent ambiguity present in whether the sound is coming from front or from behind the robot. This can be removed either by using three non-linear microphones ([7]), or by using two microphones, listening and finding an angle, rotating the robot by some degrees, listening again and finding the correct direction of the sound in 360°. One attempt to take inspiration from the binaural human system mentions this method, to resolve the ambiguity between the front and back position of sound ([8]). This approach, however, used a robot with a higher processing power than the Raspberry Pi, and does not attempt source separation. One very unique approach was to use a single microphone and a pinna-like structure to learn the direction of sound in 3 dimensions [9].

The methods described so far have used omnidirectional microphones (microphones which record the same signal in all spatial directions). Literature extensively describes an algorithm called Generalized Cross-Correlation (GCC) in which the delay estimate is obtained as the time-lag which maximizes the cross-correlation between filtered versions of the received signals [10]. Another method is the Steered Response Power (SRP). The Steered Response is a function which generally used to aim a beamformer, which acoustically focuses the array to a particular

position or direction in space [11]. The SRP algorithm has been shown to be more accurate than the GCC algorithm, but often at a high computational expense (sec. 8.1, [11]).

Chapter 3

Theory

The following theory would be a useful background in the techniques used in the project. Mathematical techniques such as the Short Term Fourier Transform, Discrete Cosine Transform, and the extraction of Mel-Frequency Coefficients have been detailed, along with some information about classification of data using Support Vector Machines.

3.1 Short Term Fourier Transform

The signal is taken in short windows of 20 ms or so (wherein the spectral properties of the sound are assumed to be static)
A Fast Fourier Transform (FFT) is performed with a suitable length such that a reasonable frequency resolution is achieved. The steps in taking the Short Term Fourier Transform (STFT) are :

1. Length M of the input buffer is chosen so as to obtain about a 20 ms piece of audio, since the spectral characteristics of the audio are assumed to be static during this period. x_m is the mth frame of the input signal.

2. A window of length M is chosen and multiplied with the part of the input signal. Without a window, the signal piece would abruptly change at the edges of the signal pieces (they would drop to zero abruptly at each end). This would be interpreted as increased components at high frequency in the spectrum, and are as such artifacts that arise from the act of segmenting the signal. Thus, window functions such as a Kaiser, Hamming, Hanning or Blackman window which smoothly go to zero at each end, reduce these artifacts and get rid of the false high-frequency elements in the spectrum. [12]

$$M_h = \frac{M-1}{2} \tag{3.1}$$

$$x_m{}^w[n] = x_m[n]w[n], \qquad n = -M_h, ..., M_h \tag{3.2}$$

Where the indices going from $= M_h$ to M_h is one way of having the notation. The indices could also have gone from 0 to $N-1$.

3. Find the power of 2 larger than M, and this will be the length N of the FFT. The signal is zero padded in the front and back to bring the length to N, in order to make the FFT computation more efficient. Thus, the signal is padded with $N - M/2$ zeros before and after the signal. (eqn. 8.5, [12])

$$x_m{}^{w,p}[n] = \begin{cases} x_m{}^w[n], & \text{if} \qquad |n| \leq M_h \\ 0, & \text{if} \qquad M_H < n \leq \frac{N}{2} - 1 \\ 0, & \text{if} \qquad \frac{-N}{2} \leq n < -M_h \end{cases} \tag{3.3}$$

4. Finally, the Discrete Fourier Transform (DFT) (or practically, the Fast Fourier Transform or FFT), is taken of this frame, and the content is saved in a spectrogram (or a frame-wise spectrum) of the signal. (eqn. 8.6, [12])

$$X_m{}^{w,p}[w_k] = \sum_{n=-N/2}^{N/2-1} x_m{}^{w,p}[n]e^{-j2\pi k/N} \tag{3.4}$$

Thus, we can see the time-varying spectral content of the signal.

5. Time normalization may be removed by using a linear phase term: (eqn 8.7, [12])

$$X_m^{\prime w,p}[w_k] = e^{-jw_k mR} X_m^{w,p}[w_k] \tag{3.5}$$

6. The window is moved ahead by a 'hop size' such that there is some overlap between frames. the choice of hop size is governed by the amount of accuracy required and the computational ability available. In general, it seems prudent to use a hop-size such that the spectral properties of the signal do not change too much as the hop is made. [13]

The STFT has a few drawbacks due to the limits of time-frequency resolution. The optimal window length is highly dependent on the spectral nature of the signal. For example, a signal with many low-frequency components would require a longer window to capture the low-frequency. A short window would not capture enough information to correctly identify the spectrum of the low-frequency segment. Furthermore, a long window would not be suitable for a high-frequency pulse or burst, as it would be difficult to correctly localize the signal in time. This is referred to as the time-frequency uncertainty principle - wherein, if we use a long window, we would get sufficient information to accurately calculate the spectrum, but we would lose resolution in time, and if we used a smaller window, we would be sure of the time-localization of the events in the signal, but would not be able to accurately calculate the spectral content. This trade-off is inherent in the STFT method.(chapter 10: Fourier Analysis of Non-Stationary Signals, page 725, [15])

3.2 Spectral Flatness (Tonality Coefficient)

One objective of the work was to distinguish between a sound such as voice or music, which has it's spectral energy concentrated at certain particular frequencies, and a sound such as white noise, which has it's spectral energy spread over most of the spectrum. One way of doing this is to compute the Spectral Flatness Coefficient over the frequency spectrum for each window.

Consider a signal x of length N samples, framed into m segments of size N/m.
The Spectral Flatness Measure (SFM) is then defined as: [16]

$$SFM = log_{10}[\frac{AM(X(m))}{GM(X(m))}] \tag{3.6}$$

for the mth frame of the signal.

The arithmetic mean of a series of numbers is always greater than the geometric mean of those numbers. The arithmetic mean is equal to the geometric mean if the numbers are all equal to one another.

Thus, the ratio of the arithmetic mean of the amplitudes at each frequency over the spectrum of a window and the geometric mean of the same, is a measure of how 'flat' the spectrum of sound is. If the Flatness Coefficient is high, the sound is spectrally flat and is closer to noise. If it is low, the sound is likely to have a spectral peak and could be a pure tone or a tone with harmonics (voice or music).

3.3 Cepstral Analysis

The Cepstrum is the log spectrum of the magnitude of the frequency spectrum.

in other words, for a signal x(t), whose fourier transform is denoted by X(w), [18]

$$C(q) = FFT(log(|X(w)|)) \tag{3.7}$$

In a certain way, it can be interpreted as the spectrum of a spectrum. It usually gives the fundamental peak in a frequency spectrum.

The independent variable of a cepstrum is referred to as quefrency, which is a measure of time, but not in the sense of a signal in the time domain. For example, suppose a tone sampled at 44100 Hz has a spectrum with a fundamental peak at 100 Hz, and harmonics at 200 Hz, 300 Hz, 400 Hz and so on. The cepstrum would show a peak at 44100/100 = 441 samples, corresponding to a peak at 100 Hz.

3.3.1 Mel-Frequency Cepstrum

The Mel-Frequency Cepstrum (MFC) is a representation of the short-term spectrum of sound, based on the spectrum of a log magnitude spectrum on a non-linear mel scale of frequency. Mel Frequency Cepstral Coefficients (MFCCs) are obtained by taking the MFC of a signal.

Steps to Compute MFCCs: [19]

- Divide the signal into frames
- Take the Fourier transform of each frame
- The frequencies are mapped onto the mel scale, using N triangular windows
- Take the log of sum of the powers at each of the mel frequencies
- Take the N point DCT of the log spectrum
- The amplitudes of the resulting spectrum are the MFCCs

The frequency resolution of the human ear is better at lower frequencies than at higher frequencies. The motivation behind using the mel scale is that the human ear works on a non-linear scale.

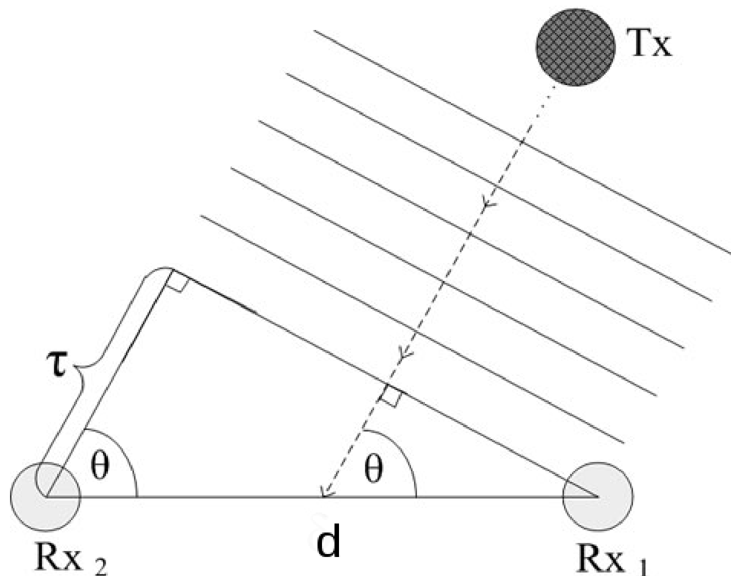


FIGURE 3.1: Diagram to calculate angle of arrival of acoustic source.
Image source : Direction of Arrival Estimation and Localization Using Acoustic Sensor Arrays,Vitaliy Kunin, Marcos Turqueti, Jafar Saniie, Erdal Oruklu

3.4 Acoustic Source Localization

The aim of the project is to enable the Roomba to move towards a single voice, ignoring other sources of sound such as music and such. The task of localizing the acoustic source can be done in many ways. Previous approaches to this task have used linear microphone arrays, in order to increase the precision in localizing the sound. In order to localize a sound in 3 dimensions, one would have to use 4 microphones, where 3 microphones would be kept at the vertices of an equilateral triangle, and the pair-wise time difference calculated to localise the sound in 360^0 in the horizontal plane, and the fourth microphone would be placed at a height , forming an overall tetrahedral shape, to locate the sound in 3 dimensions [20].

Machine Learning approaches could also be used here. A method is described in which a single microphone is used along with a pinna, to learn the direction of incoming sound [9].

Humans use a combination of Inter-Aural Time Difference and Inter-Aural Level Difference to localize sound. We also have pinnae to give us more information about the direction of the acoustic source, and also use binaural cues to help us locate sound. The Time Difference method finds the time difference between the signal arriving at each microphone, and uses the planar wavefront assumption and some trigonometry to calculate the angle of the acoustic source.

Cross-correlation is one simple way to calculate the delay between the two signals. Once the delay is found, the path difference is calculated, and some simple trigonometry done to find the angle that the source makes with the normal to the line joining the microphones, as is seen in the diagram below. 3.1

Thus, the angle can be found as:

$$\cos\theta = \frac{\tau.c}{d} \tag{3.8}$$

3.4.1 Frequency limitations of two microphone array

If the acoustic source is a single beep, finding the time difference at different microphones is a simple task. However, if the source is a constant signal, such as a single cosine of some frequency, then we need a phase difference of at most π between the two signals, in order to say without ambiguity whether one signal is leading the other. To satisfy this condition, the distance d between the microphones should be less than half of the wavelength of the incoming sound, so that (eqn. 4.13, [21])

$$d \leq \lambda/2 \tag{3.9}$$

With a spacing of about 15 cm between our ears, humans can use Inter-Aural Phase Difference to localize sound up to a frequency of about 1kHz. For frequencies of incoming sound greater than that, humans use Inter-Aural Level Differences to localize sound.

3.4.2 Finding delay between two signals

Frequency-Dependent phase difference One way to calculate the delay between two signals when the signals are very limited in frequency, or consist of one or two distinct frequencies, is to calculate the phase difference, as :

$$\begin{aligned} \delta &= \frac{\angle FFT(leftch[n])}{\angle FFT(rightch[n])} \\ t &= \frac{\delta}{2\pi f} \end{aligned} \tag{3.10}$$

where f : frequency in Hz
δ: phase difference in radians
t : time difference in seconds
n : the frequency bin which corresponds to frequency f, in the Fourier Transform.

This method is better suited for signals which occupy a very narrow band in the spectrum, such as individual tones in the spectrum, than signals which are wideband or noise-like (like knocks).

Cross-Correlation Cross-correlation of two signals measures the similarity in two signals. In this case, if the signals arriving at the right and left channels are exactly the same signal, but delayed by a few samples. The cross-correlation function performs the correlation of one signal with the other, and the peak of the cross-correlation function indicates the point at which the signals are similar.

Cross-correlations of two signals x and y of lengths N and M respectively:

$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x[n]y[n-l] \tag{3.11}$$

(Section 2.6, [22])

The output signal r_xy is of length $N + M - 1$. The delay in seconds can be found as:

$$t = \frac{ceil(\frac{n+m-1}{2}) - \arg\max(r_{xy})}{fs} \tag{3.12}$$

where t is the computed delay in seconds,
$ceil(x)$ is a function which returns the smallest integer greater than or equal to x
$\arg\max()$ are the points of the domain (arguments) of some function at which the function values are maximized.

3.5 Speech-Music separation

Once the Roomba was made to move towards a source of sound, the next task was to get it to move selectively towards a particular type of sound, namely voice, and not towards other sources such as music or noise. The aim of this part of the project was, if music is playing from one direction in space, and a voice is talking or singing at a comparable volume from another direction, the Roomba should move towards the voice and not toward the music. The primary task would then become to somehow distinguish between voice, music and noise.

In the initial stages of this project, a feature called Spectral Flatness was used to distinguish between tonal and atonal sounds. However, speech and music are both tonal sounds, and so a different approach would have to be used to distinguish between frames of speech and music. Another requirement is for the Raspberry Pi to be able not just distinguish between speech and music, but also to be able to filter the voice from a frame with music and voice playing simultaneously. Thus, the final aim would be to have a running filter box that takes a frame with noise, music and voice playing simultaneously from different directions, and outputs only the voice part of this sound, while suppressing every other source of sound.

The problem of efficient Speech-Music discrimination is still an open one. Temporal features of the audio signal would be more effective in discimination. [23].

3.5.1 Speech-Music Discrimination using Support Vector Machine and Feature extraction

Previous approaches to this problem used features extracted frame-wise, namely,

1. Spectral Flux: Music has a higher rate of change, and goes through more drastic frame-to-frame changes than speech does; this value is higher for music than for speech [24]. It is computed as:

$$SF_r = \sum_{k=1}^{N/2} (|X_r[k]| - |X_{r-1}[k]|)^2 \quad (3.13)$$

[25] where X is the spectrum, r is the rth frame and SF_r is the spectral entropy for the rth frame.

2. Spectral Entropy: A measure of uncertainty or disorder in a given distribution.
The spectral entropy for a signal is calculated as:

$$\begin{aligned} P(w_i) &= \frac{1}{N}|X(w_i)|^2 \\ p_i &= \frac{P(w_i)}{\sum_i P(w_i)} \\ PSE &= -\sum_{i=1}^{n} p_i.ln(p_i) \end{aligned} \quad (3.14)$$

where $X(w_i)$ is the spectrum of the signal x, N is the number of frequency bins, p_i represents the normalized Power Spectral Density, which can then be interpreted as a Probability Distribution Function, and the Power Spectral Entropy can then be defined as given. [26].

3. MFCC coefficients: MFCC are a compact representation of the spectrum of a signal using a non-linear scale of pitch, as experienced by humans. Mel Frequencies Spectral or Cepstral Co-efficients (MFSC or MFCC) are very often used features for audio classification tasks, providing quite good results. [27]. A description for how to extract these features has been given in section 3.3.1.

4. Zero-Crossing Rate (ZCR): The number of time-domain zero-crossings within a speech frame. [28]. Speech signals show an increase in ZCR during the beginning and end of words. Music does not show such abrupt increases in ZCR, it being largely tonal. [28]

Once the features have been extracted for each frame, they are input into a feature vector for classification using a Support Vector Machine. This classifies the frames into silence, speech or music, depending on the features input to the classifier. The accuracy also depends on the training data given to the classifier, which would have to be manually annotated.

Different classifiers have also been used, namely Naive-Bayes classifiers, Neural Networks, or k Nearest Neighbour, for classification. However, previous results have shown Support Vector Machine (SVM) to be a suitable classifier for audio classification. [29].

However, while these methods may help to discriminate between speech and music when each frame is distinctly music or speech, they may not help when the frame contains both speech and music data simultaneously. In order to suppress the non-voice or speech signal, some kind of filter must be implemented.

3.5.2 Speech-Music Separation

The task described would have to distinguish and separate Speech from a mixed audio signal. The approaches to accomplish this task include :

Non-Negative Matrix Factorization Non-Negative Matrix Factorization (NMF) is essentially a method to separate a Matrix into the product of two matrices, thus accomplishing 'separation', with the property that none of the matrices have negative elements. [30]. Thus,

$$V = WH \tag{3.15}$$

In speech-music separation, the overall matrix V is described as the spectrogram of the audio signal. [31].

The matrix V is factorized into a weighted sum of basis vectors. Let W be a set of basis vectors, and H be the corresponding weights for each of these basis vectors. Let the spectrogram V have N frames and M frequency bins. Then, the matrices into which it is decomposed are a set of basis non-negative spectra (W) and their temporal activations (H) [30]. NMF gives a useful approximation in terms of a sum of weighted basis vectors which can be split up into contributions from different sources because it is unlikely that each source would occupy exactly the same frequency bin at exactly the same time instant.

Let V be of dimension MxK, and H be of dimension KxN. So there are M frequency bins, N time samples and K decomposed components. If w_i are the columns of W, and h_i are the rows of H, we can write:

$$V \approx \sum_{i=1}^{K} w_i h_i^T \tag{3.16}$$

If we know of only two sources in the recording, say piano and singing, their contributions can be split up by choosing some subsets of columns of W, and corresponding subsets of rows of H.

Then we can get

$$\begin{aligned} V_{piano} &\approx \sum_{i \in S} w_i h_i^T \\ V_{singer} &\approx \sum_{i \in (1,\ldots,K) \setminus S} w_i h_i^T \end{aligned} \tag{3.17}$$

In reality, we will likely end up with a decomposition which never achieves this separation exactly. That is, there will be w_is that have contributions from both the singer and the piano, making it difficult to separate the two. [32]

Filtering The simplest method that could be used to separate voice from mixed speech-music signals is to apply a low-pass filter, with a cutoff near the fundamental frequency of the human voice (usually below 300 Hz for a human female voice). The method would work if the same filter is applied to the left and right channels of the microphone. Thus, simply, frequency content above the cutoff is suppressed. If voice and music are playing together, the dominant signal in the low frequency region would be of voice, although there will be some spectral content from the music signal. Thus, the Robot would take the dominant signal as the voice signal, compute the angle and move towards the voice. When music is playing without any voice, there will still be some signal content in the low frequency region. Thus, the Roomba robot will pick up this signal in the low frequency region, compute the angle of arrival and move towards it. The computation of the angle should not be affected by the application of the low-pass filters, if exactly the same filter is applied to both channels (this was also verified with a simulation in Python). If reasonably accurate, this could prove to be a very low-complexity solution for speech-music separation which requires relatively low accuracy.

An Ideal Lowpass Filter

A low-pass filter passes the part of the signal whose frequency is below the cutoff (called the *passband* with the same amplitude, and completely suppresses any part of the signal whose frequency is above the cutoff (called the *stopband*). In an ideal low-pass filter, we want the magnitude of output of the filter in the passband to be exactly the same as that of the input, and the output of the filter in the stopband to be zero, with an instantaneous transition. That is, [33]

$$\begin{cases} |T(w)| = 1, w < w_c \\ |T(w)| = 0, w > w_c \end{cases} \tag{3.18}$$

Such an ideal filter is impossible to realize in reality, because such a perfect band-limited filter would have an infinitely long impulse response in time [33]. This means that the impulse response is *non-causal*, meaning that the output would depend on future inputs in time, which is clearly not realizable. Thus, we would have to accept some approximation in the filter that can be realizable in real time.

What happens if we take the Discrete Time Fourier Transform (DTFT) of a signal and manually suppress a set of frequencies and leave the others as they are? Does this not realize an 'ideal filter'? Multiplying the signal in the frequency domain by a rectangular window is equivalent to convolving the signal in the time domain with the transform of the rectangular window (which is the sinc function in the time domain, $sin(wt)/wt$). This would output ripples or 'ringing' in the time domain, which is an unwanted artifact. The sinc function also extends infinitely in time, which would mean that it takes inputs from the future.

Filter Parameters (Chapter 7, [15])

1. **Filter Order** is the number of previous samples required to calculate the current output. A lower order filter will require fewer computations.

2. **Passband** defines the cutoffs for the range of frequencies that should be retained in the signal.

3. **Stopband** defines the cutoffs for the range of frequencies that should be blocked in the signal.

4. **Transition band** is the band which is between the passband and stopband, where the filter changes its gain

5. **Minimum stopband attenuation** defines the minimum attenuation that the filter should have in the stopband.

6. **Maximum passband attenuation** the filter should not attenuate the signal by more than the maximum passband attenuation.

7. **Transition width** defines the slope of transition from the passband to the stopband or vice-versa. A smaller transition band is often preferred and is closer to the ideal filter response, but usually requires a higher order.

8. **Passband ripple** is the maximum variation in the gain of the filter in the passband.

Non-Ideal Filters

The filters that are used in practice are mathematical functions that try to come as close to the ideal filter as possible. Two popular filters used in signal processing are the butterworth filter and the elliptic filter.
Butterworth Filter The butterworth filter is popular because it has a 'maximally flat' response in the passband and the stopband, meaning that there isn't much distortion by the filter to the amplitude of the signal in the passband. [14]
Elliptic Filter The elliptic filter satisfies the specifications of a filter at the lowest order amongst the different filter types (Section 7.6, [15]). It has a higher ripple than the butterworth filter, but provides the minimum required attenuation in the stopband and maximum admissible attenuation in the passband at a lower order than the butterworth filter, which makes it more suitable for real-time applications.

Chapter 4

Methodology

The following sections outline the Methodology used to implement the desired system, including setting up the Raspberry Pi, interfacing with the Roomba, operating the Roomba, and the methods used in making the robot follow the direction of sound.

4.1 Setting up the Raspberry Pi

The Raspberry Pi Model 2 comes with an Edimax Wifi Adapter, 5V wall power adapter, and an 8GB NOOBS memory card with Raspbian Jessie as the operating system installed on it. Upon opening the Pi, one will notice that there is neither a display, nor a keyboard. Hence, there is no way to communicate with the Raspberry Pi the first time it starts up.

4.1.1 Enable SSH

Required items : USB keyboard (say from a PC), Pi, Wall charger with appropriate adapter
Steps to start up the Pi from scratch :

1. Connect the 5V adapter to a plug with a suitable adapter to a wall socket. Confirm that the power LED (green) switches on, and that the activity LED (yellow) does not blink. Disconnect power.
2. Insert the NOOBS memory card with Jessie OS installed on it.
3. Connect a portable keyboard (perhaps from a nearby PC) before connecting the Pi to power. If you have a display, connect the display as well, but it is not needed if you press the down arrow keys correctly as described.

4. Connect the Pi to a wall socket with the 5V adapter. Wait for one minute.
5. Type 'pi', ENTER, and then 'raspberry'
6. Type *sudo raspi-config*
7. Press ⇓ (down arrow key) 8 times and then press ENTER (To reach Advanced Settings)
8. Press ⇓ (down arrow key) 3 times and then press ENTER (To reach SSH settings)
9. Press ENTER thrice and then press ESC
10. Write *sudo shutdown -h now* and wait till the yellow light stops blinking
11. If all is well, SSH should have been enabled, and you can now access the Pi with an ethernet cable.

4.1.2 Connecting to the Pi via SSH, using Ethernet cable

You can use an ethernet cable to share internet with the Raspberry Pi and assign an IP to the Pi in order to communicate with the Pi.

1. Connect the laptop to a WiFi connection and in settings (for Ubuntu, double click on the network connections icon → Edit Connections → IPv4 Settings → Choose the method as Shared to other Computers).
2. Connect the Ethernet cable to the laptop and to the Raspberry Pi, and connect the Raspberry Pi to the power supply.
3. type *ifconfig* into a terminal on the laptop to find out the Ethernet IP of the laptop. (Mine was 141.24.14.xx).
4. Install the Nmap module on the laptop, by typing *sudo apt-get install nmap*.
5. In the terminal on the laptop, type *sudo -sS nmap 141.24.14.00/24*.
6. The output should show the IP address of the Raspberry Pi. Type *ssh pi@IPaddress* . If all went well, it should ask you for a password, which is 'raspberry'. Enter the password and you should be connected!
7. Now that you are accessing the Pi's terminal, you can try downloading a song using the *wget* (since Internet is being shared from your laptop) and playing it with the Pi's inbuilt audio output. If this works, you're ready to set up WiFi on the Pi!

4.1.3 Setting up WiFi on the Pi

Now that the Pi is accessible via the Ethernet cable, WiFi can be set up on it. The difficulty was in getting the setup to connect to the Eduroam network. [34]

1. Connect the Edimax Wifi adapter to a USB port on the Pi while it is switched off.
2. Type *sudo service networking stop* to stop networking.
3. On the Pi, edit the */etc/network/interfaces* file with the following content :

```
# interfaces(5) file used by ifup(8) and ifdown(8)

# Please note that this file is written to be used with dhcpcd
# For static IP, consult /etc/dhcpcd.conf and 'man dhcpcd.conf'

# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

iface eth0 inet manual

allow-hotplug wlan0
iface wlan0 inet manual
    wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf

allow-hotplug wlan1
iface wlan1 inet manual
```

4. On the Pi, edit the */etc/wpa_supplicant/wpa_supplicant.conf* file (keeping in mind to avoid any unnecessary spaces, for example between ssid and = and "eduroam"):

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
        ssid="eduroam"
        key_mgmt=WPA-EAP
        auth_alg=OPEN
        eap=TTLS
        identity="yourID@tu-ilmenau.de" # not the email address but the ID
        password="yourPassword"
        phase2="auth=MSCHAPV2"
        pairwise=CCMP
        proto=WPA RSN
}
```

5. Run wpa supplicant by typing :
   sudo wpa_supplicant -i wlan0 -c /path/to/wpa_supplicant.conf -B

6. reboot the Raspberry Pi by typing *sudo reboot*.

7. When the Raspberry Pi reboots, the Edimax WiFi adapter should now start blinking. To test whether it has successfully connected to the WiFi network, one can type *ifconfig* in the Raspberry Pi's terminal and check whether the wlan0 section has an IP address.

8. If *ifconfig wlan0* gives an IP address, note it down and disconnect the Ethernet cable, and then type *ssh pi@WiFiIPaddress*. Enter the password 'raspberry' when prompted, and SSH into the RPi wirelessly!

4.1.4 Sending a Mail with the wlan0 IP address from the RPi at start-up

Now all that's left is to find out the wireless IP address of the Raspberry Pi at start-up and we can do this by sending a Mail from the Pi with the necessary information at start-up.

1. create an email ID for the purpose of receiving IP addresses. (in my case rpiams13@gmail.com . The following instructions work for gmail accounts.)

2. install ssmtp on the RPi by typing *sudo apt-get install ssmtp* in the Pi's terminal.

3. install heirloom-mailx on the RPi: (this took up less space than another similar program, mailutils) by typing *sudo apt-get heirloom-mailx* in the Pi's terminal.

4. edit the /etc/ssmtp/ssmtp.conf file, by adding the lines below:

```
AuthUser=rpiams13@gmail.com # mail ID created for this purpose
AuthPass=raspberryams13     # password for the gmail account created
FromLineOverride=YES
mailhub=smtp.gmail.com:587
UseSTARTTLS=YES
```

5. Write a shell script to send the mail with the ouput of the ifconfig command to the mail ID : rpiams13@gmail.com (rpiams13@gmail.com sends the mail to itself), save it on the RPi and make it executable.

```
sleep 15
echo "\$(/sbin/ifconfig wlan0)" | mail -s "mail from pi(subject of mail)" rpiams13@gmail
.com
```

6. Add a cron job to execute at startup:
   type *sudo crontab -e* # edits the crontab file.
   Add the following line at the end of the file:
   @reboot /home/pi/send_mail.sh

7. sudo reboot and check whether the mail has been received at the email ID.

4.1.5 Syncing the Raspberry Pi with a home PC

It would be advisable to have a repository of programs on a PC and automatically update the programs on the Pi after editing them on the PC. One can do this with the help of the linux command rsync, which allows for remote transfer of files. This command can be saved as a shell script and executed on the Raspberry Pi.

```
rsync -avuz --delete-after NameOfHostComputer@AddressOfHostComputer:
pathToFolderWithCodesHostComputer/* pathToTargetFolderOnPi
```

4.2 Interfacing the Roomba with the Raspberry Pi

Now that the Raspberry Pi is fully set up to be controlled wirelessly, we can try to control the Roomba using it. The Serial Interface for the Roomba has been set up with the help of the blog entry at the DKØTU blog by G.Schuller, Simple 5V Serial Interface for Raspberry Pi 2 and 3 (*http://www.dk0tu.de/blog/2016/06/25_Raspberry_Pi_Serial_Interface*). [35]

The list of components required for the same, with the component numbers from Conrad Electronics is :

1. Raspberry Pi 2 Set —Bestell-Nr.: 1317772 - 62
2. 1 TSR1-2450 Traco Power — Bestell-Nr.: 156673 - 62
3. 7-pin Mini-DIN connector — Bestell-Nr.: 731781 - 62
4. 1 circuit board Bestell-Nr.: 531109 - 62
5. 1 socket row: Bestell-Nr.: 1303422 - 62
6. cable: Bestell-Nr.: 1398078 - 62
7. 10x 1kOhm resistor — Bestell-Nr.: 1089147 - 62

8. 10x 2kOhm resistor — Bestell-Nr.: 405299 - 62
9. 2x Green LEDs — Bestell-Nr.: 180182 - 62
10. 1 Red LED — Bestell-Nr.: 173533 - 62

4.2.1 Moving the Roomba

The Roomba is controlled via the Serial Interface made. This manual
(*http://irobot.lv/uploaded_files/File/iRobot_Roomba_500_Open_Interface_Spec.pdf*)
has a list of commands and examples to move the Roomba in a straight line, rotate it in place or make it turn at a certain radius.

4.2.2 Setup

The Raspberry Pi is mounted on the Roomba iRobot. It receives power from the Roomba via the serial interface. The Roomba provides 12V of power, which is converted using a voltage converter to 5V and is used to power the Raspberry Pi. The serial interface also has Rx and Tx pins which the Raspberry Pi uses to send command bytes. The microphones are connected via a USB sound card to the Raspberry Pi, and kept attached to a ruler at a distance of about 14 cm. They are kept in the position of 'ears' on the Roomba, so that the Roomba moves forward if the sound is between the two microphones.The microphones are placed approximately across the diameter of the Roomba. The robot performs better if kept in a room with absorptive material, rather than a room with a lot of reflective surfaces and furniture. (fig. 4.1.)

4.3 Audio Source Localization

Two microphones were used to find the direction of arrival of the acoustic source. Cross-correlation was used to find the difference in samples between the left and right channels, and using the planar wavefront assumption, the angle can be calculated. [21]

To deal with the problem of localization in $360^0 s$ with just two microphones, the Raspberry Pi listens for sound and calculates an angle (figure for reference: 4.2), and then the Roomba rotates by a small angle counter-clockwise, and listens again. This was described in section 3.4.

The Roomba now rotates and if the angle decreases (becomes more negative or less positive), the source was in the front of the Roomba. (figure for reference: fig. 4.3)
If the angle increases (becomes less negative or more positive), the source was behind the Roomba. The Roomba can now rotate accordingly to face the source of sound.

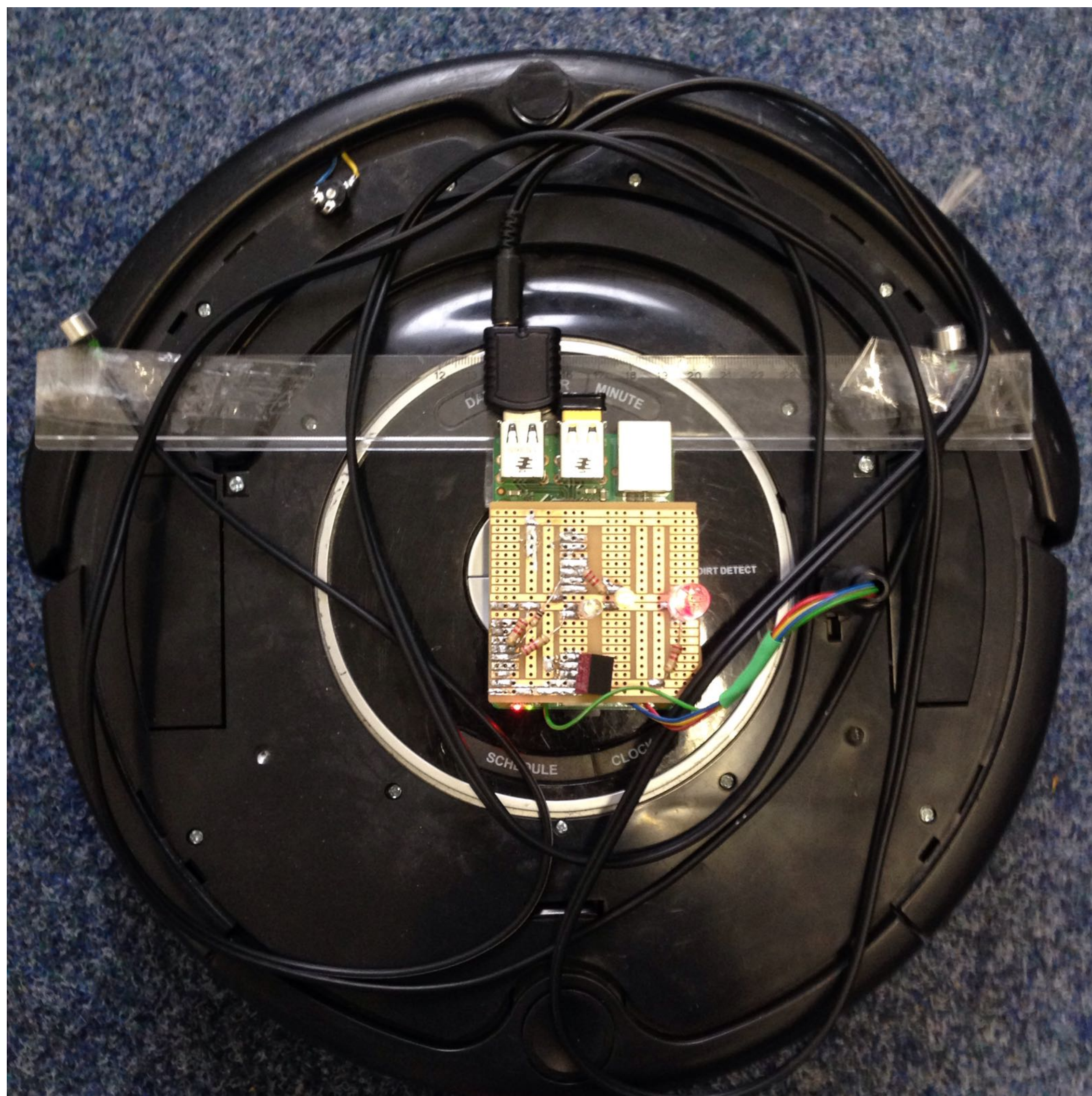


FIGURE 4.1: The Setup

4.4 Speech-Music Separation

The task of distinguishing between music and voice was attempted in different ways during the project. One way was by using features extracted from the audio and using them to train a Support Vector Machine. The classifier used was the OneVsRestClassifier from python's inbuilt module for classification, scikit-learn. The features used in classification were spectral flux, entropy, zero-crossing rate, and MFCC coefficients. The classifier was run on saved wave files, and on a PC with a higher processing power than the Raspberry Pi. It is, however, apparent that these classifiers, while being able to distinguish between frames of speech, music and silence, would not be able to separate them. The theory for this section has been described in the previous section, 3.5.

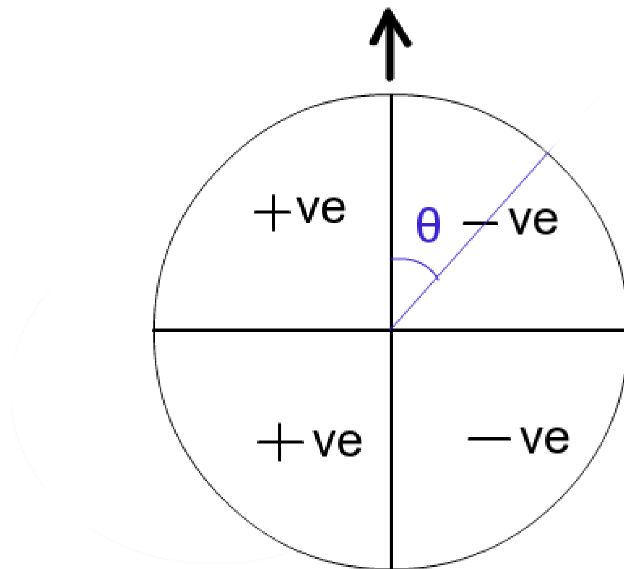


FIGURE 4.2: Top View of Roomba - before rotation

4.4.1 Speech-Music Discrimination

Speech-Music discrimination was attempted by using a OneVsRestClassifier of the Support Vector Machine class, from Python's inbuilt scikit-learn machine learning module.

1. The features were first extracted for an entire sample of acoustic guitar music, frame-wise, and then for a sample of speech, frame-wise.
2. The feature vectors were concatenated such that output was a composite feature vector for a concatenated signal of acoustic guitar music and then speech.
3. A 'frame marker' vector was manually defined, to annotate each frame as music or speech, which is known beforehand since the first few frames of the composite signal are of the acoustic guitar, and the latter half of the signal is known to be speech.
4. The Feature Vector was fed into the OneVsRestClassifier, along with the label for each frame as 'speech' or 'music' or 'silence' (manually annotated as such). This comprises 'training the classifier'

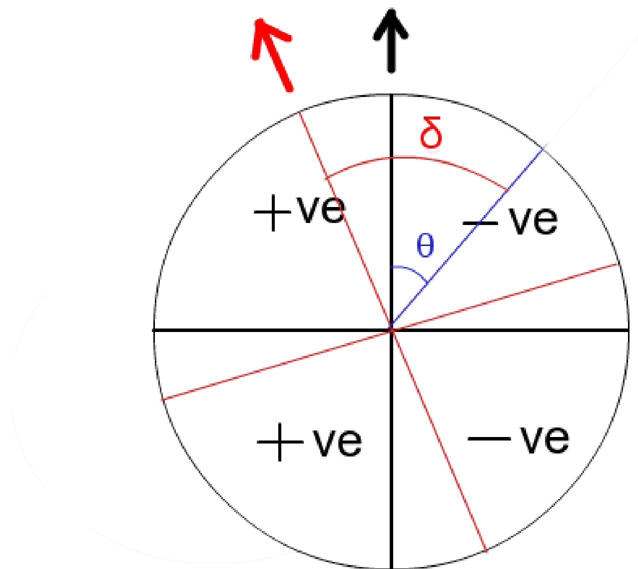


FIGURE 4.3: Top View of Roomba - after rotation

5. Two samples of audio - one of another sample of acoustic guitar music, and another sample of speech, were processed, and similarly, a feature vector was constructed after concatenating the features of the individual audio samples.

6. A prediction was made using the trained classifier and the vector plotted against the original time-domain signal. A prediction accuracy estimate was obtained.

The accuracy obtained using the features mentioned was only about 60% which is only marginally better than a random guess. A graphical representation of this result is shown in fig. 4.4. The classifier could be made more accurate by choosing different sets of features.

However, a major problem encountered with this method of discrimination, was that the method is too complex to run smoothly on the Raspberry Pi. Frame-wise feature extraction took too long, and it took upward of a minute per frame to calculate the features on the Raspberry Pi, as well as applying the classifier on the data.

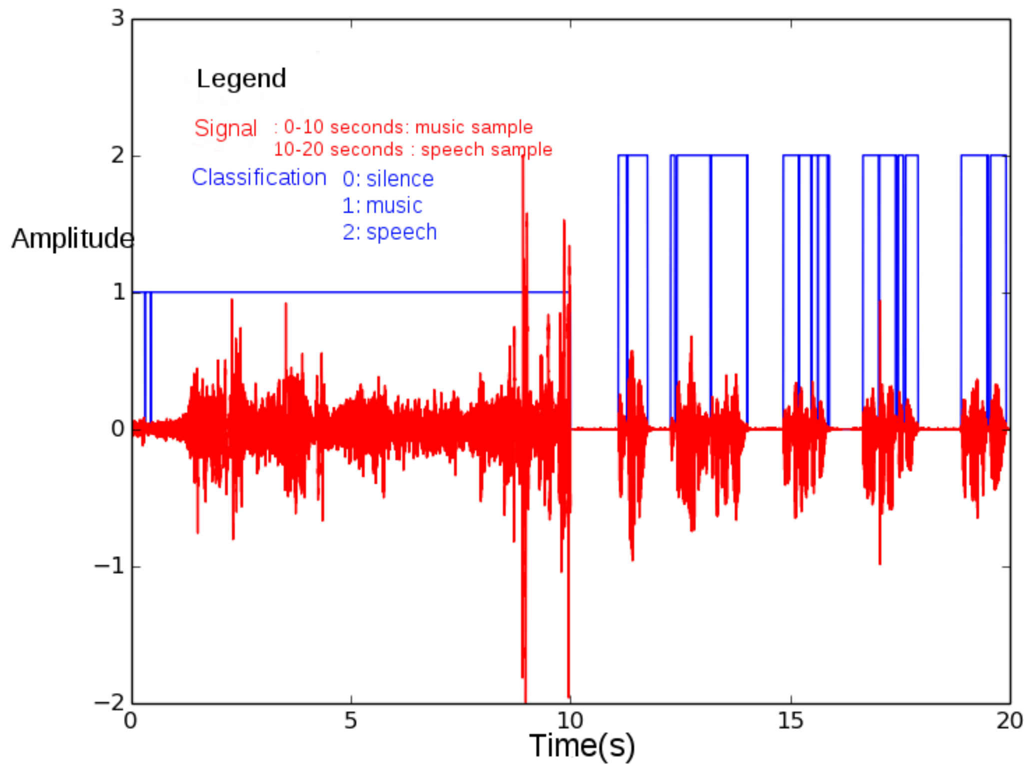


FIGURE 4.4: Signal classification by feature extraction with the given features, using the multi-class classifier from Python, on a saved wave file, on a computer with higher processing power.

Thus, a different method had to be used to distinguish between speech and music. Motivation was taken from the fact that speech contains most of its power below 4kHz, so that a low-pass filter could theoretically separate voice from music, using the fact that most musical instruments have a broader bandwidth than voice.

4.4.2 Speech-Music Separation

The theory for the working of filters was detailed in the previous section , 3.5.2. Speech-Music separation was done using this method with an elliptic filter with a cutoff frequency of 400 Hz, a minimum stopband attenuation of 60 dB and a maximum passband attenuation of 5, which led to a filter order of 6.This identical filter was applied to the left and right channels of the stereo microphone. The elliptic filter was used despite the distortion in the passband, due to its steep attenuation even at a lower order, as opposed to a Butterworth filter which requires a higher order to meet the minimum stopband and maximum passband attenuation requirements. The lower order ensures more attenuation at less computational power, which makes it suitable for

this real-time application of continuous frames of data.

It should be ensured that the PyAudio module is started just before data acquisition is to take place, and then stopped immediately after a block of data is acquired, to avoid delays which would affect the delay calculated by the RPi, and which would then affect the angle calculated by the RPi.

Since the Roomba was initially moving towards the dominant source of sound, the low pass ensures that low frequency sounds are dominant over higher frequency sounds. Music has a wider bandwidth than music, which means that some part of the music will still be retained by the low-pass filter. In the specific case of music of higher frequency (say, violin music at a higher octave) playing along with a voice singing to the music, the higher frequencies of the music are suppressed, making the voice the 'dominant signal' towards which the Roomba eventually moves. However, when the music is playing alone, the higher frequencies of the music are suppressed and the lower frequencies retained, and since there is no dominant low frequency voice, the lower frequencies of the music become the 'dominant signal' towards which the Roomba eventually moves.

Chapter 5

Results

Two microphones are sufficient to localize sound in 360° in a place if accompanied by a slight rotation. The Raspberry Pi was able to communicate with the Roomba iRobot and can be controlled entirely wirelessly. The system moves towards voice iteratively when high frequency instrumental music is playing in the background. The feature extraction and Support Vector Machine approach, or source separation, could be implemented on a system with higher processing power with reasonable accuracy. On the available platform, limited processing power prevents the use of complex algorithms like feature extraction and training with a support vector machine, or separation with non-negative matrix factorization.

Two videos have been uploaded onto the TU Ilmenau website (*https://www.tu-ilmenau.de/index.php?id=44974*) and can be viewed by interested readers:

- Roomba iRobot responding to foot-tapping on the ground
- Roomba moving towards voice despite music playing in the background

Chapter 6

Conclusion

This thesis aimed to investigate the applications of acoustic source localization and source separation in robot control, and implement a system for source localization on the Raspberry Pi, a processor with limited computational power. The aim was to take an approach that would have the least complexity in terms of number of elements and complexity of algorithms. In this aspect, the binaural microphone system implemented on the Roomba iRobot with the Raspberry Pi satisfies the goal of the thesis. The biologically-inspired method of using two microphones accompanied by a turning movement enabled us to localize the sound source in 360° without having to use more than two microphones, which avoided the trouble of synchronizing multiple microphones. The solution of using a low-pass filter on each frame of data instead of a complex source separation algorithm enabled the system to run smoothly and with an acceptable latency of a few seconds. The resultant system iteratively changes its direction and moves toward the source, and becomes more accurate as it gets closer to the source. The system works with less accuracy in a noisy,reverberative environment, but the robot still reaches the source, albeit after a larger number of iterations.

Future work in this direction could look closer into developing more sophisticated algorithms with low computational complexity for source-separation. Noise suppression algorithms and reverberation filters could be included to improve the accuracy of the system.

Chapter 7

References

[1] Durkovic, Marko. Localization, Tracking, and Separation of Sound Sources for Cognitive Robots. Diss. Universitätsbibliothek der TU München, 2012.

[2] Yoshioka, Takuya, et al. "Making machines understand us in reverberant rooms: robustness against reverberation for automatic speech recognition." *IEEE Signal Processing Magazine* 29.6 (2012): 114-126.

[3] Bentler, Ruth, and Li-Kuei Chiou. "Digital noise reduction: An overview." *Trends in Amplification* 10.2 (2006): 67-82.

[4] Hofman, P., and A. Van Opstal. "Binaural weighting of pinna cues in human sound localization." *Experimental brain research.* 148.4 (2003): 458-470.

[5] Athanasopoulos, Georgios, Henk Brouckxon, and Werner Verhelst. "Sound source localization for real-world humanoid robots." *Proc. SIP.* Vol. 12. 2012.

[6] Valin, J-M., et al. "Robust sound source localization using a microphone array on a mobile robot." Intelligent Robots and Systems, 2003.(IROS 2003). *Proceedings. 2003 IEEE/RSJ International Conference on.* Vol. 2. IEEE, 2003.

[7] Gu, Hung-Yan, and Shan-Siang Yang. "A sound-source localization system using three-microphone array and crosspower spectrum phase." *2012 International Conference on Machine Learning and Cybernetics.* Vol. 5. IEEE, 2012.

[8] Murray, John C., Harry Erwin, and Stefan Wermter. "Robotics sound-source localization and tracking using interaural time difference and cross-correlation." *AI Workshop on NeuroBotics.* 2004.

[9] Saxena, Ashutosh, and Andrew Y. Ng. "Learning sound location from a single microphone." *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on.* IEEE, 2009.

[10] Knapp, Charles, and Glifford Carter. "The generalized correlation method for estimation of time delay." *IEEE Transactions on Acoustics, Speech, and Signal Processing* 24.4 (1976): 320-327.

[11] DiBiase, Joseph Hector. A high-accuracy, low-latency technique for talker localization in reverberant environments using microphone arrays. Diss. Brown University, 2000.

[12] Smith, Julius O., III. "The Short-Time Fourier Transform — Spectral Audio Signal Processing." *The Short-Time Fourier Transform* — Spectral Audio Signal Processing. N.p., n.d. Web. 01 Dec. 2016.

[13] Smith, Julius O., III. "Choice of Hop Size." *Choice of Hop Size.* N.p., n.d. Web. 01 Dec. 2016.

[14] Podder, Prajoy, et al. "Design and Implementation of Butterworth, Chebyshev-I and Elliptic Filter for Speech Signal Analysis." International Journal of Computer Applications 98.7 (2014).

[15] Oppenheim, Alan V and Ronald W Schafer. Digital Signal Processing. 1st ed. Englewood Cliffs, N.J.: Prentice-Hall, 1975. Print.

[16] Ma, Yanna and Akinori Nishihara. "Efficient Voice Activity Detection Algorithm Using Long-Term Spectral Flatness Measure". *EURASIP Journal on Audio, Speech, and Music Processing* 2013.1 (2013): 21. Web.

[17] "SVM - Understanding The Math : The Optimal Hyperplane". *SVM Tutorial.* N.p., 2016. Web. 1 Dec. 2016.

[18] Randall, Robert B. "A history of cepstrum analysis and its application to mechanical problems." *International Conference at Institute of Technology of Chartres, France.* 2013.

[19] "Practical Cryptography". *Practicalcryptography.com.* N.p., 2016. Web. 1 Dec. 2016.

[20] Guentchev, Kamen, and John Weng. "Learning-based three dimensional sound localization using a compact non-coplanar array of microphones." *Proc. AAAI Spring Symposium on Intelligent Environments.* 1998.

[21] Scola, Carlos Fernández. Direction of arrival estimation–A two microphones approach. Diss. Blekinge Institute of Technology, 2010.

[22] Proakis, John G and Dimitris G Manolakis. Digital Signal Processing. 1st ed. Upper Saddle River, N.J.: Prentice Hall, 1996. Print.

[23] Balabko, Pavel. Speech and music discrimination based on signal modulation spectrum. *Tech. Rep,* 1999.

[24] Scheirer, Eric, and Malcolm Slaney. "Construction and evaluation of a robust multifeature speech/music discriminator." *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on.* Vol. 2. IEEE, 1997.

[25] Subramanian, Hariharan. "Audio signal classification." EE Dept, IIT Bombay (2004): 1-5.

[26] Sleigh, James W., et al. "Entropies of the EEG: the effects of general anaesthesia." (2001).

[27] Muñoz-Exposito, José Enrique, et al. "Speech/Music discrimination using a single Warped LPC-based feature." *Proc. ISMIR.* Vol. 5. 2005.

[28] Saunders, John. "Real-time discrimination of broadcast speech/music." *icassp.* Vol. 96. 1996.

[29] Chen, Lei, Sule Gunduz, and M. Tamer Ozsu. "Mixed type audio classification with support vector machine." *2006 IEEE International Conference on Multimedia and Expo.* IEEE, 2006.

[30] Smaragdis, Paris, and Judith C. Brown. "Non-negative matrix factorization for polyphonic music transcription." *Applications of Signal Processing to Audio and Acoustics, 2003 IEEE Workshop on..* IEEE, 2003.

[31] Bryan, Nicholas, and Sun, Dennis. "Source Separation Tutorial Mini-Series II: Introduction to Non-Negative Matrix Factorization." DSP Seminar. Center for Computer Research in Music and Acoustics, Stanford University. 9 Apr. 2013. Web.

[32] Atul Ingle (2016, November 28). Non-negative matrix factorization for audio separation - why does it work? [Msg 25]. Message posted to http://dsp.stackexchange.com/questions/35878/non-negative-matrix-factorization-for-audio-separation-why-does-it-work

[33] "The Ideal Lowpass Filter — Spectral Audio Signal Processing." *The Ideal Lowpass Filter* — Spectral Audio Signal Processing. N.p., n.d. Web. 01 Dec. 2016.

[34] Price, Will. "Eduroam on the Raspberry Pi." *Eduroam on the Raspberry Pi.* N.p., n.d. Web. 01 Dec. 2016. *http://www.willprice.org/2014/03/17/eduroam-on-the-raspberry-pi.html*

[35] G. Schuller. "Raspberry Pi Serial Interface." *Raspberry Pi Serial Interface.* N.p., n.d. Web. 01 Dec. 2016. *https://www.dk0tu.de/blog/2016/06/25_Raspberry_Pi_Serial_Interface/*

Chapter 8

Appendix

This section contains some of the Python codes used in the project.

8.1 Voice Detection and Separation

8.1.1 Frame-Wise Fundamental Freqency

```
import pyaudio
import numpy as np
# enter the required frequency resolution that you would like
reqdRes = 30 #Hz
fs = 44100 # Hz
CHANNELS = 2
WIDTH = 2
# required resolution = fs/N, where N  is the size of the fft
# find the closest power of 2 greater than fs/reqdRes
fftLen = 2**(np.ceil(np.log2(fs/float(reqdRes))))
CHUNK = 4096 #number of samples processed at a time

p = pyaudio.PyAudio()
stream = p.open(format=p.get_format_from_width(WIDTH),
           channels=CHANNELS,
           rate=fs,
           input=True,
           output=True,
           frames_per_buffer=CHUNK,
      )
# The arrays to plot frequency and time
R = []
# power thresold for deciding noise or music
threshold = 5.0
# frequency array
freq = np.linspace(0,int(fs),fftLen/2)
background = 1000.0
while(1):
```

```
            data = stream.read(CHUNK)
            input_wave = np.fromstring(data, dtype=np.int16)
        # take FFT of newest frame
            R = np.abs(np.fft.fft(input_wave[0:fftLen]))[0:fftLen/2]/np.sqrt(fftLen)
        # find average power in the frame
            pwr = abs((1/float(fftLen/2.0))*np.sum(np.square(input_wave[0:fftLen/2.0])))
        # find the power of the frame with minimum power while recording (assumption - minimum power
would be that of the background noise)
        # and assign that to the variable 'background'
            if pwr<background:
                background = pwr
        # find the ratio of power of the current frame (pwr) to the minimum power recorded till now (
background)
        # in order to have a relative idea of how loud the input is. rel_pwr will always be >1 as if
pwr is less than background (i.e. the power
        # of the current frame is less than the minimum), then the minimum power (background) will be
updated.
            rel_pwr = pwr/float(background)
        # find the peak frequency
            funda_freq = abs(freq[np.argmax(R)])
            funda_freq = funda_freq if funda_freq<fs/2 else fs-funda_freq
        # decide whether noise, music or voice based on relative power and peak frequency
        # threshold is manually decided by checking the rel_pwr variable as some noise is brought near
 the microphone
            if rel_pwr>threshold and funda_freq > 370:
                print 'music'
                print funda_freq
            elif rel_pwr<=threshold:
                print 'noise'
            elif rel_pwr>threshold and funda_freq <= 370:
                print 'voice'
                print funda_freq
            print ' '
        stream.stop_stream()
        stream.close()
        p.terminate()
```

8.1.2 Useful Functions used in Feature extraction

```
import numpy as np
from scipy.io import wavfile
import matplotlib.pyplot as plt
from scipy.fftpack import dct
from scipy.stats.mstats import gmean
from scipy.signal import butter, lfilter,buttord, hilbert, freqz, ellip,iirdesign

#======================================================================
#======================================================================
# Mel Frequency Scale
# inputs f: frequency vector
# output :  Mel-Frequency scaled frequency vector
```

```
def M(f):
   return 1125*np.log(1+(f/700.0))

#====================================================================
#====================================================================
# Inverse Mel Frequency Scale
# inputs m: Mel-Frequency scaled frequency vector
# output :  output frequency vector
def M_inv(m):
   return 700*(np.exp(m/1125.0)-1)

#============================================================================
#============================================================================
# Mathematical function
# inputs filename: filename of the .wav file
#     length : desired length of the file in seconds
# output :  pwr : frame-wise power in signal
def H(x):
   return (x+np.abs(x))/2.0

#============================================================================
#============================================================================
# Function to find the differential of an array
# inputs arr: the input array
# output :  differential of the array
def diff(arr):
   sum1 = 0
   for i in range(1,len(arr)):
      sum1+=0.5*np.abs(sign(arr[i])-sign(arr[i-1]))
   return sum1/float(len(arr))

#============================================================================
#============================================================================
# Signum Function
# inputs x: the input number
# output :  sgn(x)
def sign(x):
   if x<0:
      return -1
   elif x==0:
      return 0
   elif x>0:
      return 1

#====================================================================
#====================================================================
# Frame-Wise MFCCs of a signal
# inputs input_wave: input signal
#     factor: overlap factor
#     frame_length: length of a time frame
# output :  E_DCT : vector of frame-wise MFCC coefficients
def mfcc_coeff(input_wave,factor,frame_length):
   fs = 16000.0
   record_time = len(input_wave)/fs
   num_frames = int(np.floor(len(input_wave)/frame_length))
```

```
    S = np.zeros((num_frames/factor,frame_length/2))
    print 'num_frames/factor',num_frames/factor
    lower_mel_freq = M(300)
    upper_mel_freq = M(fs/2)
    freqs = np.fft.fftfreq(256, 1.0/fs)
    freq = np.linspace(0,256,256)
    num_banks = 26
    # compute mel filterbank, 26 banks
    m = np.linspace(lower_mel_freq,upper_mel_freq,num_banks)
    h = M_inv(m)

    # first fft bin - fs/N, second bin - 2*fs/N and so on
    # so, to check which bin each of the frequencies in h is closest to, use
    # Here N = so if first frequency bin number = fs/N,

    f = np.floor((frame_length+1)*h/fs)
    print 'ffffff',f
    # now, create the filterbanks
    H = np.zeros((num_banks,frame_length/2))
    for i in range(0,num_banks):
       for j in range(int(frame_length/2)):
          if j<f[i]:
             H[i,j] = 0
          elif  j>=f[i] and j<f[i+1]:
             H[i,j]=(j-f[i])/(f[i+1]-f[i])
          elif j>=f[i] and j<f[i+2]:
             H[i,j] = (f[i+2]-j)/float(f[i+2]-f[i+1])
          elif j>=f[i+2]:
             H[i,j]=0
    E = np.zeros((int(num_frames/factor),num_banks))
    for i in range(0,int(num_frames/factor)):
       frames=input_wave[i*factor*frame_length:frame_length*(1+factor*i)]
       if len(frames)<frame_length:
          frames = np.concatenate( [frames,np.zeros( frame_length-len(frames)) ] )
       FFT = np.fft.fft(frames)[0:len(frames)/2]
       S   = (1.0/frame_length)*np.square(np.abs(FFT))
       for j in range(num_banks):
          if np.sum(S*H[j,:]) == 0:
             E[i,j] = np.log(1)
          else:
             E[i,j] = np.log(np.sum(S*H[j,:]))
    # Take a 26 bin DCT of this energy spectrum
    E_DCT = dct(E)
    print 'len(E_DCT)',len(E_DCT),'len(E_DCT[0])',len(E_DCT[0])
#   return the first 13 coefficients
    return E_DCT[:,1:13]


#=====================================================================
#=====================================================================
# Function to implement the Short Term Fourier Transform of a signal
# inputs signal: the entire signal
#      fftLen: required length of the FFT vector
#      frameLen: length in samples of one frame in time
#      frameShift : desired amount of overlap
```

```
# output :  X : The spectrum of the spectrum of each frame
#     S : The squared magnitude of at each frame
def stft(signal,fftLen, frameLen, frameShift):
   numFrames = int(len(signal)/(frameLen*(frameShift/float(frameLen))))
   X = np.zeros((numFrames,fftLen))
   S = np.zeros((numFrames,fftLen))
   for m in range(numFrames):
      temp = signal[m*frameShift:m*frameShift+frameLen]
      if len(temp)!=frameLen:
         temp=np.concatenate(( np.zeros(frameLen-len(temp)),temp ))
      X[m,:]=np.abs(np.fft.fft(np.concatenate(( np.zeros((fftLen-frameLen)/2), temp , np.zeros
((fftLen-frameLen)/2) )) ))
      S[m,:]=np.square(np.abs(X[m,:])  )
   return X,S


#=============================================================================
#=============================================================================
# Function to implement the Inverse Short Term Fourier Transform of a signal
# inputs X: Short Term Fourier Transform of signal
#     overlap: overlap in samples
# output :  x : reconstructed signal
def istft(X, overlap=4):
    fftsize=(X.shape[1]-1)*2
    hop = fftsize / overlap
    w = scipy.hanning(fftsize+1)[:-1]
    x = scipy.zeros(X.shape[0]*hop)
    wsum = scipy.zeros(X.shape[0]*hop)
    for n,i in enumerate(range(0, len(x)-fftsize, hop)):
        x[i:i+fftsize] += scipy.real(np.fft.irfft(X[n])) * w   # overlap-add
        wsum[i:i+fftsize] += w ** 2.
    pos = wsum != 0
    x[pos] /= wsum[pos]
    return x

#=============================================================================
#=============================================================================
# Function to plot the Spectrogram of a signal
# inputs x: frequency vector
#     y: time vector
#     z: Spectrogram array
#     xlabel: xlabel of the plot
#     ylabel: ylabel of the plot
#     zlabel: zlabel of the plot
#     title : title of the plot
def spec_plot(x,y,z,xlabel,ylabel,title):
   fs = 44100.0
   fig, ax = plt.subplots()
#  ax.set_title(cmap_category + ' colormaps', fontsize=14)
   ax.set_yscale('linear')
#  ax.pcolormesh(x, y, z,cmap='RdBu')
   ax.pcolormesh(x, y, z)
#  plt.xlim([0, 100])
   plt.xlabel(xlabel)
   plt.ylabel(ylabel)
```

```
   plt.title(title)
   plt.show()


#==============================================================================
#==============================================================================
# Function to find the frame-wise zero crossing rate for a signal
# inputs signal: the input signal
#      factor: overlap factor, as a fraction of blockSize
#      blockSize : size in samples of each block
# output :  vsc : frame-wise zero crossing rate
def crossingRate(signal,factor,blockSize):
   num_blocks = np.int16( np.floor(len(signal)/blockSize))
   print 'number of iterations',num_blocks/factor
   vsc = np.zeros(int(num_blocks/factor))
   for i in range(0,int(num_blocks/factor)):
      start = int(i*factor*blockSize)
      stop  = min(len(signal)-1,int(blockSize*(1+factor*i)))
      vsc[i] = diff(signal[start:stop])
   return vsc



#==============================================================================
#==============================================================================
# Function to extract a single channel of a required length from a .wav file
# inputs reqdLen   : desired length of the file in seconds
#      samp_rate : Sampling rate of the audio file
#      snd      : stereo or mono numpy array
#
# output :  input_wave: ouput signal
def make_single_channel(reqdLen,samp_rate, snd):
    if np.size(snd[0])==2:
        if len(snd)>(reqdLen*samp_rate)-1:
            input_wave = snd[0:(reqdLen*samp_rate)-1,0]
        else:
            input_wave = snd[:,0]
    elif np.size(snd[0])==1:
        if len(snd)>(reqdLen*samp_rate)-1:
            input_wave = snd[0:(reqdLen*samp_rate)-1]
        else:
            input_wave = snd[:]
    return input_wave


#==============================================================================
#==============================================================================
# Function to read a .wav file into an array
# inputs filename: filename of the .wav file
#      length : desired length of the file in seconds
# output :  wave1 : a numpy array, single channel of required length
def get_wave(filename,length):
   pathname = '/home/sramnath/Downloads/'
   sampFreq, snd = wavfile.read(pathname+filename)
   snd = snd / (2.**15)
```

```
    wave1 = make_single_channel(length,44100,snd)
    return wave1

#============================================================================
#============================================================================
# Frame-wise Spectral Flux
# inputs filename: filename of the .wav file
#      length : desired length of the file in seconds
# output :  SF : frame-wise spectral flux
def spectral_flux(signal,factor,fftLen):

    num_blocks = np.int16( np.floor(len(signal)/fftLen))

    SF = np.zeros(int(num_blocks/factor))
    R  = np.zeros((int(num_blocks/factor),fftLen))
    print 'I am the dead',len(signal),num_blocks
    for i in range(0,int(num_blocks/factor)):
        start =int(i*factor*fftLen)
        stop  = min(len(signal)-1,int(fftLen*(1+factor*i)))
        sig = signal[start:stop]
        if stop-start < fftLen:
            sig = np.concatenate((sig,np.zeros(fftLen-len(sig))),axis = 0)
        R[i,:] = np.abs(np.fft.fft(sig))
        SF[i] = np.sum(   H( np.abs(R[i])  - np.abs(R[i-1]) ) )
    return SF

#============================================================================
#============================================================================
# Power in a signal
# inputs filename: filename of the .wav file
#      length : desired length of the file in seconds
# output :  pwr : frame-wise power in signal
def pwr(signal,factor, fftLen):
    num_blocks = np.int16( np.floor(len(signal)/fftLen))
    R = np.zeros(fftLen)
    pwr = np.zeros(int(num_blocks/factor))
    for i in range(0,int(num_blocks/factor)):
        start =int(i*factor*fftLen)
        stop  = min( len(signal)-1,int(fftLen*(1+factor*i)) )
        R = np.abs(np.fft.fft(signal[start:stop]))
        if len(R) < fftLen:
            np.concatenate((R,np.zeros(fftLen-len(R))),axis = 0)
        pwr[i] = np.mean(np.square(R))
    return pwr



#============================================================================
#============================================================================
# Frame-Wise Spectral Flatness Coefficient
# inputs filename: filename of the .wav file
#      length : desired length of the file in seconds
# output :  pwr : frame-wise power in signal
def spectral_flatness(signal,factor,fftLen):
    num_blocks = np.int16( np.floor(len(signal)/fftLen))
```

```
    LSFM = np.zeros(int(num_blocks/factor))
    for i in range(int(num_blocks/factor)):
        start = int(i*factor*fftLen)
        stop  = min(len(signal)-1,int(fftLen*(1+factor*i)))
        R = np.abs(np.fft.fft(signal[start:stop]))
        if len(R) < fftLen:
            np.concatenate((R,np.zeros(fftLen-len(R))),axis = 0)
        AM = np.mean(R)
        GM = gmean(R)
        LSFM[i] = np.log10(AM/float(GM))
    return LSFM



#==============================================================================
#==============================================================================
# Moving Average Filter
# inputs a: input array
#     n: order of filter
# output :  filtered signal
def mving_avg(a,n):
    ret = np.cumsum(a,dtype=float)
    ret[n:] = ret[n:]-ret[:-n]
    return ret[n-1:]/n

#==============================================================================
#==============================================================================
# Function to apply a filter of variable type on a signal
# inputs signal: input signal
#     cutoff_freq : cutoff frequencies as a number or array
#     filter_type : desired type of filter
#     fs      : sampling rate
# output :  filtered signal
def filterit(signal,cutoff_freq,filter_type,fs):
#   cutoff = cutoff_freq/(float(fs)/2)
#   (N, Wn) = buttord(wp=norm_pass, ws=norm_stop, gpass=2, gstop=30, analog=0)
#   N = 3
#   print cutoff
#   b, a = signal.butter(4, 100, 'low', analog=True)
#   (b, a) = butter(N, cutoff, btype='low', analog=0)

    # design filter
    norm_pass = cutoff_freq/(fs/2)
    norm_stop = 3.0*norm_pass
    N = 6
#   (N, Wn) = buttord(wp=norm_pass, ws=norm_stop, gpass=2, gstop=30, analog=False)
#   print N,Wn
    (b, a) = butter(N, cutoff_freq/(44100.0/2), btype=filter_type, analog=False, output='ba')
    # filtered output
    return lfilter(b, a, signal)


#==============================================================================
#==============================================================================
# Downsampler
```

```
# inputs signal: input signal
#     factor : downsampling factor
# output :  downsampled signal
def downsample(signal,factor):
   # signal assumed to be sampled at fs Hz originally
   # downsampling by a factor of 2 would bring the new sampling at fs/2
   return signal[::factor]

#==============================================================================
#==============================================================================
# Running Mean filter
# inputs X: input signal
#     N : number of samples per frame
# output :  filtered signal
def runningMeanFast(x, N):
   return np.convolve(x, np.ones((N,))/N)[(N-1):]

#==============================================================================
#==============================================================================
# Frame-Wise Power Spectral Entropy
# inputs signal: the input signal
#     factor: overlap factor, as a fraction of blockSize
#     fftLen : size in samples of each block
# output :  Frame-Wise entropy
def entropy1(signal,factor,fftLen):
   fs = 44100.0
   freq = np.linspace(0,fs,fftLen)
   num_blocks = int( np.floor(len(signal)/fftLen))
   entr = np.zeros(int(num_blocks/factor))
   for i in range(0,int(num_blocks/factor)):
      start = int(i*factor*fftLen)
      stop  = min(len(signal)-1,int(fftLen*(1+factor*i)))
      tot = np.sum(np.square(signal[start:stop]))
      quo = np.abs(signal[start:stop]) / tot
      if ~(quo.all()):
         quo = np.array([1 if quo[j] == 0 else quo[j] for j in range(len(quo))])
      entr[i] = np.sum(np.multiply(quo, np.log10(quo)))
   return entr
```

8.2 Acoustic Source Localization

8.2.1 Moving the Roomba

```
# Program to listen to audio and move the roomba towards the source of audio
import pyaudio
import numpy as np
import struct
import serial
import time
from moveRoomba import roomba_start, roomba_rotate, roomba_move, roomba_stop
```

```
port = serial.Serial("/dev/ttyAMA0",baudrate = 115200, timeout =3.0)
from scipy.signal import butter,lfilter, freqz, ellip

block = 4096 #number of samples processed at a time
fs = 44100 # Hz
CHANNELS = 2
WIDTH = 2
sound_speed = 340.29*100 # cm/s
L = block
N = 4000
M = N/2
freq = np.linspace(0,fs,block)
t = np.linspace(0,float(block)/fs, np.ceil(float(block)/M)+1)
t1 = np.linspace(0,float(block/2)/fs, np.ceil(float(block/2)/M)+1)
print 'len(t)',len(t),'len(freq)',len(freq)

# start up the roomba
roomba_start(port)

# open pyaudio object
p = pyaudio.PyAudio()

# power thresold
threshold = 1.7
background = 10000.0

# angle coordinates : (top view of the roomba)
#
#             ##|##
#  top left      #    |    #       top right
#       # +ve  |  -ve #
#      #       |       #
#      #---------------#
#      #       |       #
# bottom left    #  +ve |  -ve #   bottom right
#         #    |    #
#             ##|##
#
# The roomba listens for a sound, find the angle of the sound
# then rotates counter-clockwise again by a small angle, listens again and finds the angle
# if the angle reduced (less positive or more negative), the sound source was in front.
# if the angle increased (more positive or less negative), the sound source was at the back.
# if the object is at top left of the roomba, it rotates by a positive angle theta (counter-
clockwise)
# if the object is at top right of the roomba, it rotates by a negative angle theta (clockwise
)
# if the object is at bottom left of the roomba, it rotates by theta+90 (counter-clockwise) ,
theta being positive
# if the object is at bottom right of roomba, it rotates by theta-90 (clockwise), theta being
negative

def find_angle(leftch,rightch):
#  normalized left and right channels (interleaved in the signal)

   zcr= ((signal[:-1] * signal[1:]) < 0).sum()
```

```
#   leftch = filterit(leftch,320,'low',fs)
#   rightch = filterit(rightch,320,'low',fs)
    print zcr
#   find time difference between the two signals reaching the two microphones
#   using cross-correlation
    corr_arr = np.correlate(leftch,rightch,'full')
    max_index = (len(corr_arr)/2)-np.argmax(corr_arr)
    print 'max_index',max_index
    time_d = max_index/float(fs)
#   print 'time diff',time_d
    val = (time_d*sound_speed)/(22.0-8.0) # update distance between microphones
    print 'val',val

#   if it's valid then move
    if (val<1 and val>-1) and time_d !=0 and zcr>=2000:
       print 'case 1'
       angle = 90.0-(180.0/3.14)*np.arccos(val) # distance between microphones = 14 cm
       print angle,'degrees'
#      plt.plot(corr_arr)
#      plt.show()
    elif (val>1 or val<-1 or time_d==0 or zcr< 2000):
    # invalid value : dont move
       print 'something''s wrong'
       angle = 0.0
    print angle,'degrees'
    return angle

# Design the elliptic filter with a cutoff at 350 Hz, and a minimum stopband attenuation of 60
dB
b, a = ellip(4, 5, 60, [65.0/(fs/2),350.0/(fs/2)], 'band', analog=False)

while(1):
#   start stream
    stream = p.open(format=p.get_format_from_width(WIDTH),
            channels=CHANNELS,
            rate=fs,
            input=True,
            output=True,
            frames_per_buffer=block,
       )
#   read data
    data = stream.read(block)
    # once the data has been acquired, stop listening
    stream.stop_stream()
    stream.close()
    shorts = (struct.unpack( 'h' *2* block, data ))
    signal=np.array(list(shorts),dtype=float)
#   divide into left and right streams
    leftch = signal[0::2]
    rightch= signal[1::2]

#   now filter the left stream and the right stream with the same filter
    filtered_lnoise = lfilter(b,a,leftch)
    filtered_rnoise = lfilter(b,a,rightch)
```

```
	#  find power in left and right streams
	   lfnoisepwr = np.sum(np.square(filtered_lnoise))/float(len(filtered_lnoise)+len(
filtered_rnoise))
	   rfnoisepwr = np.sum(np.square(filtered_rnoise))/float(len(filtered_lnoise)+len(
filtered_rnoise))


	   pwr = lfnoisepwr+rfnoisepwr
	   if background > pwr:
	      background = pwr
	#  print 'background',background
	   pwr_back = pwr/float(background)
	   if pwr_back>threshold:
	      print 'listening...'
	      angle1 = find_angle(filtered_lnoise,filtered_rnoise)

	#     if the angle is sufficiently large, rotate counter-clockwise by 5 degrees and listen
again,
	      if abs(angle1)>1:
	   #     now rotate counter-clockwise by  5 degrees and listen again
	         print 'preliminary rotation...'
	         roomba_rotate(port, 10, 150)
	         time.sleep(0.2)
	         print 'listening again...'
	   #     once again open the audio stream and listen for another block of data
	         stream = p.open(format=p.get_format_from_width(WIDTH),
	            channels=CHANNELS,
	            rate=fs,
	            input=True,
	            output=True,
	            frames_per_buffer=block,
	            )
	         data = stream.read(block)
	         # stop listening immediately after data acquisition
	         stream.stop_stream()
	         stream.close()

	         shorts = (struct.unpack( 'h' *2* block, data ))
	         signal=np.array(list(shorts),dtype=float)
	         leftch = signal[0::2]
	         rightch= signal[1::2]
	   #     Apply the filter on right and left channel streams
	         filtered_lnoise = lfilter(b,a,leftch)
	         filtered_rnoise = lfilter(b,a,rightch)
	#        new angle found
	#        if the angle has 'reduced' (reduced on the absolute scale i.e. less positive for a
positive angle or more negative for negative angle)
	#        then the source was in front. If the angle increased, the source was at the back.
	         angle2 = find_angle(filtered_lnoise,filtered_rnoise)
	         sgn = 1 if angle1>0 else -1 # check if object was initially in left or right
hemisphere

	         if angle2<angle1:
	            print 'source is in front of me...'
	            roomba_rotate(port,int(angle2),200)
```

```
            time.sleep(0.2)
            roomba_move(port,150,300)
            time.sleep(0.2)
         else:
            print 'source is behind me...'
            roomba_rotate(port,int((sgn*90)+angle2),200)
            time.sleep(0.2)
            roomba_move(port,150,300)

         print 'done rotating...'
      else:
#     angle wasn't large enough
         print 'angle wasn''t large enough to rotate:',angle1
   else:
#  sound wasn't loud enough - do nothing
      stream.stop_stream()
      stream.close()
      print 'no sound detected'
   print ' '
```

8.2.2 Functions to Move the Roomba

```
import serial
import time


def roomba_start(serialObj):
   serialObj.write(b'\x80')
   time.sleep(1)
   serialObj.write(b'\x82')
   time.sleep(1)


# for moving anything
# [137][velocity high byte][velocity low byte][radius high byte][radius low byte]
# to move straight : radius = 32768 or 0x8000
# to rotate in place : radius = 0x0001

def roomba_rotate(serialObj,angle,velocity):
   # velocity in mm/s (v=wr, w = vr , r = 258/2 mm, so w = v*2/258) , angle specified in
degrees..
   # so angle_rad = angle_deg*pi/180
   w = velocity/129.0
   time_sleep = (abs(angle)*3.14)/float(w*180)
   print time_sleep
   if angle > 0:
      serialObj.write(b'\x89'+chr(velocity/256)+chr(velocity%256)+b'\x00\x01')
   # now stop the roomba
   else:
      serialObj.write(b'\x89'+chr(velocity/256)+chr(velocity%256)+b'\xff\xff')
   time.sleep(time_sleep)
```

```
        # now stop the roomba
        serialObj.write(b'\x89\x00\x00\x00\x00')

def roomba_move(serialObj,distance,velocity):
        # velocity in mm/s, distance in mm so time in s
        time_sleep = distance/float(abs(velocity))
        serialObj.write(b'\x89'+chr(velocity/256)+chr(velocity%256)+b'\x80\x00')
        time.sleep(time_sleep)
        # now stop the roomba
        serialObj.write(b'\x89\x00\x00\x00\x00')

def roomba_stop(serialObj):
        serialObj.write(b'\x89\x00\x00\x00\x00')
        serialObj.close()
```