University of Southern Queensland

Faculty of Health, Engineering & Sciences

Make Garfield (6 axis robot arm) Smart through the design and implementation of Voice Recognition and Control

A dissertation submitted by

Kyle Tonkin

in fulfilment of the requirements of

Course ENG4111 Research Project part 1 & ENG4112 Research Project part 2

Towards the degree of

Bachelor of Electrical Engineering

Submitted: October, 2013

ABSTRACT

The Voice is a powerful tool used in everyday life. It expresses our feelings, emotions and is unique to every individual. Developing a system to decipher its functionality and deal with the complexities involved in its digitalisation, becomes quite the task. Once harnessed, it has the potential to become one of the most popular methods of control throughout the world.

Garfield (a 6 axis robotic arm) is in need of an extra method of control that can essentially make it 'smarter'. In trend with current popularity, Voice Recognition has been selected as the method for control and the processes taken to implement such a system are explored in the following dissertation. Its outlines the design, development and implementation of a working voice recognition system used in conjunction with robotic arm software. In essence, this project explores the application of such a technology to the industrial robot industry and tests its performance as a whole.

A signal processing methodology was investigated, designed and developed in order to implement the voice as a control tool on the robotic system. This included the research and development of a voice detection algorithm based on the principles of voice recognition and required integration with robotic control software to execute appropriate movement.

Existing literature was explored in order to understand and apply the concepts of voice recognition and integrate it into a system responsive to user utterances. The basic outline of the processing techniques used followed those involved with Mel Frequency Cepstral Coefficients (Kumar & Rao 2011) and the recognition techniques involved with Euclidean distances (Muda, Begam & Elamvazuthi 2010). In applying these techniques within the Matlab platform, some Voice Processing tools (Brookes 1998) were sort out and used. A number of final design parameters were evaluated and tested in order provide recommendation to further system users and set objectives for future work.

The results of system performance testing discovered that jogging the system via the voice controlled method was viable with further improvement to the system response and quality of the control algorithm. Furthermore, possible solutions to the blemishes in the system are explored and avenues for further research and development stated.

University of Southern Queensland

Faculty of Health, Engineering & Sciences

ENG4111 Research Project Part 1 &

ENG4112 Research Project Part 2

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Health, Engineering and Sciences, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering and Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled “Research Project” is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Professor Frank Bullen

Dean

Faculty of Health, Engineering and Sciences

CERTIFICATION

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Student Name: Kyle James Tonkin

Student Number: 0061005209

_____

Signature

_____

Date

ACKNOWLEDGEMENTS

I would firstly like to acknowledge the support and advice given to me by my supervisor, Dr Tobias Low. Without his presence, I would not have been able to progress through the year and submit my final dissertation. His knowledge and skills have been flawless, helping me through all of the problems I have faced.

I would also like to thank my family and friends, who have been there for me throughout the year and supported everything I have done. Hopefully they will remain there for me as I progress into the future. Thank you.

KYLE

TONKIN

University of Southern Queensland

October 2013

Contents

List of Figures

List of Tables

Nomenclature

VR	Voice Recognition
ASR	Automatic Speech Recognition
PSD	Power Spectral Density
HMM	Hidden Markov Model
VQ	Vector Quantisation
ANN	Artificial Neural Networks
MFCC	Mel Frequency Cepstral Coefficients
DFT	Discrete Fourier Transform
FFT	Fast Fourier Transform
DCT	Discrete Cosine Transform
WER	Word Error Rate

Chapter 1

Robotic Voice Recognition and Control

1.1 Introduction

Ever since humans have walked the earth, they have sought faster and more efficient ways in which to perform everyday tasks. It is only now, where human technology has reached a level allowing the creation of machines that can out-think and out-perform the average human being. Robotics has been an ever expanding industry since the early 1900's and has been a major contributor to the success of our growth as a species. It's involvement in mass production, medicine, exploration and transport has made it an invaluable tool, in which we simply cannot move in to the future without. Robotics has evolved dramatically in the past century, not only has the world seen faster, more accurate and reliable systems, but ones that have grown to display human like characteristics and thinking. It is with this expansion in the intelligence of robots, that a need arises for greater control of these machines (Gates 2007).

Today there are a number of methods available that allow the standardised control of machines. Every machine is run by a distinct set of programs and procedures where code is executed to perform both simple and complex tasks. The computer interface and remotes or pendants are the main control methods available and are largely focussed on user input and interactivity. Newer machines are capable of determining inputs and conditions without user input, following the implementation of advanced sensors and machine learning techniques.

All in all, voice as one of these control methods, forms a low percentage of those readily available today and is still yet to fully utilise the effectiveness of 'the voice' as a control technique. Developing a perfect system on the basis of voice recognition, that has the ability

to follow commands without error and recognise utterances from all types of speakers, is one that is currently still in creation.

Voice recognition has become a quite popular technique in recent years as a simple, hands free control method, whether it be used for voice dialling in mobile phones or for movement response in children's toys. The development of voice recognition and control has been quite rapid as a result of this popularity and its application in industry is becoming more and more likely. As the algorithms increase in quality and are refined to lock on to specific speech patterns, greater accuracy is achieved and can be implemented without risk of failure (Rabiner 2004). A major consideration for this technology is its response to 'noise' and unexpected inputs. Not only can these cause the wrong output, but put the safety of the operator at risk. Depending on what type of activity is being conducted, any mishap or wrong movement could result in millions of dollars of production losses or even loss of life. Knowing these risks, it becomes quite important that voice recognition is simulated thoroughly before being applied to a system and tested in the real world.

Voice, even sound in general, is a quite complicated phenomenon. It is used every day without any effort or thought into how it is being produced. Every human being is unique, no matter how similar one person may be to another and with this diversity, comes a variety of speech patterns. These patterns vary in both frequency and magnitude and can be matched using a number of techniques. The most reliable technique involves finding the Cepstral properties of the recorded sound. These properties present subtle differences which can be used to distinguish between 2 phonemes or utterances, i.e. the words 'one' and 'two'. It is important to understand that single word voice recognition is much simpler than continuous word recognition, as it does not require the constant determination and processing of word boundaries. Essentially, once the important properties of a particular utterance are obtained, they can be compared with a previously built model in order to determine the spoken word. The most commonly used method of matching is based on the Hidden Markov Model and its use of probability (Iqbal, Mahboob & Khiyal 2011). Once the word is spoken, analysed and matched, an output is determined and presented to the user, whether it be in the form of authentication, movement, words or even sound. There are a number of other methods which have shown considerable success when it comes to voice recognition, these will be explored later in the report.

A number of software packages and programs have been previously developed that recognise, to an extent, the speaker's voice and translate it either to text or to another specific output. Dragon Naturally Speaking is one of these programs that convert everything you say into text. This particular program is up to 99 percent accurate in interpreting the correct words, considering the amount of money invested in such a program this is quite reasonable (Communications 2013). In a recent journal involving voice recognition for Automatic Teller machine security, an 86.67% acceptance rate was recorded, as well as a 13.33% rejection rate. The system used in this instance was built using Matlab. It implemented a Mel Frequency Cepstral Coefficient and Hidden Markov Model based algorithm which was quite successful in determining voice properties in the presence of noise (Iqbal, Mahboob & Khiyal 2011). This section of the project becomes quite an important first step. Without the correct determination of verbal commands there will be nothing to call a response from the robot.

Currently Garfield, the 6 axis robot arm, is controlled by an application called RobotStudio. This particular program uses various coding techniques to control a number of variables, which in turn allow the execution of movements. In order to improve the capabilities of Garfield, it seems appropriate that some other form of control be implemented. Considering the growing popularity of voice recognition and its ever expanding applications, integrating a system like this onto a 6 axis robot arm seems quite smart. Newer technologies are pointing towards a future in which we rely on robotics to sustain our world and learning about the methods involved in the control of these machines through such a project, seems quite suitable.

Figure 1 - ABB 6 axis Robot Arm

In dealing with a 6 axis robot arm, a detailed knowledge of the programming language used for its operation is required. An understanding of its capabilities and its limitations is also imperative. Ideally using voice recognition technology as a method of control would require a fast response to input from the user. Considering the processing power of the average

computer and the time needed to execute an algorithm, response to input for this particular project can be expected to be relatively slow.

Creating a reliable system with the resources available can be quite difficult and time consuming. Thus it is important to state that providing an input and obtaining a response from Garfield is the basic objective needed to be reached. From here, increasing the efficiency, response time and manoeuvrability of Garfield becomes the next focus in this project.

1.2 Project Aim

The primary aim of this project is to design, develop and implement a voice recognition and control system onto Garfield, the 6 axis robot arm. This prototypical software-based system will look at demonstrating the concepts of voice recognition and control and explore its limitations. The project will involve the creation of a voice recognition algorithm in Matlab and a program in RobotStudio which will integrate together over a TCP/IP socket connection. The system must provide an accurate means of control, allowing the user to display the capabilities of 6 axis robot arm control. With regards to speed, it is important that the arm responds quickly in real time to ensure the safety of the user. The system must first demonstrate a basic response to voice input and be simulated on RobotStudio software before being implemented onto the 6 axis robot arm itself.

This project aims to investigate methods of voice recognition, analyse the best method for this application and implement it using various sound processing techniques and algorithms. This prototypical software will be tested, analysed, evaluated and further developed to improve such performance criteria as speed, accuracy and manoeuvrability.

Upon completion of the project, it is intended that the prototypical software created in this process can be further developed and implemented for an alternative control method for Garfield, the 6 axis robot arm and be used as a possible learning tool for voice recognition systems within the University of Southern Queensland.

1.3 Project Objectives

The project aim was reviewed and split into a number of objectives for completion.

- Identification of current voice recognition techniques
- Identification and Investigation of best suited voice analysis method
- Development of algorithm in Matlab addressing this method
- Development of code using RobotStudio software to integrate with this method
- Combine and Implement these two developments in simulation package
- Evaluation of prototypical software
- Investigate any modifications to improve system performance
- Implement these modifications and Evaluate
- Implement the prototypical software onto Garfield
- Make recommendations for further development

1.4 Overview of Dissertation

The dissertation is organised as follows:

Chapter 2 Investigates the relevant research papers on the voice recognition subject area and explores the techniques used to apply this form of control. These techniques will involve signal processing, model building, program communication and simulation elements. There relevance to this particular application will be discussed and evaluated.

Chapter 3 States the methodologies to be undertaken in order for project completion. It outlines the strategies used in creating the system software and its implementation onto Garfield. This chapter also looks at the relevant risks involved in this project and a formal assessment and evaluation of these. Resources needed for the project are also identified and discussed.

Chapter 4 Details the design of the Voice Recognition and Control system as per the relevant theory explored in Chapter 2. The methods used in this process are explored and creation of the system followed all the way to the implementation stage. This involves investigating the processing algorithms, communication link, simulator code and command theory.

Chapter 5 looks at simulation performance involving Error Rates, rejection and system response times. Evaluation of the system is also conducted to seek out and identify problems such as delay and misinterpretation. Future improvements for the system are also somewhat explored.

Chapter 6 summarises the work done throughout the project and the final result. Further work and recommendations will also be made here.

Chapter 2

Literature Review

2.1 Chapter Overview

Over the years many people have dedicated time and money into researching the properties of speech which gives every person a sense of individuality. An understanding of how to manipulate these properties and effectively use them to transform the way we produce sound has been sought with high priority. With such an understanding, it has paved the way for voice to be used as a means of control and security. This technology has the potential, with the right technology, to replace a number of existing control methods to make our lives easier and quite possibly safer.

This chapter examines the basic concepts of speech, in particular its spectral properties and how they are obtained. It also looks at the way in which these properties can be manipulated in order to distinguish between a number of different utterances or words. Relevant literature will be explored to extract the techniques involved in speech analysis and detail their physical operation.

ABB's 6 axis robotic arms are also explored, with regards to their development and current capabilities. The software required to run these robotic arms is investigated and literature detailing the communication link theory between Matlab and RobotStudio is also explored.

2.2 The concept of Speech

The human voice is a very unique concept and varies greatly from person to person. Even though everybody is different in this way, it just so happens that the same technique is applied by everyone in order to produce sound. From the moment you first breathe air into your lungs muscles in your throat are preparing for whichever word you are planning on saying or in some instances, planning on not saying. No matter how you approach the projection of speech there are always 3 distinct ways in which speech can be excited (Plannerer 2005):

- Voiced excitation
- Unvoiced or Voiceless excitation
- Transient excitation

Voiced Excitation

When exhaling, air streams through the larynx, passing through the glottis (essentially the vocal chords and the space between them). During voiced excitation the glottis remains closed until enough pressure is present to cause it to open. In doing so, this creates a quasi-periodic vibration which can differ in both period and magnitude. This essentially creates the frequency at which the sound is released.

Unvoiced (or Voiceless) Excitation

Unvoiced excitation happens when the glottis remains open during this process. Turbulent air is generated through friction between the air and a narrow passage in the pharynx (oral cavity). A noise signal is formed that is dependent on the narrowness (constriction) of this passage, which in turn gives it a desired spectral shape.

Transient Excitation

This excitation happens when the muscles in the pharynx (oral cavity) are expanded and contracted to produce changes in air pressure. When opening the mouth, this air pressure gives a certain burst that gives definition to particular syllables in words.

In knowing these simple characteristics of speech we can now look at it from an analysis perspective. An utterance will be created using a range of these 3 excitations, with each excitation having important variables.

- Firstly we can look for a fundamental frequency as generated by voiced excitation.
- Secondly we can look for peaks in the spectral distribution of energy and zero crossings.
- Thirdly we can look for short silences produced over time caused by the build up of pressure during transient excitation (Kumar & Rao 2011).

From here we now know what to look for when conducting an analysis and with the right tools create a model that correctly identifies an utterance. The variables we have mentioned above can be found and manipulated using different techniques in signal processing.

2.3 Speech Processing

Today, speech processing has become an important part of people's lives. It is involved in nearly every part of people's day to day communication activities. Without this sort of understanding, the world would be a very different place and technology would be nowhere near as advanced as it is currently. Luckily enough, the technology that has been developed to model speech patterns is extremely accurate and efficient, allowing the reproduction and manipulation of sound in all types of industries. Knowing what was stated above in 2.2, it is easy to process and extract the essential information required to represent a signal (Muda, Begam & Elamvazuthi 2010).

2.3.1 Sampling

In order to obtain the information detailed in section 2.2, a short time signal must be recorded into a processing engine such as Matlab. This is generally done using a microphone and the user's voice, or the sound can be generated by a program or instrument. During recording, the short time signal (sound) undergoes an analogue to digital conversion at a pre-defined sampling frequency. For the case of the human voice, 8 or 16 kHz is an appropriate sampling frequency depending on the desired accuracy and quality. This data can then be saved as a physical recording in the desired format (.wav).

2.3.2 Pre-emphasis

Once the signal has been sampled and data is saved, it requires a certain amount of filtering in order for further processing to continue. The now 'digitised signal' is run through a pre-emphasis stage where the higher frequencies are magnified within the data set. This pre-emphasis, essentially eliminates the **-6 dB** per octave decay of spectral energy (Plannerer 2005).

The pre-emphasis filter can be defined in the following equation.

$$Y[n] = X[n] - 0.95\,X[n-1] \qquad \text{...equation (2.1)}$$

Where

X[n] is the digitised signal

Y[n] is the magnified output signal.

0.95 represents the percentage of the original signal assumed to be present in the following sample.

This filter ensures that the energies present in the signal at the higher frequencies are highlighted for further processing.

2.3.3 Framing

Once a time signal has been pre-emphasised, the need arises for it to be framed. Framing, divides the data into a series of overlapping segments that range from 20 to 40 ms in length. The importance of framing is based on the fact that, smaller time segments produce a more accurate stationary representation of the time signal ensuring the spectral estimate obtained later on is reliable. If the segments are to small there are not enough samples to give a reliable estimate and if the segments are too big the signal changes too much in order to obtain an accurate estimate (Lyons 2009).

The segmentation will allow the signal to be processed over smaller time frames of N samples. Adjacent frames become separated by M, which is less than N. Typical values for N are = 128, 256 or even 512 and that for M are of the order of M<N (i.e. 50, 100, 200).

2.3.4 Hamming Window

Now considering that the time signal has been segmented into a number of overlapping frames it becomes quite important that a hamming window is applied to each frame. The hamming window allows for the smooth transition between frames by reducing the side lobe that causes unwanted radiation in the frequency domain. This improves the time signal quality and removes unwanted frequencies caused by the framing process.

Using the output frequency from before Y [n], it is multiplied by a windowing equation W[n] in order to obtain the result Y1 [n].

This is shown in the following equation:

$$W[n] = 0.54 - 0.46 \cos(2\pi n/N-1) \quad \text{...equation (2.2)}$$

For:

$$0 \leq n \leq N-1$$

where:

N = the Number of samples in the frame (frame length)

Therefore **Y1 [n] = Y [n] x W[n]** **...equation (2.3)**

2.3.5 Discrete Fourier Transform

The next step in the process involves addressing each frame of N samples created above and converting them from the time domain to the frequency domain. This will enable the distinction of the key frequencies present in the sound as explained in section 2.2.

Currently the frames are in a form that is understood to be a function of time. Performing a Discrete Fourier Transform on each of these frames changes it to a function of frequency. This is also referred to as changing it to the frequency domain, where the frequency distribution and magnitude of this distribution in a sound is displayed. This form shows the most dominant frequencies present in the sound, which is an important factor is determining the major differences between two utterances.

The Discrete Fourier Transform can be applied as follows (Kumar & Rao 2011):

$$S_i(k) = \sum_{n=1}^{N} Y1(n) * e^{-j2\pi kn/N}$$ **...equation (2.4)**

for:

$$1 \leq k \leq K$$

where:

K = length of the DFT analysis per frame (standard 256 or 512)

2.3.6 Mel Frequency Scale and FilterBank

Knowing that the human ear does not follow a linear frequency resolution and instead builds several clusters of frequencies and sums the spectral energies within these clusters, the non-linear warping of the axis values can be modelled using the correct method. This method involves using the so-called *Mel Scale*. This particular scale aligns with those properties that need to be extracted from speech signals making it an effective analysis tool. The *Mel Scale* can be seen in the following diagram.

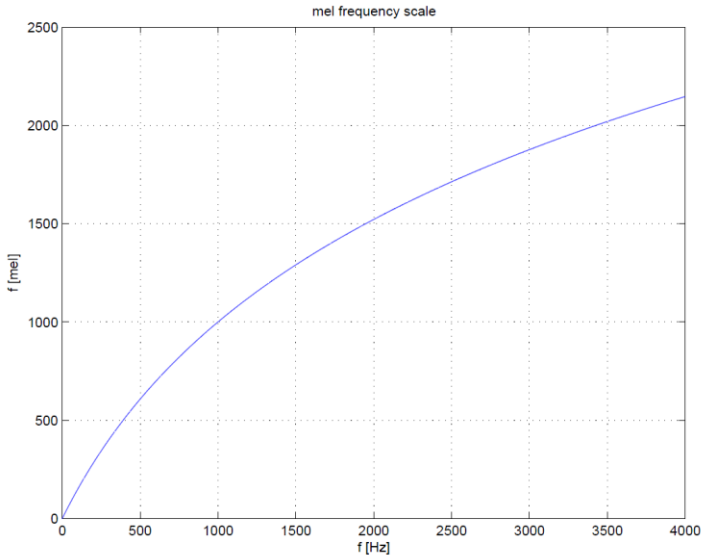


Figure 2 - Mel Frequency Scale

In order to find the right spectral properties within the speech signal a number of triangular filters are applied that calculate the sum of the function's spectral components. The equation below defines the Mel Frequency Scale as seen above; the filters are spaced according to this equation.

$$f_{mel}(f) = 2595 \cdot \log(1 + f/700\ Hz) \qquad \text{...equation (2.5)}$$

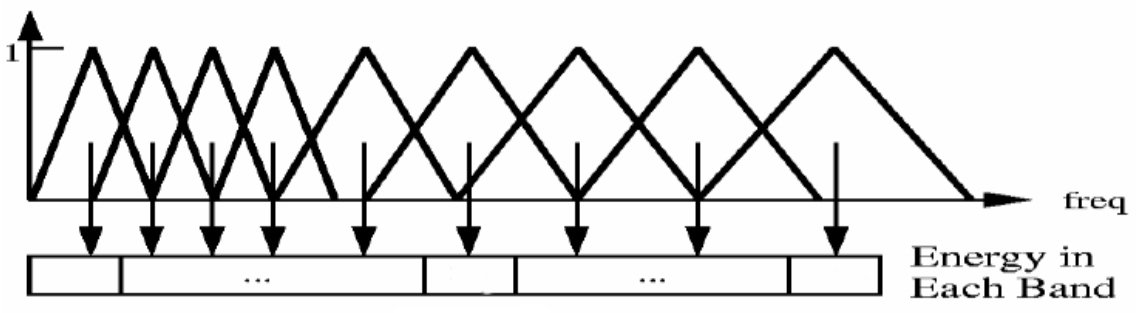


Figure 3 - Triangular Spaced Filters

Applying this filter bank will give a number of spectral values depending on the amount of triangular filters used. The standard number of filters in voice processing is set at 26. Each of these filter values will represent the sum of the spectral components found inside the filter (Lyons 2009).

2.3.7 Mel Frequency Coefficients

Now having Mel values from each filter output, the log of the filter bank energies can be obtained and can be converted back to the time domain using the Discrete Cosine Transform. This, in turn produces the so-called Mel Cepstrum Coefficients, which represent the acoustic vectors for this particular pattern of speech. Every utterance analysed in this process now is outputted into a sequence of acoustic vectors for further interpretation (Lyons 2009).

Following this common procedure there are a number of principles that can be applied to essentially perform the task of voice recognition. These are:

- Hidden Markov Model
- Dynamic Time Warping
- Artificial Neural Network
- Vector Quantisation

2.3.8 Hidden Markov Model (HMM)

The Hidden Markov Model has become the most popular way in which general purpose voice recognition is performed today. This popularity is due to its simplicity and computational feasibility in comparison to other methods. Essentially this statistical form of modelling outputs a sequence of symbols or quantities based on a stationary input signal. Considering that each window of speech being analysed is over such a short period of time, it can be seen as a stationary signal and applied using the HMM.

Given a range of stationary inputs, HMM uses a combination of comparison and prediction to match the input with a pre-recorded input model. It will determine likelihood based on the statistical distribution from each observed vector input from the Discrete Cosine Transform. The most likely model representation will then be found and can be used to display the utterance that was used as the input. In general the models used by this process are easy to train, pushing the capabilities of a HMM to deal with vocabularies of 5000+ words. Although, the theory shown above is of a basic nature and more refined techniques would be needed to increase vocabulary size, speaker adaption and noise resistance (Iqbal, Mahboob & Khiyal 2011). Re-creating this technique for a system described by this project would allow for great recognition results and larger vocabulary sizes, but the coding involved in this process may absorb too much time.

2.3.9 Dynamic Time Warping (DTW)

The Dynamic Time Warping process follows on from the basic process above, using the Delta energy and Delta Spectrum feature calculations to then find the optimal alignment between two time series. The special part about DTW is that these time series do not have to be of the same length and similarities can still be found through warping the series non-linearly. When applying this technique, a series of templates representing different utterances must be created and used as a comparison for the time series in question. The time series determined by the process above is compared with each template and a measure of the distance along the warp paths, created between the two series is taken and summed. The minimum summing distance determined by this process will then point to the most likely match in the templates and in turn be displayed or used to output a command. This process is effective in

distinguishing between two different people but may be less effective when distinguishing between utterances, especially if they are of similar patterns.

2.3.10 Artificial Neural Network (ANN)

This particular method used in voice recognition has been quite successful in isolated word detection, speaker adaption and phoneme classification. It uses a method similar to that of the Hidden Markov Model, but does not make any assumptions on the statistical properties of the input features. Instead of basing its prediction on probability and the pre-recorded input model, ANN's set out to 'learn' from the observed data and make a judgement based on what they know. This process does work well for isolated word detection but runs into problems when performing continuous word detection, limiting its power of computation. This would make it a good choice for the recognition stage of the project considering one word commands are to be used, although coding this particular type of network could be quite a hassle.

2.3.11 Vector Quantisation (VQ)

Vector Quantisation is a process of mapping vectors from a large vector space to a finite number of regions in that space. Each region or cluster contains a centre, called a codeword and this, along with other codeword's is used to form a codebook (Kumar & Rao 2011). When using VQ, a codebook is created through training a speaker's acoustic vectors into clusters for each specific utterance. Each speaker will have a specific codebook in which their utterances are trained. When it comes to recognising an unknown voice, the distances between the unknown vector and each codeword is calculated to give various VQ distortions or errors. The trained speaker corresponding to the least distortion is identified and outputted. This particular method is quite easy to implement in an algorithm, as well as being very accurate (Kumar & Rao 2011).

There are a lot more techniques and tricks in the industry today that allow voice to be used as a tool for control. These are used in very complicated algorithms that have been developed over a number of years, with results that far surpass those obtained using these basic principles. In order to explore the concepts of this technology these basic methods needed to be used in order to obtain an understanding and some basic results.

2.4 Speech as a form of control

Understanding the properties of speech and the way in which it can be manipulated becomes a very important step in conducting this project. Following this, is an understanding of how voice can be used in order to control a robotic device. Knowing what has been said in the sections above, some knowledge of voice and sound properties can be applied in order to define parameters for control.

It is well known that people have a voice that is unique to them and them only. It is quite common however for people to have a voice with the same pitch as another. Even though they may have the same pitch, they will have different spectral properties which affect both loudness and clarity. It is important to understand that when deciding to use properties of voice for a means of control, using consistent utterances of commands is vital, especially when basic methods of voice recognition are being implemented. It also makes it difficult to achieve successful results in the presence of noise.

Creating a list of commands for the purpose of control is a major part of the design aspect. Not only must these words be straight forward and clear, but they must avoid any similarities between them. Having similar sounding and pronounced words can affect the success rate of the algorithm and confuse one action with another; this is mainly due to the simplicity of the algorithm.

In order for voice to be used to control more than 1 variable, a number of considerations must be taken into account:

- As mentioned previously, the commands used and what they stand for
- The complexity of the algorithm, i.e. isolated word or multiple word recognition
- The speaker – algorithm may only be trained using 1 speakers voice

- Pitch or frequency can be difficult to manipulate by the average person
- Background noise

With these considerations in mind, the extent to which the voice can be used is limited and the flexibility of the system is impacted. In order to combat these limitations a number of solutions could be implemented. These include:

- Using more advanced algorithms – which would allow larger vocabularies, the use of multiple word commands and multiple speaker recognition
- Adding a simple tone or frequency generator to control an extra variable
- Adding numbers prior to or after the commands

The success of the voice recognition system will depend on which technique is chosen by the operator and how it performs in real time.

2.5 The 6 axis Robot arm

Robotic arms are a very popular choice for industrial work around the world in this present day. They are called upon to do a wide range of tasks whether it is, to simply lift, cut, weld or shape various materials and objects. Without them, the manufacturing industry would not only be less efficient and less productive compared to now but less successful and more expensive. It is clear that robotics will play an important part in the future of the human race and will require more complicated forms of control as a result.

2.5.1 History

The development of the 6 axis robot arm began in 1956, when two men named George Devol and Joe Engleberger established a company called Unimation. From this company came the first patent for the industrial robot arm which would be later sub-licensed around the world.

From this point on a number of developments have been made with regards to the robot arm, these include:

- 1959 – 'Unimate' – first ever robot arm created which was controlled by a program on a magnetic drum.
- 1961 – First robot arm installed at GM factory to assists in the manufacture and movement of car parts.
- 1969 – First automated spot-welding robot was installed at GM factory.
- 1970- First world conference on Robotics which gave researchers and engineers an opportunity to present their work and share ideas.
- 1973 – 3000 industrial robots in operation.
- 1974 – First microprocessor controlled industrial robot created in Sweden and First mini computer controlled robot arm marketed in Cincinnati.
- 1975 – First Cartesian-Coordinate robot used in assembly applications.
- 1978 – First 6 axis arm with its own control system.
- 1981 – Direct drive introduced for movement and later on electro-drive motors for greater control.
- 1992 – First robot packaging application (6 robots packing pretzels) sold.
- 1999 – Laser beam guiding introduced with robot arm
- 2003 – 800 000 industrial robots in action around the world.
- 2004 – Remote pendant used for control of arm.
- 2006 – changes to material properties allowed reduction in weight and more responsive
- Present Day – memory systems are becoming physically smaller but increasing in size (with regards to memory) and becoming faster (Robotics 2012).

These particular developments reflect the important changes that pushed industrial robots into modern society. A number of other developments have taken place over the years which have led to improvements in efficiency, speed, accuracy, capability and safety. These would include the rising development of computers and processors, discovery of new material properties and use of more efficient processes, just to name a few. With these increases over the years profitability has risen exponentially and adding Voice recognition compatibility would only increase this more. It would also open the robotics industry up to a broader range of applications and increase its flexibility (Juang & Tsuhan 1998).

2.5.2 RobotStudio

Ever since the birth of the robot arm, methods for its control have been ever developing as technology has been expanding. From simple programs to artificially intelligent algorithms, the industrial robotics industry has come a long way in its short lifetime. Today, ABB robotic arms use a software tool called RobotStudio, which implements a simple programming language to initiate a range of functions, controlling all aspects of all 6 axes. It allows the user to move the end of the arm to any point in the 3-Dimensional space it is mechanically limited to. This movement is based around the Cartesian coordinate system where a 3D vector is used to move the arm to different points in the space. In this particular case the arm has two 3D vectors that it requires input for, these being:

- The Base coordinate (x, y , z)
- The Tool coordinate (x, y , z)

The base coordinate moves the main body of the arm with reference to axis 1 (or the base) and the tool coordinates move the tool tip with reference to the centre of axis 6. The use of these two coordinate systems allows the arm to operate within an accuracy of approximately 0.01 mm. The speed at which this machine can operate depends on the users input settings but is also based on the programs effective use of zones and angles when moving between designated points. This variable speed is measured in millimetres per second (mm/s). As mentioned in the previous sentence, it also has the capability of smoothing out its travel path with its clever use of zones which is also user defined. These zones allow the arm to pass near a point in the path (or set of data points) instead of travelling straight through it, to prevent any ‘square cornering’ and smoothen out its desired path. This not only increases the efficiency of the robot arm but reduces wear on the axis joints (Robotics 2013).

The Cartesian coordinate system mentioned above, is the most effective method of executing a program at the desired accuracies that industry requires today. Another system, based on individual joint movement, can be employed as another form of control on these robotic arms. This system involves manually changing the effective angles of each axis rather than allocating a point in space to move towards. Implementing this type of movement does make it harder to move towards a fixed point with accuracy, especially if the angles for this point are unknown. Moving the joints individually can be more useful for jogging the machine towards a point and alternating the approaches in which the tooltip could reach this point. In saying this,

the joint movement system would be perfect for maintenance inspections and defect checking, as each axis could be individually moved with ease without having to worry about '*which point in space would best show me the limits of axis 4*'.

2.5.3 RobotStudio Coding (RAPID)

The RobotStudio platform is run by a language called RAPID which has many similarities to those techniques found in C, C++, Java and Matlab. In general, the language is quite simple and has been created to ensure easy access to the many functions of the 6 axis robot system. The RAPID language has a modular based approach, where many different routines can be created and called upon when needed, making it quite simple to add in routines whenever or however many times they are needed (Robotics 2013).

Each foundation program is held within a ***Module*** and is annotated as follows:

MODULE Socket	//Start of module → 'Module name'
ENDMODULE	//End of module declaration

Figure 4 - Module annotation

Following the declaration of the start of a ***Module***, all data values contained with the module must be declared. These values can be:

- Variable (**VAR**) – can be changed through program execution
- Constant (**CONST**) – cannot be changed once program is executed
- Persistent (**PERS**) – will retain value if it is changed during program execution and the program is restarted

Within each ***Module*** that is created a number of ***Procedures*** can be called which deal with:

- various movements
- socket connections

- error checking
- configuration settings

An example of this basic layout can be seen below:

```
MODULE Socket                  //Start of module → 'Module name'
VAR num speed:= 100;           //Variable declaration (datatype = number)
CONST string one:= "one";      //Constant declaration (datatype = string)
PROC main()                    //Start of procedure → 'Procedure name'
ENDPROC                        //End of procedure
ENDMODULE                      //End of module declaration
```

Figure 5 – RAPID program Layout

It is important to note that all declared procedures can be called from ***PROC main()*** and do not have to be created in order of operation. There order will depend on how they are listed within the main procedure and be executed chronologically (Robotics 2013).

Creating programs or procedures within RobotStudio can be done in 2 distinct ways.

Offline mode

In this mode the procedures are listed within an editor for user read and write access. Multiple functions are also available to the user to help create the procedures (e.g. Path creation). Simulations can be run through this mode and visualised on screen.

Flex Pendent

This mode gives complete control to the remote flex pendant where editing of the procedures can also occur. A number of quick setting functions are also available to reduce programming time and increase efficiency. Control of the robot can also be given to the Flex Pendent and the procedures run remotely from this handheld device.

Control must be allocated to either of these two methods before changes can be made, as one cannot be changed whilst the other is in control.

2.5.4 Previous Industrial Application

Implementing a technique such as voice control on this sort of system has been done before as seen in (Pires 2005). Here an ABB 6 axis robotic arm similar to that of Garfield was programmed and controlled to perform welding functions in the lab.

- The robot arm was controlled using Automatic Speech Recognition (ASR) software. Microsoft's speech engine, Microsoft Speech Application Programming Interface (SAPI), Microsoft's Speech SDK (v5.1) and .NET 2003 framework were the packages used for the software platform.
- This particular project used simple commands to firstly recognise the robots name (in case other robots were included), then it identified the user and finally it performed the commanded tasks.
- The user called upon routines to follow as opposed to directing the arm about (mainly due to the precision involved in the welding process).

This particular paper showed great success with this software implementation on the industrial robot and showed the potential of voice recognition programs within the industry (Pires 2005).

2.6 Communication Link (TCP/IP)

In order for any machine to communicate with another, a link is always established in real time for the handling of instructions and data. An essential part of this Voice recognition and control system is the ability for the two programs in question to communicate with each other seamlessly. In order to do so, a connection using TCP/IP protocol seems to be the right idea. This protocol is supported by both Matlab and RobotStudio and allows interaction between the two programs in a client/server relationship.

Socket messaging is quite a simple way for programs to communicate. It has the ability to send arrays of data and most importantly strings. This paves the way for commands to be transferred across the data link and recognised on the other side. Even using integers as a representation of a command could be much simpler.

Socket messaging can be implemented using the following code:

Server Socket ***t = tcpip('0.0.0.0', 30000, 'NetworkRole', 'server');***

IP Address — Port — Enables support for server sockets — Defines socket type

fopen(t);

Opens Socket connection and waits for client connection.

data = fread(t, t.BytesAvailable, 'double');

Reads data once connection is made.

Client socket ***data = ? (command data)***

Define 'data' ready for sending.

t=tcpip('localhost', 30000, 'NetworkRole', 'client');

fopen(t)

Create client interface and open it.

fwrite(t, data, 'double');

Write 'data' to the socket.

Once this is done the socket can be closed and the link broken once the program has finished sending data and/or commands.

2.7 Chapter Summary

This section has identified the literature outlining relevant methods involved in creating a Voice Recognition system for a 6 axis Robot Arm. It has explored the concepts of speech signal processing, application to application communication and the use of RobotStudio. The search for a greater form of control has been proven to be on-going and requires the investigation into more futuristic methods. The use of the voice as a tool for control is quite viable and developing a system which can demonstrate its functionality is of high priority.

Allowing the voice to control the actual individual movements involved with 6 axis robotic arms has been proposed by this project and will be investigated in the following chapters. The methods proposed in (Plannerer 2005) and (Lyons 2009), as well as (Kumar & Rao 2011) will be investigated for the purposes of real time voice control.

Chapter 3

Methodology

3.1 Chapter Overview

This chapter looks at covering the research and development methodology utilised for the development and implementation of the voice recognition system on Garfield, the 6 axis robot arm. This chapter also contains the risk assessment and identification of hazards and suggests suitable strategies for their minimisation. Any resources required for the completion of stages of the project are also outlined in this chapter.

3.2 Research and Development Methodology

The research and development methodology presented here was developed with the project objectives in mind. These objectives have been broken down into a series of major tasks to highlight what needs to be achieved.

Implementing a Voice Recognition and Control system on the 6 axis robot arm, Garfield has called for the design and production of a system that is simple to use, reliable and works in real time. This system needs to able to distinguish the relevant voice commands from sample inputs in order to produce movement on the robotic arm.

For the basic development of this system, there is the need to research and obtain, both *hardware and software* components. These two components will form the basis of the system and allow its completion.

At a *hardware* level, the system requires a microphone in order to record voice models and receive commands. It is to be connected to a suitable computer that has both a compatible sound card and the software programs in which it is to interface with. This computer will most likely be one that has access to the Robot arm simulation software and eventually change to the computer in charge of operating Garfield.

On a *software* level, the system requires a platform that:

- Waits for and receives a range of inputs
- Manipulates these inputs into a form that can be compared and identified
- Produce an output that reflects the true nature of each input (in real time)
- Communicate this output over a TCP/IP socket between 2 separate pieces of software
- Recognise these outputs and use them as viable inputs in order to create movement with the arm.

The requirements stated above are an important part of the software layer for this project. Choosing the right programs for this purpose and considering their compatibility with each other is another major step in this process.

A simple block diagram of the interaction between the hardware and software levels is shown below.

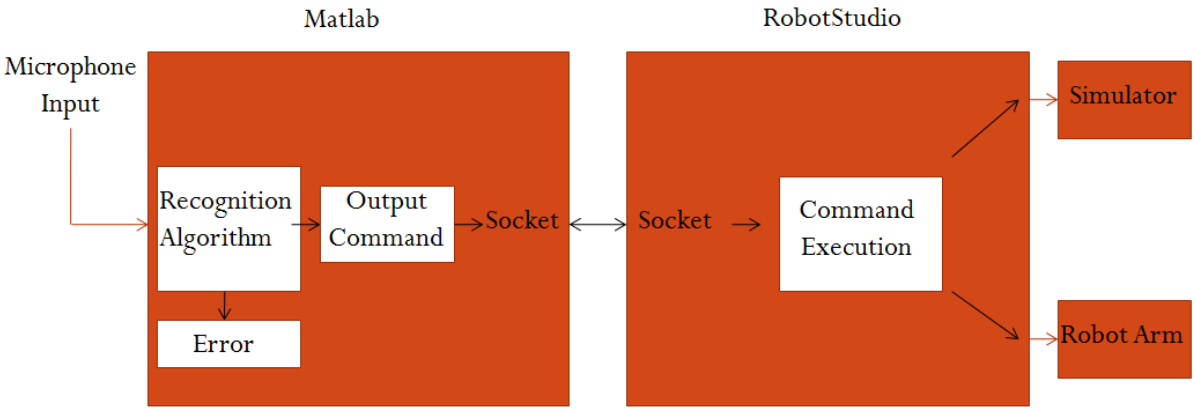


Figure 6 - System Block Diagram

In order to evaluate the performance of this system, the design of suitable testing methodologies and analysis strategies have to be considered. The tests involved with such a system should present information that reflects recognition accuracies and rejection rates, as well as information on response times for real time operation. Parameters used during testing will be kept constant and those uncontrollable variables dealt with to ensure the integrity of the results. Evaluation of the system based on the results achieved will be included and be used to form recommendations for further work and future development.

3.3 Task Analysis

All in all the development of this project can be refined to the following processes:

1. Identification of required hardware and software for this application.
2. Research and development of Voice recognition techniques.
3. Design and Implementation of Voice Recognition Algorithm.
4. Design and Implementation of Robot arm code.
5. Design of communication link between programs (integrate the system into one).
6. Train, Test and evaluate the system in simulation
7. If time is available, test on Garfield itself.

Ensuring these 7 steps are achieved and appropriate results are obtained, the project can be deemed successful. They will act as milestones in the development of the system and in turn, indicators of project progress. The following is an analysis of the details of each of the tasks above and identifies the resources required for each.

3.3.1 Hardware and Software identification

A number of items must be identified and obtained before different parts of the system begin development.

In saying this, the voice recognition system will require the following resources:

- Microphone – model yet to be determined (preferably of high quality)
- Computer – Lab computers with access to required simulation software. The computer must be appropriate with regards to processing speed and sound card quality.
- Matlab – most recent version or closest to, with complete toolbox
- RobotStudio – software required to run ABB 6 axis robot arms
- TCP/IP socket protocols – for communication between programs

These resources will be used throughout the following processes in order to achieve completion.

3.3.2 Voice Recognition Algorithm

The system will require the use of appropriate voice recognition techniques mentioned in chapter 2. It is imperative that these techniques are developed into an algorithm to create the front end of this system. The algorithm must communicate and receive input from the microphone in order to initiate the detection of specific commands. It must also be able to reject noise and unknown commands safely or at an effective Error Rate.

From here it will be required to interface with the simulation software required in testing for the 6 axis robot arm. Incoming inputs will need to be recognised and matched with specific outputs to communicate across this interface.

This algorithm will use functions associated with the Voicebox Toolbox (Brookes 1998) to create a form of voice recognition.

A simple block diagram of the two features of the algorithm is shown below in figure below.

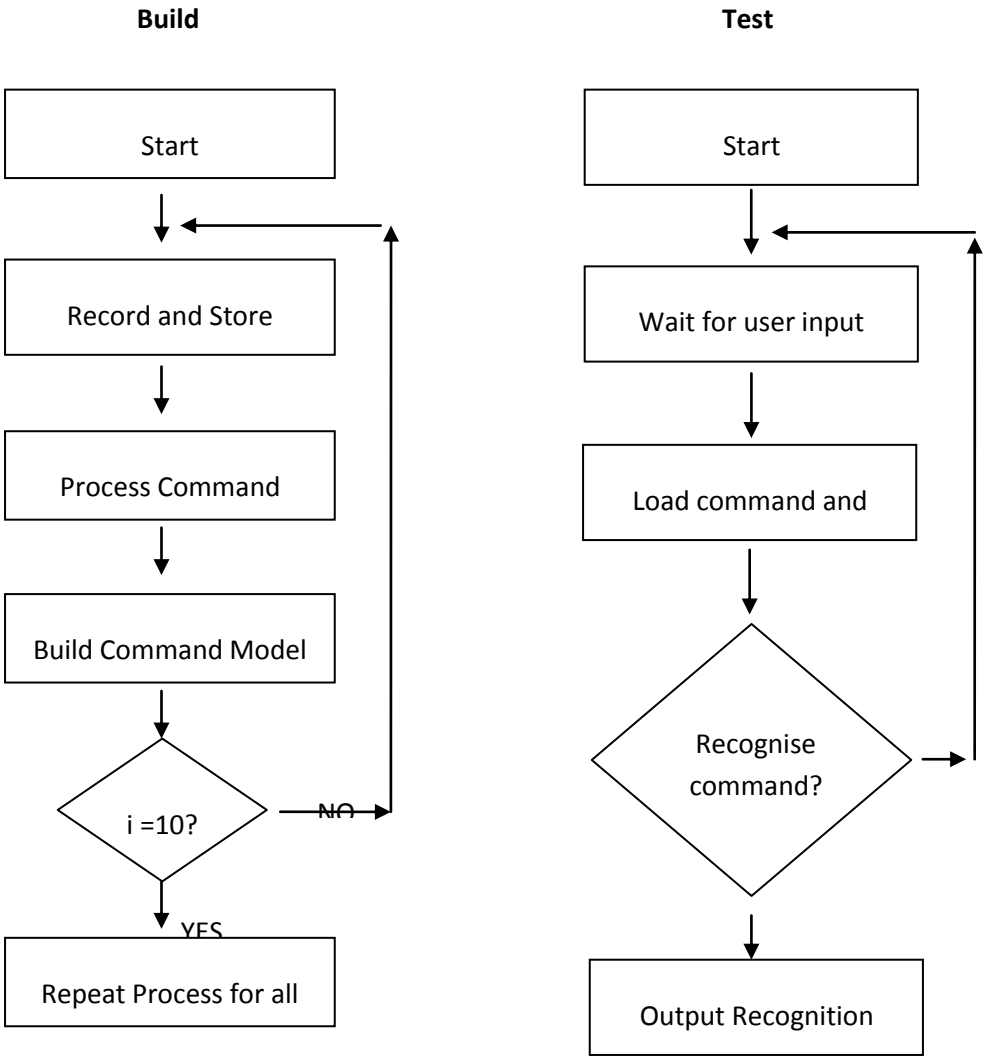


Figure 7 - Simple Process Diagram

Building the command models is done using the simple process simulated by the block diagram on the left in figure 7. 10 utterances must be recorded for each command to achieve

fair results. Testing an utterance against these command models is shown by the block diagram on the right.

This will require the following resources:

- Microphone
- Computer with pre-installed Matlab software and coding

3.3.3 Robotic arm Coding

This process will involve preparing a program based on the ABB programming language used by RobotStudio. The routines created in this process will perform command specific actions that are called for by the inputs from the Matlab algorithm. The actions could be any of the following:

- Axis 1 – Left or Right
- Axis 2 – Up or Down
- Axis 3 – Up or Down
- Axis 4 – Clockwise or Anticlockwise
- Increase or Decrease Speed
- End procedure
- Perform Demonstration

The coding will be based on the following principle:

- *Each command will move the robot a specific distance in whichever direction the command defines. It will not just call a program with pre-defined points in the 3D space to execute, unless demonstration of this idea is needed.*

It is important that consideration of the final integrated system be taken into account when designing this code, so that it can adapt easily when being controlled by another program.

The resources required for this include:

- Computer with pre-installed RobotStudio Software
- Input source to test coding
- Manual or Library of commands and routines
- Network license for Software

3.3.4 Communication Link

This step in the project is quite important for the complete integration of the system. Without a link between the two programs for the voice recognition and RobotStudio code, the system will not function. This particular link will use a tool common to both applications that can be applied to send and receive messages. TCP/IP socket messaging is the ideal method of communication between these programs as it uses Ethernet connection protocol to relay messages between the client and server. It is essential that this link is developed quickly to allow more focus on the other parts of the project.

3.3.5 Testing and Evaluation

To validate the performance of the voice recognition system, testing must be conducted under a number of conditions. These conditions include:

- no background noise – generally total silence with only the user speaking
- Background noise – the user speaking with industrial background noise
- Background noise – random noise/commands

Applying these conditions to the algorithm in question and comparing the results will allow appropriate evaluation of the system. Once the system is deemed reliable enough, it can be applied to Garfield in a real life situation.

Testing the response time of the system is another important part of the project, as it will give the user an idea of how well the system would suit specific jobs that may require its use. These response times will be calculated over a range of movement increments and compared.

From the results of this stage, a qualitative analysis can be conducted and future improvements can be recognised and applied to the system for more testing. This will increase the reliability of these methods and create a better platform for further expansion.

The resources required for this task are:

- All system components
- User voice input

3.4 Consequential Effects

3.4.1 Sustainability

The Institute of Engineers Australia (Engineers 1997) provides a set of guidelines that addresses the aspects of sustainability within Australia. This particular institution forms the largest collective group of professional engineers within Australia with world renowned accreditation available for its members. The guidelines they have laid out form the basis of evaluation processes that relate to the environmental, socioeconomic and cultural sustainability of a project, over its lifetime.

When discussing the affects of Voice Recognition and Control technology, the industries in which it is being implemented need to be taken into account. The industrial sector is a major user of 6 axis robot technology in the present day and this particular sector is responsible for a lot of finite resource consumption and waste. Implementing this sort of technology into this sector could possibly improve methods of control which could improve efficiencies and in turn lower resource consumption and waste products.

With regards to environmental safeguards within the project dimensions, avoiding excessive emissions created by the equipment could be of benefit, as well as utilising power saving techniques when operating in the laboratory.

This sort of technology is one that could slowly develop into that seen in current day science fiction movies where robots are fully autonomous and have a mind of their own. Generally these types of robots are portrayed as out of control and dangerous but on the other hand these artificial humanoids could be a great use in everyday life. Performing those tasks which are boring, repetitive and even dangerous. Society by this time, would be completely different than what it is today and there is no way of knowing exactly how it could have impacted.

The most likely people impacted by this technology would firstly be those involved with industrial robotics and then quite possibly it could evolve and impact those everyday people. In this case, before the technology is even implemented around the world a consensus should be reached by the people to decide whether or not they feel it is appropriate for such autonomy to be present. Will it be ethical for us to create and almost living machine?

Project costing for this project is quite low considering the hardware and software required for all aspects of the project have been made readily available by the university.

The impact of this technology on the labour force could be quite substantial. Due to the improved interaction between machines and human beings more jobs will be semi automated taking away the number of people required to do specific processes. On the lower level, low income earners will be affected dramatically but on the higher level, big business will prosper with increased productivity and efficiency.

Overall this project has the potential to influence a large proportion of people and provide a better understanding of robotics and automation in the process.

3.4.2 Safety

There is a number of safety aspects involved with this project that could result in injury or death. The likelihood of a fault or bug in the developed code is quite high, raising the risk put on the user when operating Garfield. It is possible for a command to be misjudged by the robot arm when it’s in operation, which could cause damage or injury (i.e. the spillage of harmful materials in industry). Parts could also be broken and equipment damaged due to

crashes and put the user at risk. Measures have been taken to ensure the safety of users in case of such an occurrence. Shielding surrounds the 6 axis arm in case this sort of error reaches the real life implementation stage, otherwise bugs and errors are generally picked up during simulation.

The equipment used in the project was required to conform to relevant Australian and international safety standards. Where it was necessary, precautions were taken to minimise the risk of the operator and any others in the vicinity.

3.4.3 Ethical Considerations

These considerations are based on the code of ethics provided by Engineers Australia which lists guidelines for engineering practices. The guidelines cover such qualities as integrity, leadership, competence and sustainability. It is essential that this code of ethics be followed closely at all times during the project. This will ensure that the outcomes reached and ideas presented are done with the best interests of the community and environment in mind.

3.5 Risk Assessment

The major risks involved in conducting work on this project need to be carefully considered. Once these have been identified, strategies will need to be developed in order to minimise the risk and prevent the occurrence of injury or harm. Not only will these risks be a danger to the user but also to the project itself. Any event that causes work to be lost or damaged during the project must be identified as a risk and dealt with appropriately. The details of these risks and there appropriate solutions will be found in Appendix B.2.

3.6 Research Timeline

The tasks laid out in the Gantt chart found in Appendix B, show the expected start and finish times of particular sections of the project and reflect those pointed out in the methodology.

3.7 Chapter Summary

Research and Development methodologies have been explored throughout this chapter, ready for further expansion in later sections. The project was organised into a number objectives that must be completed in order to achieve success, along with the resources required for each of these. The objectives were also scheduled within a timeline to set dates of achievement.

A formal risk assessment was undertaken, underlining those risks that are most likely to hinder project completion and cause injury or harm. Strategies were developed in order to manage these risks and ensure the safe completion of the project.

Chapter 4

Design & Implementation

4.1 Chapter Overview

The following chapter contains detail on the design and implementation of the Voice recognition and Control system, based on the methodologies and theoretical content defined in the literature in Chapter 2 and requirements highlighted in Chapter 3. This chapter will outline the processing methodology used by the recognition system, the interfacing method used for communication with the robot software and its implementation as a complete system.

4.2 Voice Recognition Methodology

The voice recognition and control system used in this project requires a certain level of understanding, about the processing techniques involved with each of the utterances used. In order to expand this understanding, the need arises to explore the processing techniques used on the data and how it is manipulated to create a voice based recognition system. The details of this voice based systems implementation will be discussed in later sections.

The signal processing techniques used for this project (as seen in section 2.3) can be divided into the following stages:

- Pre-emphasis
- Silence Detection and Data reduction
- Framing and Windowing
- Discrete Fourier Transform

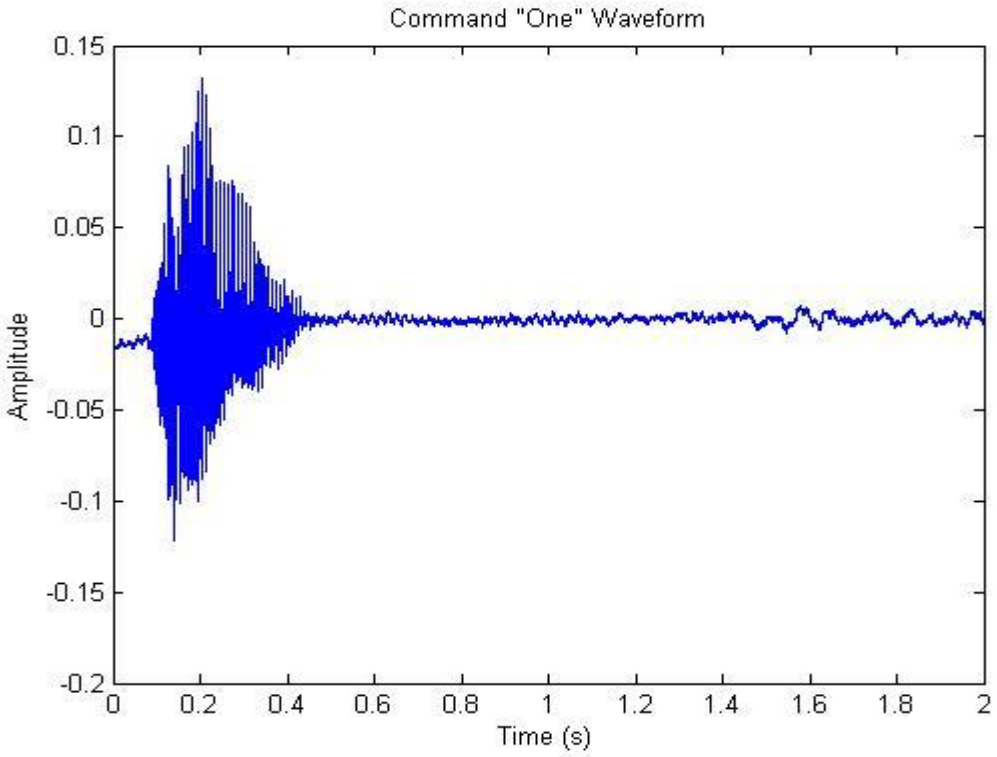
- Periodogram-based power spectral estimate
- Mel Filter Bank
- Log filter bank energies and Discrete Cosine transform (Finalised MFCC)
- Model Building
- Recognition using Euclidean Distance

It is important to understand that the following processes aim to extract those features in the voice that are essential in identifying the linguistic content of the speech and also discarding those features that represent things such as background noise and emotion.

These stages will now be investigated in detail.

4.2.1 Pre-Emphasis

In order for the processing techniques to begin, 10 voice samples of each command were recorded at a sampling frequency of 16 kHz and stored in 'command' arrays. These voice samples were selected to last for the duration of 2 seconds, considering that the average



utterance of a word only lasts between 200 and 800 milliseconds.

Figure 8 - Speech Signal of Command 'ONE'

The above figure shows the utterance of the command ‘ONE’ before pre-emphasis. In its current form little can be done in terms of recognition, therefore it requires further processing and normalisation to obtain the data relevant to the problem.

To prepare it for further processing and obtain the best results from each signal, each utterance is run through a pre-emphasis filter with the following characteristics:

$$Y[n] = X[n] - 0.95\,X[n-1] \qquad \text{...equation 2.3}$$

Where Y[n] = output signal

X[n] = input signal

The value of 0.95 is the standard value for a pre-emphasis filter and can be changed if higher frequencies need to be emphasised more or less.

The result from the filtering process can be seen in the figure below.

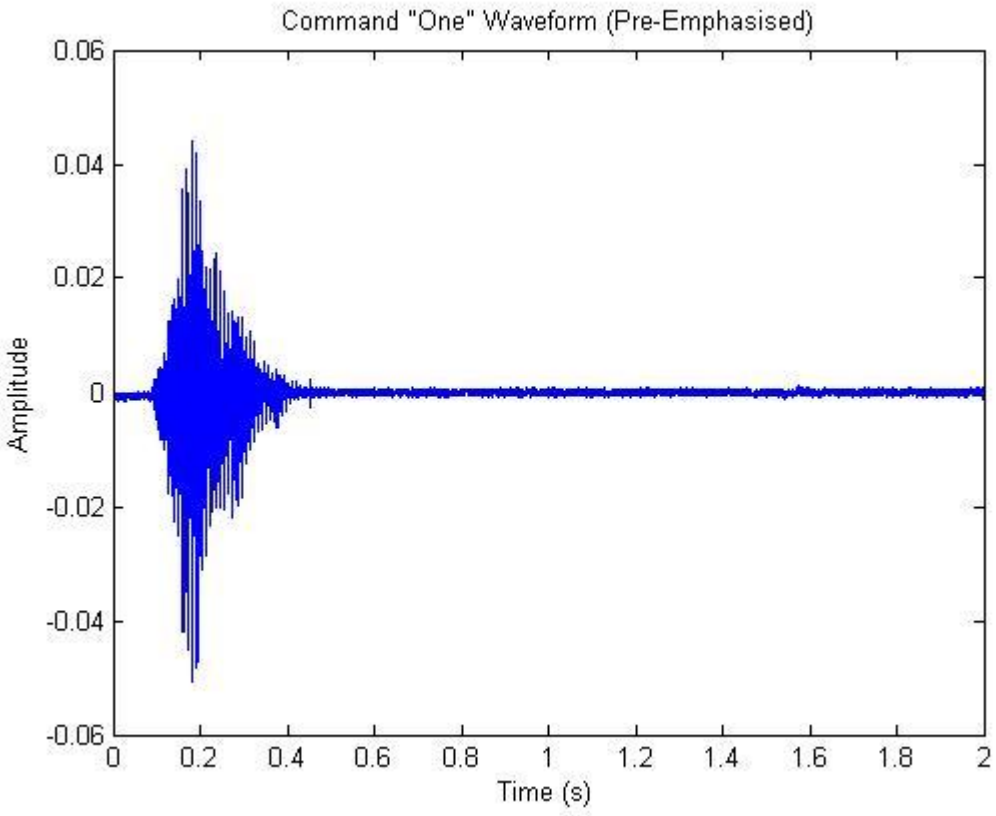


Figure 9 - Command "ONE" Pre-emphasised

4.2.2 Silence Detection and Data Reduction

Following the process of Pre-emphasis, the speech signal can now be shortened to obtain the most important samples for the recognition process. This means reducing the total number of samples in each utterance by removing the 'silence' before each utterance. In order to do so, a small detection technique is applied to the front end of the signal.

This detection technique involves:

- Setting up a threshold to determine the difference between the 'utterance' and 'background' or 'ambient' noise.
- Searching the front end of the data sample for breaches of this threshold.
- Determining whether the breach is actually the 'utterance' or some form of random noise.

The threshold limit selected for the project was an |amplitude| of ***0.04*** **units**. This threshold was determined to be a reasonable borderline between ambient noise and the utterance itself. The following excerpt of code from Appendix D shows the detection process.

```
% Silence Detection

% Wait for clear start of signal

while abs(Preemphtest(i))< 0.004 || abs(Preemphtest(i+20))< 0.004 ||
abs(Preemphtest(i+40))< 0.004 || abs(Preemphtest(i+60))< 0.004 ||
abs(Preemphtest(i+80))< 0.004 || abs(Preemphtest(i+100))< 0.004

    i = i+1;

end
```

Figure 10 - Silence Detection Code

It can be seen that whilst the conditions remain under the threshold the 'i' counter will continue to increment. Once all of the conditions are breached in the same iteration, it is clear that some form of utterance is present in the data and the increment is stopped.

From here the code will take the incremented value (where the utterance starts) and extract the succeeding 8000 samples from each utterance data set.

```
% Signal will start from the calculated position
atest = Preemphtest(i:length(Preemphtest));


% Reduce samples from 32000 to 8000 for remainder of processing (as
command
% will be present within the 8000 samples)
atest(8001:length(atest))= [];


Preemphtest = atest;
```

Figure 11 - Data Reduction

This reduces the amount of samples from 32000 back to 8000, which will decrease the calculation time and improve the response time. The result of this process can be seen in the following figure.

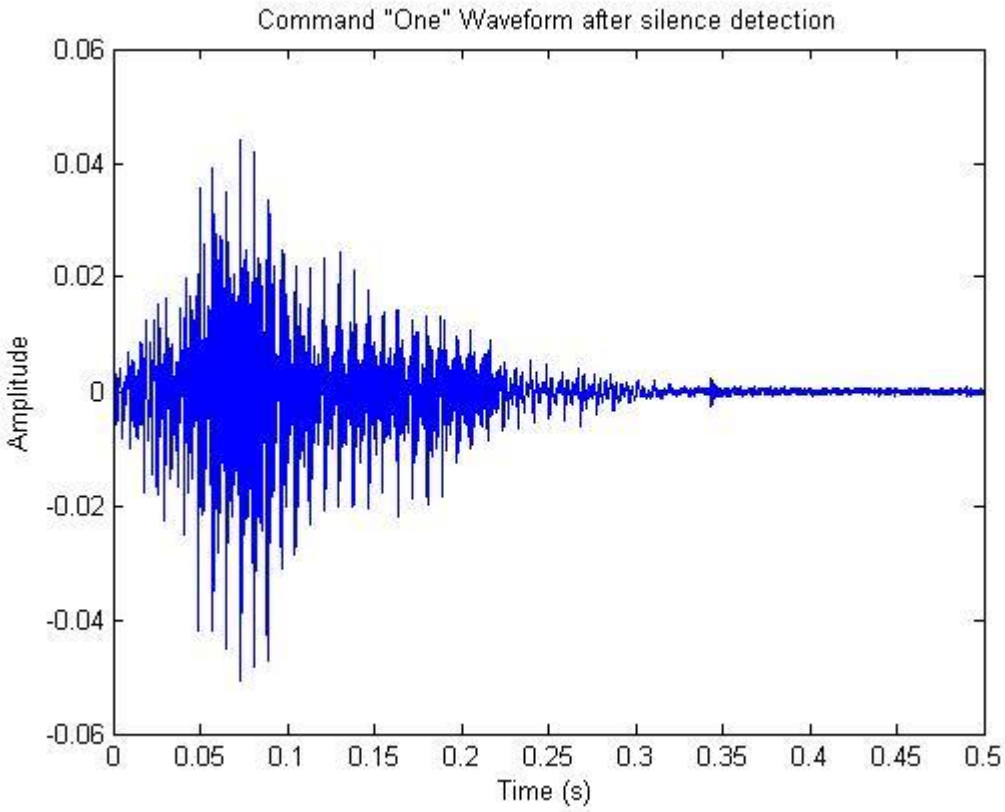


Figure 12 - Command 'ONE' after silence detection

Employing this simplistic method comes with its disadvantages, as it can cause the code stop at any instance of random noise before the utterance has occurred. It can also stop if ambient noise is present that exceeds the threshold. A degree of care needs to be taken in this case to ensure almost no noise is present during the recording of utterances for the model building stage and that thresholds are adjusted to reject noisy commands in the testing stage.

4.2.3 Framing and Windowing

The next step in the process is to break up each dataset of 8000 samples into a number of M frames of size N, with a step size between these frames of Δ = N/2. The following parameters could then be chosen:

- Segment Length = N = **320 samples**
- Step size = Δ = **160 samples**
- Number of frames = M = **49**

These parameters produce frames that are 20 ms long which is believed to be the optimum length for obtaining a good spectral estimate from the frame. If the frames are any longer, the signal changes too much within the frame to get an accurate stationary estimate of the time signal. If they are any shorter, there are not enough samples in the frame to obtain a reliable estimate of the time signal properties.

The framing process can be seen in the following diagram.

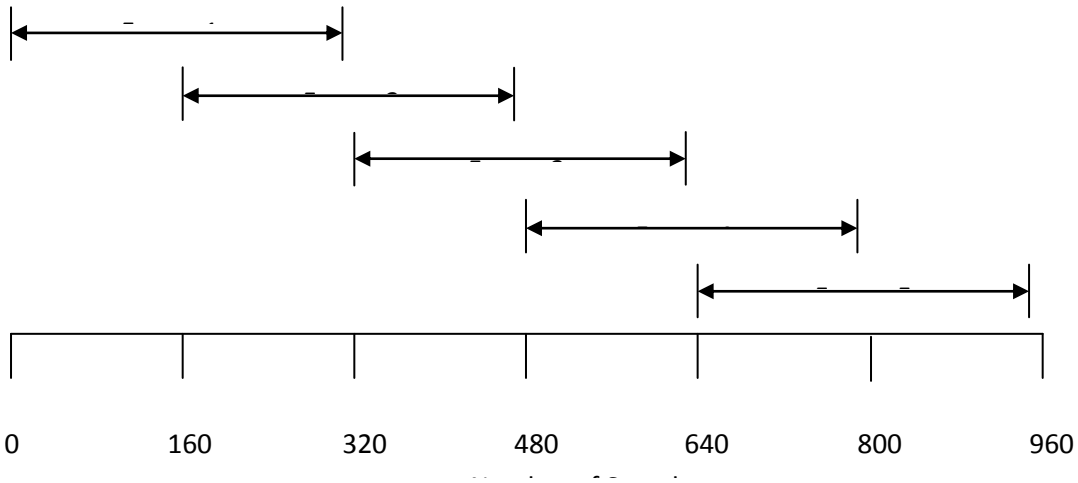


Figure 13 - Framing Diagram

Once each frame has been assigned to values from the dataset, a window must be applied to each frame. In this case the 'hamming' window has been selected as it is the standard window type for any Mel Cepstral Feature extraction. The fundamental equation for this window can be found in section 2.3.

A 320 point hamming window can be seen in the following diagram.

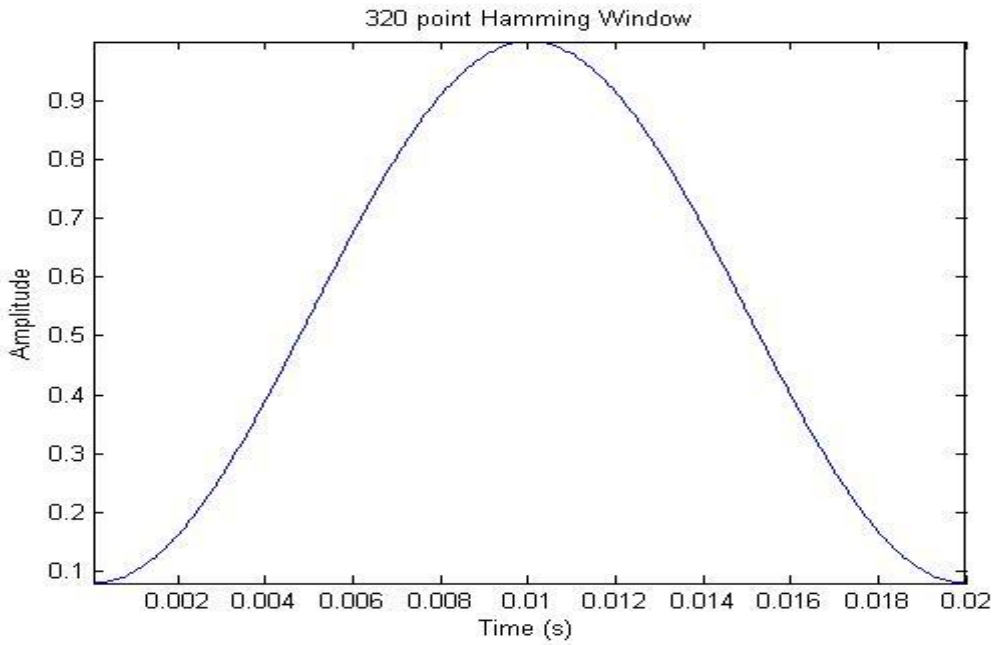


Figure 14 - Hamming Window

Applying this window to each of the frames is done before the Fast Fourier Transform to smooth out the response on the edges of the frames. It essentially minimises the side lobe and prevents any unwanted radiation in the frequency domain, improving the quality of the sound and accuracy of the feature extraction (Plannerer 2005). The result of multiplying the hamming window with the first frame is shown below.

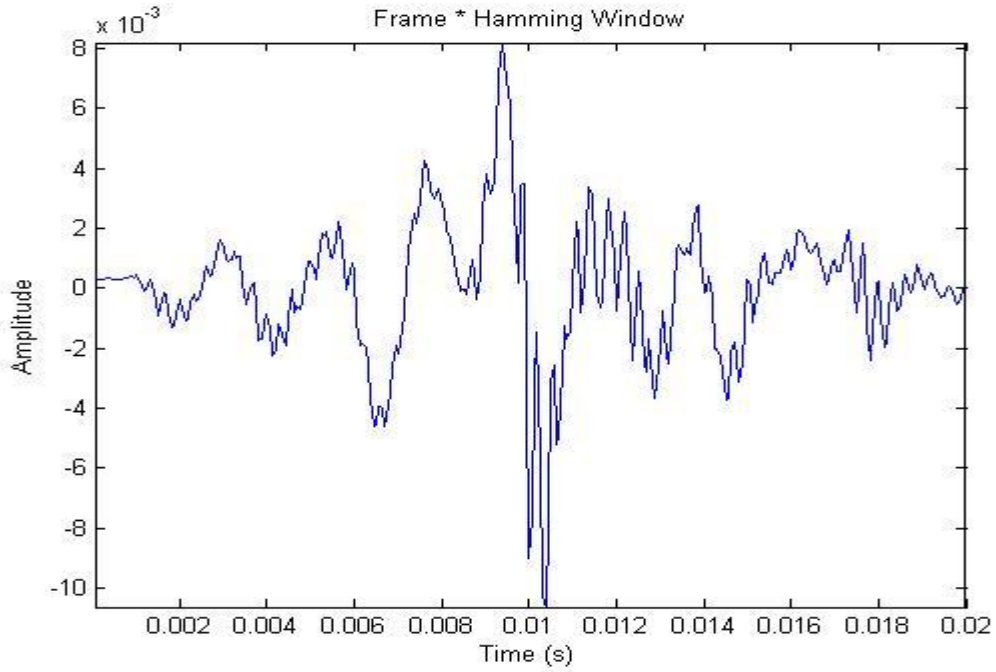


Figure 15 - Hamming Window applied to the frame

4.2.4 Discrete Fourier Transform

At this point in the processing stage the data is ready to undergo a change to the frequency domain through the Discrete Fourier Transform (DFT) as defined earlier in section 2.3.

$$S_i(k) = \sum_{n=1}^{N} s_i(n) * h(n) * e^{-j2\pi kn/N}$$

for

$$1 \leq k \leq K$$

Where:

- 'i' denotes the current frame
- 'N' denotes the sample size
- 'K' denotes the length of the DFT analysis
- s(n) is the time domain signal
- h(n) is the hamming window
- $S_i(k)$ is the complex DFT

This change of domain allows the spectral components of the utterance to be accessed and manipulated for the recognition process. The DFT can be performed in Matlab using a Fast Fourier Transform algorithm that reduces calculation time.

A 512 point DFT is performed on each frame to reveal a set of 512 coefficients, in which the first 257 are kept and the rest disregarded (remaining coefficients have little importance in the feature extraction process). For this particular task outside help can be obtained through the use of **VOICEBOX : Speech Processing Toolbox** (Brookes 1998). The *rfft.m* function, that can be found in Appendix D, was used in order to do all of the above processing in 4.2.4 in one easy step.

4.2.5 Periodogram based Power Spectral Density (PSD) estimate

The PSD estimate is a calculation based on those formulae defined earlier in section 2.3. This process, along with the DFT, acts to mimic the operation of the cochlea (the organ in the human ear). It extracts those frequencies that human's base their perception of speech upon, which becomes very useful when dealing with automatic speech recognition (ASR).

The 257 coefficients obtained from the DFT in 4.2.4 are applied to these calculations to obtain a Power Spectral Density Estimate for the current frame. This is done by taking the absolute value of the complex Fourier transform, squaring it and multiplying it by 1/frame length.

$$P_i(k) = \frac{1}{N} |S_i(k)|^2$$

From the above equation a set of 257 coefficients representing the PSD estimate are calculated.

4.2.6 Mel Filter Bank

The Mel filter bank is a bank of linearly spaced, overlapping, triangular filters that stretch across the frequency spectrum. The Mel scale to which these filters are spaced aligns with that of the human ear. This makes the algorithm more sensitive to pitch changes at lower frequencies and essentially simulates the operation of the human ear. The diagram in the figure below shows the simple concept of the Mel filters and their application to the PSD estimate (Lyons 2009).

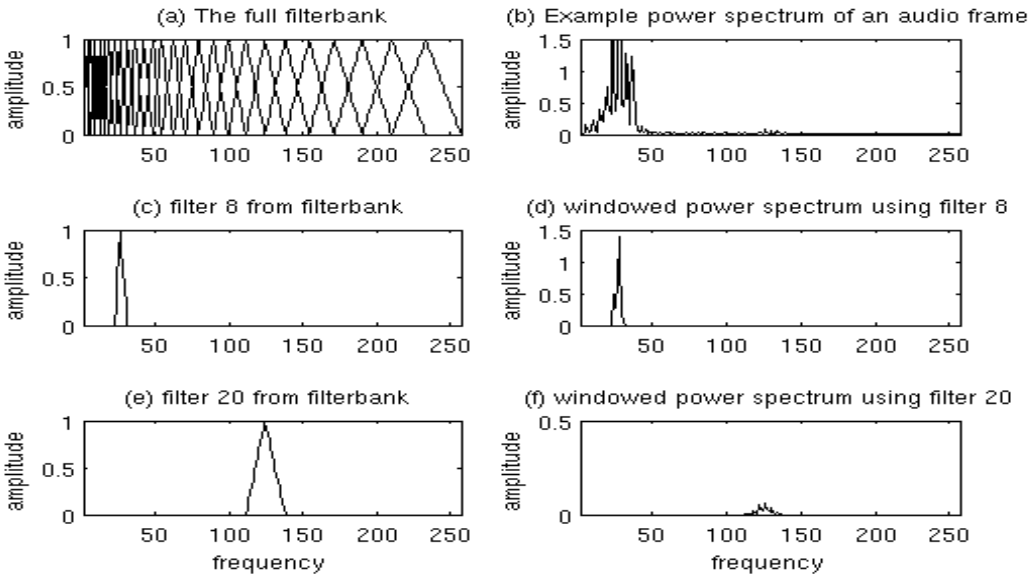


Figure 16- Mel Filter Bank (Lyons 2009)

In order to create a filter bank based on the data obtained previously, another function, *melbankm.m* was used. This function was also taken from the **VOICEBOX : Speech Processing Toolbox** (Brookes 1998) to obtain a filter bank of 26 filters (26 is the standard number). Multiplying this with the result from the *rfft.m* function would now obtain a set of 26 coefficients demonstrating the sum of the components within each of the 26 triangular filters.

4.2.7 Log filter bank energies and Discrete Cosine Transform

In keeping with the spectrum of the human ear the log of the filter bank energies must be taken. This due to the fact that loudness is not heard on a linear scale and the respective energy levels may give false information when comparing on a linear scale.

Since the application of the filter bank, correlation between the data has increased due to the overlapping triangular filters. In order to decorrelate the data and reduce the effects of correlation the Discrete Cosine Transform is applied to the frame, in turn achieving an accurate representation of the feature vectors that needed to be calculated. These, in essence, are the Mel Frequency Cepstral Coefficients (MFCC) (Lyons 2009).

For the purpose of this speech recognition algorithm the first 13 coefficients of this result relating to spectral power were kept for the recognition process and the rest discarded. This meant that 13 coefficients per frame were kept and collectively stored in their respective command matrices.

4.2.8 Model Building

The creation of the command 'Models' is an important part of the recognition process, without appropriate Models for comparison, the system will have no chance of determining the correct command for output to the RobotStudio simulation package. In order to build the command matrices, each set of 13 feature vectors (1 column) from each frame was placed side by side for each utterance of each command as follows:

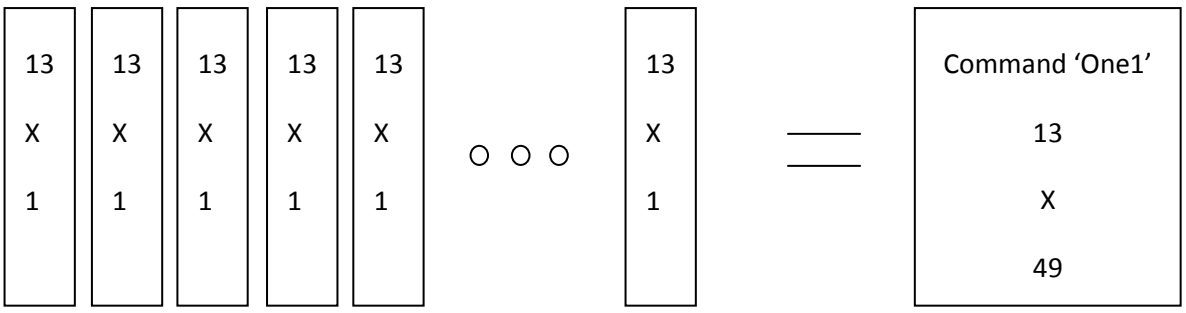


Figure 17- Model Diagram of utterance 'One1'

Once all of the matrices were built for each individual utterance a matrix was built for each command, meaning one command matrix would contain 10 utterances matrices above. This can be seen below:

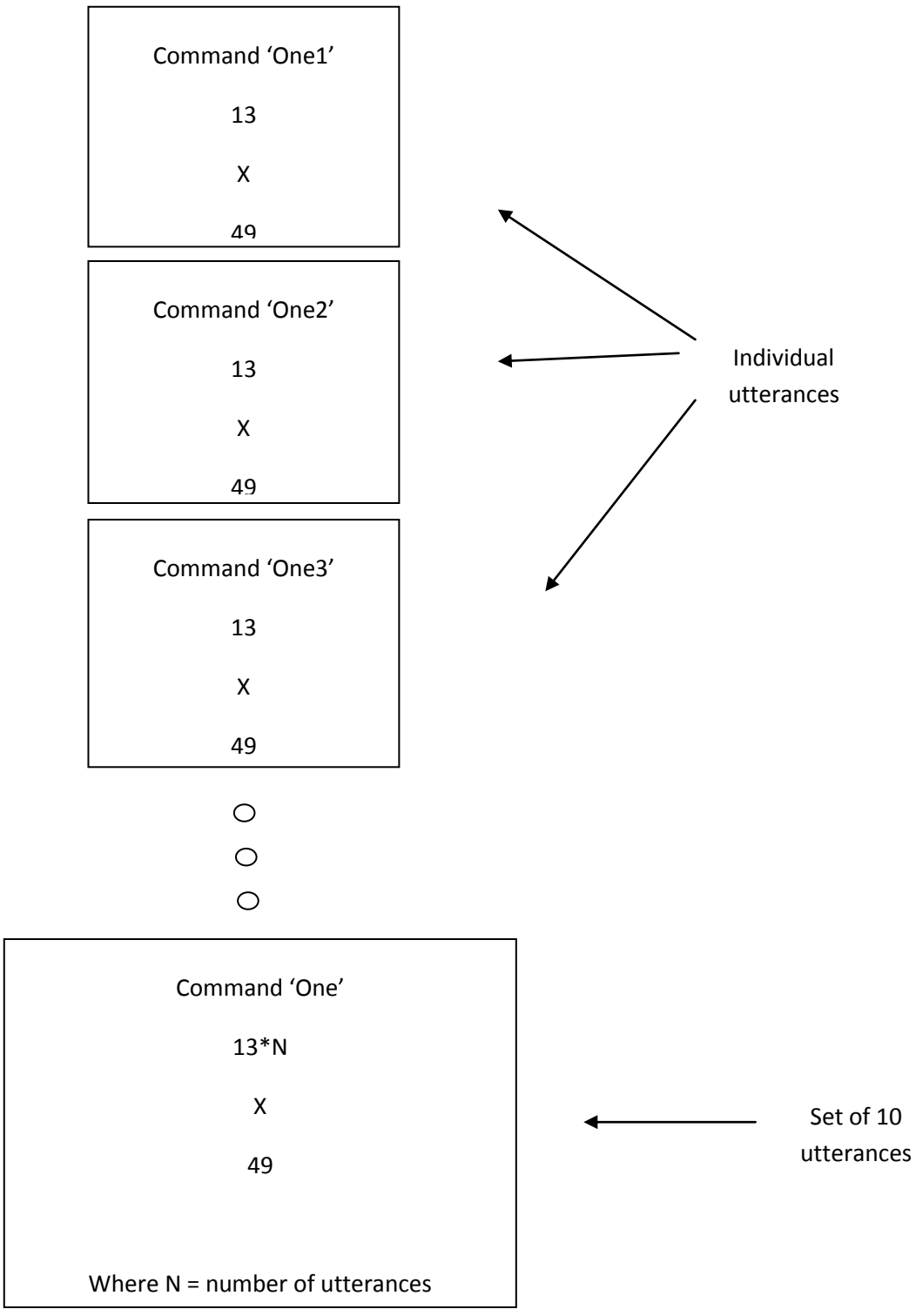


Figure 18- Command Model 'ONE'

This process was done for each and every command.

4.2.9 Recognition using Euclidean Distance

The most important part of the Recognition process is determining the right method to compare models and determine the correct output. For most current voice recognition packages, Hidden Markov Models (as explained in section 2.3) are used quite successfully for recognition and model comparison. This method is based on probabilities and can be quite confusing to anyone planning on using it for the first time. Originally it was planned to use a Hidden Markov Model but due to timing constraints and knowledge barriers it became quite clear that implementing such a recognition technique on this project would be quite the task. As it so happens, a simple method of utterance distinction was chosen to be used and has produced impressive results.

This particular recognition technique uses the Euclidean Distance calculation between the feature vectors of the utterance in question and the databases of utterances created in 4.2.8.

The calculation is as follows:

$$\sqrt{\sum_{n=1}^{N} (utterance_n - test\ utterance_n)^2}$$

The error created from each comparison between the test utterance and the utterances of the model is stored in a matrix. From these, the minimum error is then computed for each of the command matrices and the smallest error amongst these is chosen as the most likely solution.

If this error happens to exceed the threshold, the test utterance is considered an unknown command and outputs an error; otherwise the corresponding command is identified and outputs the command from the recogniser.

4.3 Command Action Methodology

Any robotic system requires a number of techniques to fully utilise the functions of the device that is being controlled. This could be done through manipulating speed, direction, program efficiency and/or execution time. This section explores the processing techniques and commands used to obtain the desired response from 'Garfield' when presented with commands from the voice recognition process.

4.3.1 Command Structure

All robots respond to pre-determined commands or procedures. These procedures can range from simple movements to complex ones and need to be simulated before applying them in real world situations. Ensuring that the right movements are chosen for these commands is quite essential, as any misinterpretation of a command and its action could cause injury or damage to the system.

Although, in the event of a fault or breakdown, the system does have pre-installed limitations and warnings to disrupt program execution and prevent any further movement of the robotic arm. It becomes quite important to ensure that some form of extra protection against any unknown commands be implemented and tested, to increase the safety of the system.

The commands chosen for this particular system have remained quite basic as to increase the integrity of the voice recognition algorithm. The use of larger words with more complicated pronunciations is for advanced systems where voice recognition elements are investigated more deeply and many more properties extracted. Considering the system was developed based on the content in earlier sections a simple numbers based approach has been taken in order to control the first 4 major axes of the robotic arm. Other commands have been included that share little similarity with the numbers and been used for extra variable control. The figure below has been created to help with the 'command - action' interpretation.

It can be seen that the numbers 1 to 9 translate late to a specific direction in which each axis is to turn and be manipulated. Using opposite numbers to move in alternative directions is the base of the logic here.

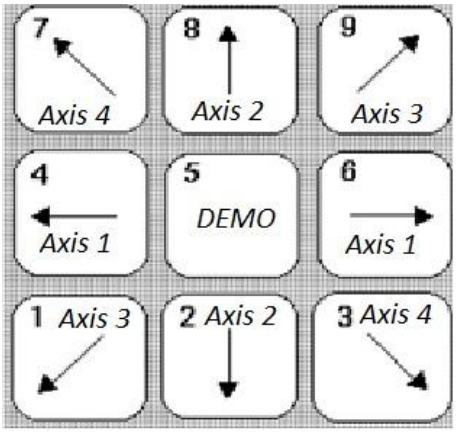


Figure 19 - Command-Action number pad

The commands used to control various movements and variables of the robot can be summarised in the following table:

Command	Sent String	Action
ONE	'one'	Rotate axis 3 forward
TWO	'two'	Rotate axis 2 forward
THREE	'three'	Rotate axis 4 clockwise
FOUR	'four'	Rotate axis 1 anti-clockwise
FIVE	'five'	Perform Axis limit demonstration
SIX	'six'	Rotate axis 1 clockwise
SEVEN	'seven'	Rotate axis 4 anti-clockwise
EIGHT	'eight'	Rotate axis 2 backward
NINE	'nine'	Rotate axis 3 backward
STOP	'stop'	Return robot to initial position
END	'end'	End Command loop and return Robot to initial position
HIGHER	'higher'	Increase arm speed (+100)
LOWER	'lower'	Decrease arm speed (-100)

Table 1 - Command List

The idea of these commands is to give the operator near complete control of where he or she wants the robot to move and in which way. This means that independent movement and control of the 6 axes is essential, requiring individual commands for each axis. It is important to note that this type of control does not make it easier to reach a defined point or target object, it essentially allows for greater user defined control of where each axis needs to be positioned. This can have a number of advantages, which will be explored in later sections.

4.3.2 Building Commands

The process of interpreting a command from the Matlab code and instructing the robot to do something plays a major role in this project. More importantly, creating a system that can show flexibility throughout the 3D space instead of just executing a pre-defined program is of high priority.

In this section the procedures for each of the commands will be investigated and explored.

END

The 'END' command is there to exit the voice recognition routine from any state. After it has been said no more commands will be recognised and accepted at the input and the robot itself will be returned to the home position.

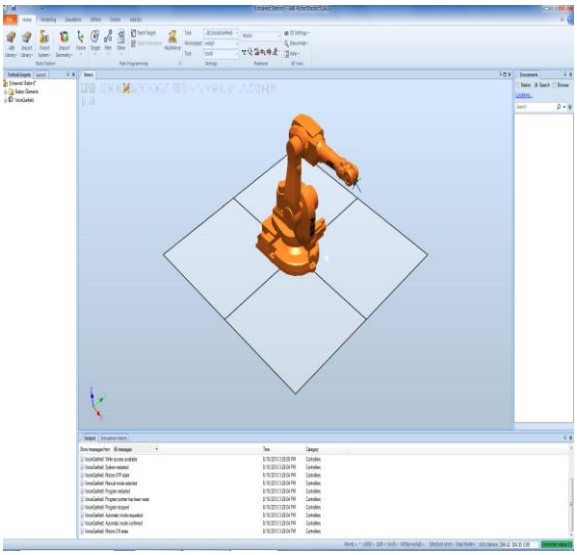


Figure 20 - Initial Robot Position

STOP

The 'STOP' command has a high priority within the system program. Whenever a situation arises where the system recognises this command, the robot is to move from its current location to the initial position where all axes are reset to zero degrees.

ONE & NINE

The commands 'ONE' and 'NINE are associated with axis 3 control. This axis is involved in the vertical movement of the robotic arm and is limited (with reference to the centre of axis 1) to:

- a positive angle (in the forward direction) of 55˚
- a negative angle (in the backward direction) of -235˚

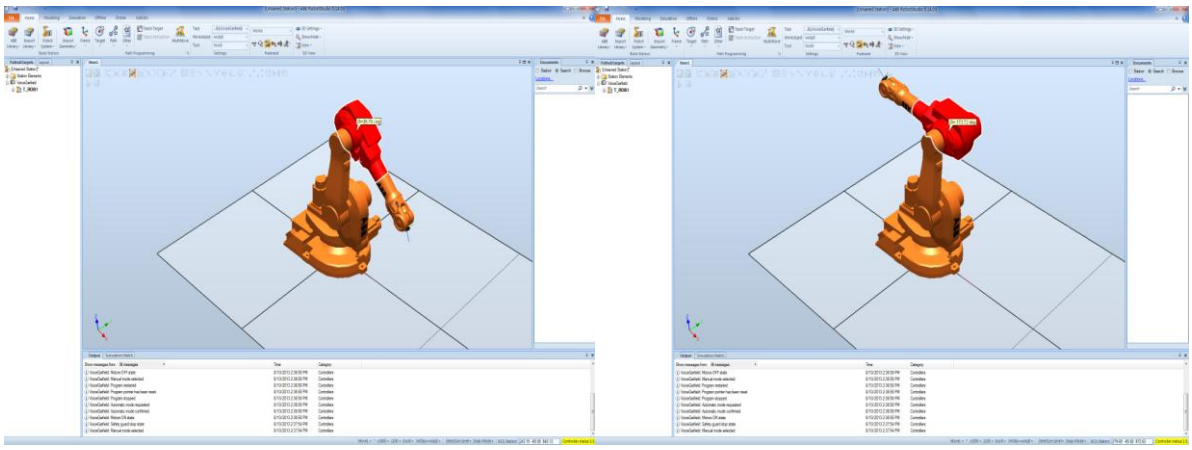


Figure 21 - Axis 3 movement

TWO & EIGHT

The commands 'TWO' and 'EIGHT' are associated with axis 2 control. This axis is centred off the base of the robotic system and controls an element of vertical movement. The limitations of this axis are:

- A positive angle (in a clockwise direction) of 63˚
- A negative angle (in an anti-clockwise direction) of -136˚

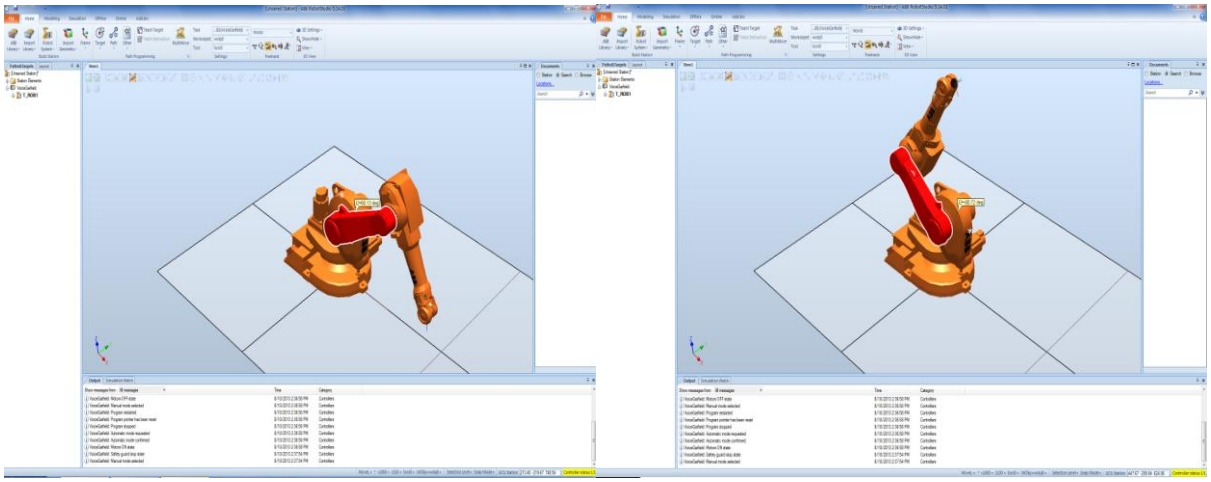


Figure 22 - Axis 2 movement

THREE & SEVEN

The commands 'THREE' and 'SEVEN' are associated with axis 4 control. This axis acts as the pivotal motion for the 'wrist' of the arm where the limitations of the axis are:

- A positive angle (in a clockwise direction) of 200˚
- A negative angle (in an anti-clockwise direction of -200˚

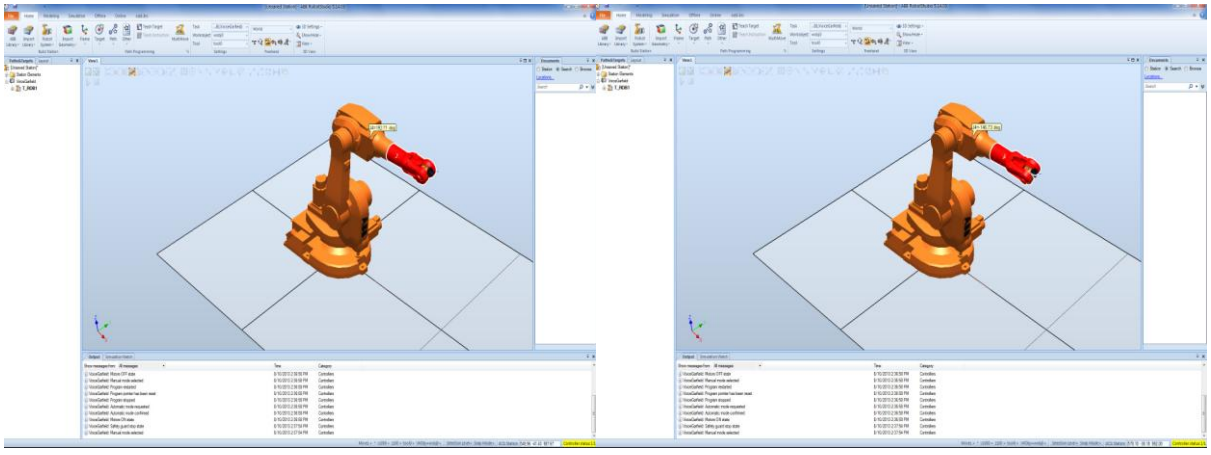


Figure 23 - Axis 4 movement

FOUR & SIX

The commands ‘FOUR’ and ‘SIX’ are associated with axis 1 control. This axis is the base of the robotic system and controls the horizontal movement through a centre pivot. The limitations of this axis are:

- A positive angle (in a clockwise direction) of 180˚
- A negative angle (in an anti-clockwise direction) of -180˚

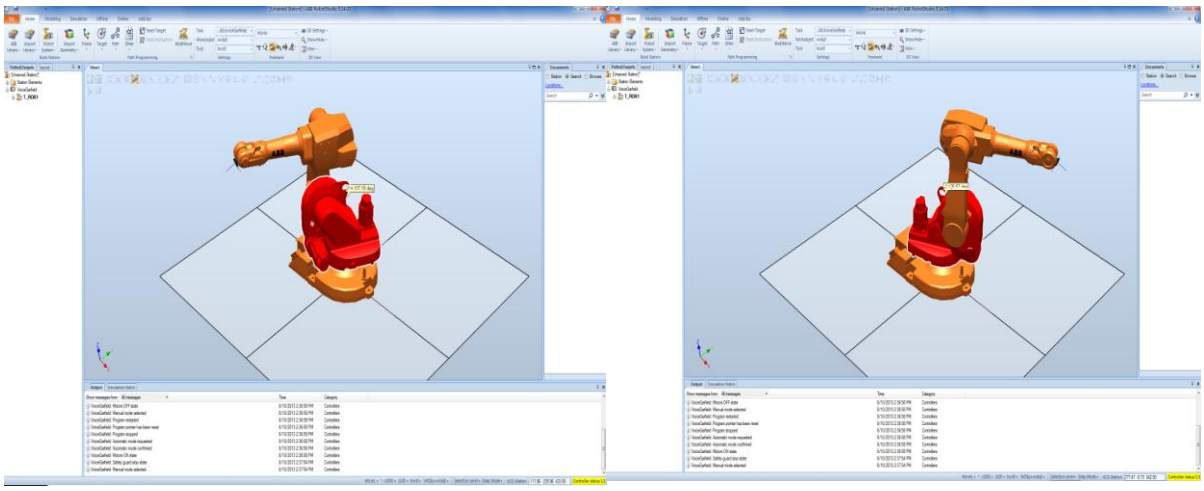


Figure 24 - Axis 1 movement

FIVE

The command ‘FIVE’ executes a demonstration of each of the first 4 axes of Garfield; in particular the limitations in which they can reach. This means, avoiding error warnings caused by the mechanical limits of the robot arm. Each limit is explored without breaching the maximum or minimum value in order of axes (i.e. 1 2 3 4) and then the arm is returned to the initial position ready for the next command.

Once executed, all of the number based commands above are displaced by a set angle that can be defined by the user. This set angle can only be changed once simulation has ended and therefore places limitations on the accuracy and efficiency of the system. Setting the angle quite high results in less commands but reduces the accuracy when trying to move to a fixed point or object. Setting the angle quite low increases this accuracy but decreases efficiency, as it will take more utterances of the same command to reach the same distance as it took one command with the higher angle.

With more time and effort a different strategy could be developed using the same concepts, but for now it is important to find a balance between accuracy and efficiency when using this system.

HIGHER

The 'HIGHER' command is a simple command to increase the speed of the robot by +100 mm/s. In order to do so, the speed variable will not only be incremented when this procedure is called but the current speed checked to see if it's at a maximum. This can be seen in the following code:

```
Speed = 100                          //Initial Speed variable defined at program start
PROC    commandhigher()
IF speed = 1000 THEN
TPwrite "Speed at maximum";          //Command to write information to the Flex pendent
ELSEIF speed <1000 THEN
speed := speed + 100;                //Increment Speed
TPwrite "Speed increased to " \Num:=speed ;
ENDIF
ENDPROC
```

Figure 25 - RAPID code for Higher command

LOWER

The 'LOWER' command is a simple command to decrease the speed of the robot by -100 mm/s. In order to do so, the speed variable will not only be decremented when this procedure is called but the current speed checked to see if it's at a minimum. This can be seen in the following code:

```
Speed = 100                          //Initial Speed variable defined at program start
PROC commandlower()
IF speed = 100 THEN
```

```
TPwrite "Speed at minimum";          //Command to write information to the Flex pendent
ELSEIF speed >100 THEN
speed := speed - 100;                //Decrement Speed
TPwrite "Speed decreased to " \Num:=speed ;
ENDIF
ENDPROC
```

Figure 26 - RAPID code for Lower command

4.3.3 Command Execution

In order for each of the commands defined in the command structure to work, the RobotStudio functions needed to be manipulated in order to achieve appropriate movement. The main function used in this movement process was MoveAbsJ, its structure can be seen below.

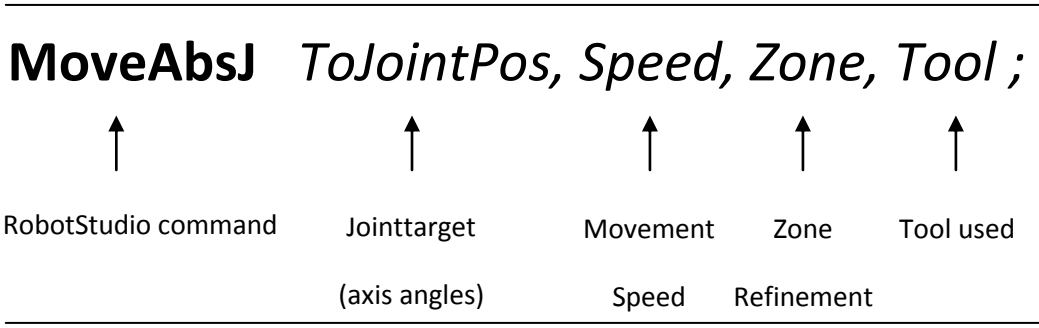


Figure 27 - MoveAbsJ Structure

MoveAbsJ is used to adjust the angles (in degrees) of each of the individual axes without using any form of linear (X, Y, Z) positioning. The angles of each of the 6 axes and any externally connected axes can be manually changed within their pre-defined boundaries or mechanical limitations by editing the following:

```
[[0,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
```

Figure 28 - Jointtarget structure

The above figure shows a form of data known as a *jointtarget*, specifically created to be executed within the MoveAbsJ command. The first 6 digits represent each of the 6 axes of the robotic arm and can be varied using values in degrees. The remaining 6 digits are for any externally connected axes to be used during normal operation of the arm in industry (ie. conveyors, turntables, etc).

'Speed' is a variable in units of *mm/s* that also must be defined each time the command is executed. The speed is being stored as a variable that can be manipulated upon command ('higher' and 'lower') and then be used within the command.

'Zone' refers to the area around a point in space, in which the end point of the tool must reach when moving towards or through it. This allows the use of smoother movements when executing many point to point commands and for our purpose is set to *fine* (i.e. must travel all the way to the point).

The last defined variable is the 'tool' type, where different tools can be defined and called upon. For this project *tool0* is used, where this essentially means there is no tool connected.

4.4 Interfacing and Communication

This section describes the processes involved and steps taken in order for Matlab and RobotStudio to interface and communicate with each other.

Matlab to Matlab

As described in section 2.6 above, TCP/IP socket communication has been implemented across the two applications Matlab and RobotStudio. In order to first understand the messaging process a socket was created between two separate windows of Matlab using the following socket settings.

- Port number = 1234
- Computer IP address = 139.86.166.26 (or 0.0.0.0 if network is not being used)
- Number of retries = 40

A server was firstly setup with the above port number setting, number of retries and a message ready to be sent. In this instance the message is a string of the command (i.e. 'one'). This required executing the *server.m* function that can be found in Appendix D.

A client was then setup up on the alternate window with the same port number and number of retries, as well as the IP address of the server. Once the server was running an executing its retries the client was run using the *client.m* function found in Appendix D. Connection could then be made between the two applications and the message ('one') be written by the server

and read by the client. The following screenshot shows the successful connection in the command window.

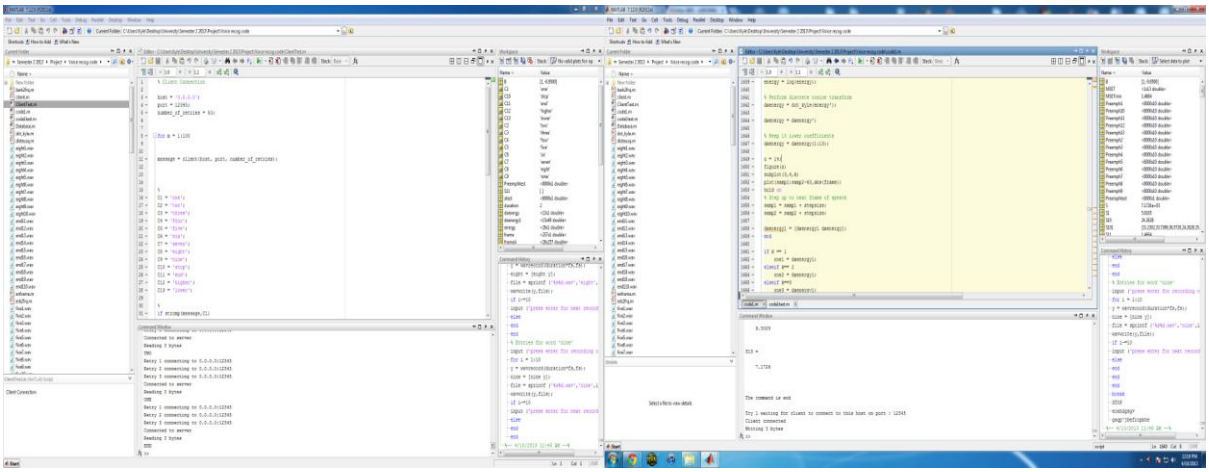


Figure 29 - Socket connection screenshot

Once this process was proven to work it was time to begin integration of this system into RobotStudio.

Matlab to RobotStudio

For the creation of the Matlab side of the communication link, the server configuration properties remained the same, as well as the code used to do so. The main challenge here was to configure the socket properties for RobotStudio in its own RAPID language. In order to set up the client socket the following code was used.

PROC Socketconnect1()

SocketCreate client_socket;

SocketConnect client_socket, "139.86.166.26", 1234 \Time:=1;

ERROR

IF ERRNO = ERR_SOCK_TIMEOUT **THEN**

IF retry_no < 40 **THEN**

WaitTime 1;

retry_no := retry_no + 1;

TPwrite ""\Num:=retry_no;

```
RETRY;
ELSE
RAISE;
ENDIF
ENDIF
ENDPROC
```

Figure 30 - Socket Connection RAPID

This code allowed the RobotStudio program to remain in a continuous loop of trying to connect to the server. If the amount of retries exceeded 40, the program would output a socket connection error and require restarting. This meant that the program could wait up to approximately 40 seconds for an incoming connection from Matlab, which is ideal for the command output control and could easily be extended by changing a couple of variables.

Once connection had been made with the server the following code was executed in order to read the incoming message.

```
SocketReceive client_socket \Str := message;
SocketClose client_socket;
retry_no := 0;
```

Figure 31 - Socket Receive RAPID

At this point, the socket is closed and the message received is processed to determine which action corresponds to the transferred command. The action is then performed and the RAPID program re-enters the loop waiting for connection from the server.

The following screenshot shows the two programs interacting through the socket communication process:

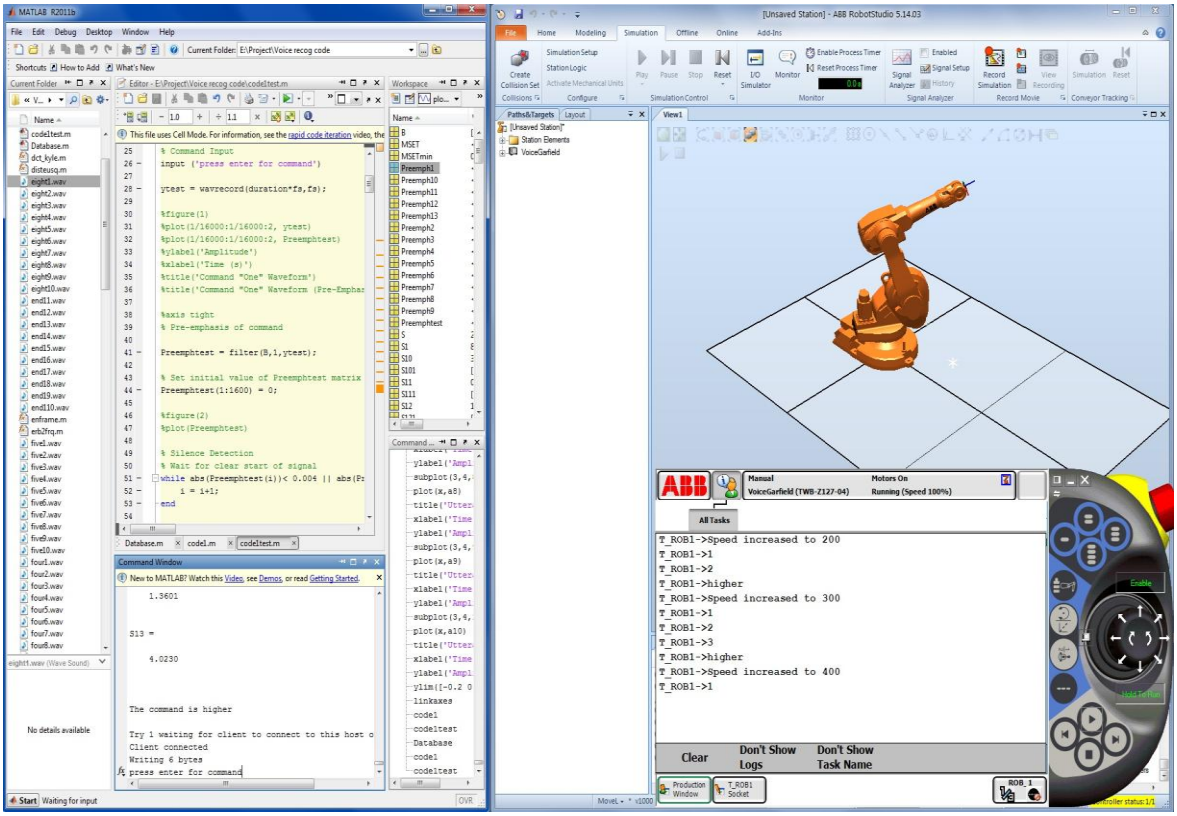


Figure 32 - Interaction between Matlab and RobotStudio

As can be seen on the left hand side, Matlab has processed a command, recognised it and output it through the server socket. On the Flex Pendant screen we can see the recognition of the command 'higher' and the value to which the speed has been increased to due to this command. The numbers 1, 2 & 3 show the number of retries the RAPID program went through before finding that the server was ready for connection.

4.5 Chapter Summary

This chapter has provided detailed information on the design, development and implementation of the voice recognition and control system. It has explored; the signal processing techniques used to extract vocal features, the communication technique to be used between the software platforms and the coding techniques used for interpreting outputs and executing robotic movement. It has also provided the command structure to be used for testing and evaluation in Chapter 5.

Chapter 5

Performance and Evaluation

5.1 Chapter Overview

This chapter aims to explore the performance of the voice recognition and control system throughout the testing and simulation stages. The results of these tests and simulations are to be evaluated and further work and improvement considered. Any improvements implemented onto the system will be discussed and appropriate tests conducted to demonstrate increased performance.

5.2 Performance Methodology

Evaluating the performance of the system under specific conditions is designed to demonstrate both; the suitability of the proposed techniques used and how well the system would respond to different environments. The evaluation consisted of a number of components, these being:

- Testing Error Rates of both the original system and the modified system. This would be done under 3 conditions (No noise, background noise, random noise).
- Testing the usability of the commands with the robot arm between the initial position and a fixed point. (A time based test with using various angle increments)
- Testing response time of the system to the command given (real time operation)
- Qualitative Analysis of operation

Essentially, these tests will aim to provide an insight into how functional the algorithm is and where it can be improved, as well as the possible real world applications of this sort of control method.

5.3 Error rate Performance

5.3.1 Methodology

Assessing the error rate of the recognition algorithm will give a crude approximation of the reliability of the system with regards to correctly processing an individual command. In the early stages of building the algorithm, small testing was conducted using only a few commands to try and grasp an early result or breakthrough. The original system was tested using 10 utterances of each command and the results recorded. A few minor changes were made and some different processes used to create a modified system with a much higher success rate.

Once the modified system was built up further to accommodate more commands and some more minor adjustments made, 10 utterances of each command were input into the algorithm under the following 3 conditions:

- Little to no background noise (closed off room)
- Background noise (music)
- Background noise (fan)

Each of these conditions displays 3 typical situations involving changes in ambient noise, whether it is in a laboratory or a workshop. In order for the performance test to be deemed reliable each command could not be said more than twice in a row, as to prevent repeating at the same pitch and tone for all 10 utterances of each command. This somewhat varies the properties of the utterance making it more difficult for the algorithm to recognise and in turn, testing its limitations.

Before testing the algorithm the user must first train the algorithm ready for recognition. This process can take a couple of minutes.

Ideally the error rate would be expected to be quite low for any advanced systems where all or almost all utterances were recognised. Expectations for this particular algorithm are however quite low and recognition failures are expected.

5.3.2 Measurement and Testing

The following tables display the results of those error rate tests describe above.

Original System

Command	No. Times correct	No. Times incorrect	Error Rate = No. Errors/ No.Utterances *100%	
'One'	6	4	40%	
'Two'	5	5	50%	
'Three'	3	7	70%	**Total Error Rate**
'Stop'	7	3	30%	47.5%

Table 2 – Original System Recognition Error rates (No noise)

Command	No. Times correct	No. Times incorrect	Error Rate = No. Errors/ No.Utterances *100%	
'One'	3	7	70%	
'Two'	4	6	60%	
'Three'	2	8	80%	**Total Error Rate**
'Stop'	2	8	80%	72.5%

Table 3 - Original System Recognition Error rates (Background music)

Command	No. Times correct	No. Times incorrect	Error Rate = No. Errors/ No.Utterances *100%	
'One'	3	7	70%	
'Two'	3	7	70%	
'Three'	3	7	70%	**Total Error Rate**
'Stop'	2	8	80%	72.5%

Table 4 - Original System Recognition Error rates (Background Fan)

Modified System

It should be noted that the Error Rate is based on the number of times the algorithm incorrectly identified a command.

Command	No. Times correct	No. Times incorrect	No. Times rejected	Error Rate = No. Errors/ No.Utterances *100%	
'One'	9	0	1	0%	
'Two'	9	0	1	0%	
'Three'	8	1	1	10%	**Total Error Rate**
'Stop'	10	0	0	0%	2.5%

Table 5 - Modified System Recognition Error rates (No noise)

Command	No. Times correct	No. Times incorrect	No. Times rejected	Error Rate = No. Errors/ No.Utterances *100%	
'One'	0	0	10	0%	
'Two'	1	0	9	0%	
'Three'	0	0	10	0%	**Total Error**
'Stop'	0	0	10	0%	0%

Table 6 - Modified System Recognition Error rates (Background music)

Command	No. Times correct	No. Times incorrect	No. Times rejected	Error Rate = No. Errors/ No.Utterances *100%	
'One'	1	0	9	0%	
'Two'	2	0	8	0%	
'Three'	3	0	7	0%	**Total Error Rate**
'Stop'	0	0	10	0%	0%

Table 7 - Modified System Recognition Error rates (Background Fan)

Once testing had been completed on the modified system and reasonable results were obtained, the algorithm was expanded to increase the vocabulary size to house the full range of commands. This was then tested as seen in the following tables.

Complete Modified System

Command	No. Times correct	No. Times incorrect	No. Times rejected	Error Rate = No. Errors/ No.Utterances *100%	
'One'	9	0	1	0%	
'Two'	10	0	0	0%	
'Three'	10	0	0	0%	
'Four'	8	2	0	20%	
'Five'	9	1	0	10%	
'Six'	9	0	1	0%	
'Seven'	10	0	0	0%	
'Eight'	10	0	0	0%	
'Nine'	10	0	0	0%	
'Stop'	10	0	0	0%	
'End'	10	0	0	0%	
'Lower'	8	1	1	10%	**Total Error**
'Higher'	10	0	0	0%	3.08%

Table 8 - Complete Modified System Recognition Error rates (No noise)

Command	No. Times correct	No. Times incorrect	No. Times rejected	Error Rate = No. Errors/ No.Utterances *100%	
'One'	0	0	10	0%	
'Two'	1	0	9	0%	
'Three'	0	0	10	0%	
'Four'	1	0	9	0%	
'Five'	0	0	10	0%	
'Six'	0	0	10	0%	
'Seven'	0	0	10	0%	
'Eight'	0	0	10	0%	
'Nine'	0	0	10	0%	
'Stop'	0	0	10	0%	
'End'	0	0	10	0%	
'Lower'	0	0	10	0%	**Total Error**
'Higher'	0	0	10	0%	0%

Table 9 - Complete Modified System Recognition Error rates (Background music)

Command	No. Times correct	No. Times incorrect	No. Times rejected	Error Rate = No. Errors/ No.Utterances *100%	
'One'	2	0	8	0%	
'Two'	1	0	9	0%	
'Three'	1	0	9	0%	
'Four'	0	0	10	0%	
'Five'	0	0	10	0%	
'Six'	1	0	9	0%	
'Seven'	0	0	10	0%	
'Eight'	0	0	10	0%	
'Nine'	0	0	10	0%	
'Stop'	2	0	8	0%	
'End'	0	0	10	0%	
'Lower'	0	1	9	10%	**Total Error**
'Higher'	0	0	10	0%	0.77%

Table 10 - Complete Modified System Recognition Error rates (Background Fan)

5.3.3 Discussion of Results

The results contained in Table 2 show a clear problem with the processes used for the algorithm and clear need for improvement and restructuring. At this current state, the algorithm would not be capable of performing at a suitable level for commanding a robotic arm. An Error Rate of 47.5% is far too high, when it comes to achieving the desired result and the completion of the project objectives.

Further testing conducted under the noisy conditions as seen in Table 3 and Table 4, only re-enforced the belief that more needed to be done to the algorithm in order to be ready for socket communication with RobotStudio. The higher error rates around 72.5%, give the impression that the algorithm is also being affected by the different background noise. Considering the high probability of errors, the data produced under noisy conditions could be in some way correlated to that of the no noise data and be caused by other factors within the code, not just noise.

In stepping forward and improving the original system, a rejection element was included to prevent any unwanted commands sneaking through the system or any commands being confused with other commands. This rejection element involved setting a pre-defined limit for the dissimilarity value, any value beyond this limit would not be considered and output an error. It was quite effective when similarities were low, but as the vocabulary expanded and similar utterances were added, some commands snuck through.

It was also found that the original system had not been calculating and comparing the utterances in the correct manor. Simple syntax errors and some arithmetic mistakes were not in fact calculating the 13 Mel Cepstral Coefficients needed for the voice recognition process. A considerable amount of time was spent reviewing this stage of the project to ensure that the right information was being extracted and processed. Once it had been reviewed and modified it produced the results shown above in Tables 5, 6 and 7.

A clear sign of improvement was evident during the next batch of testing. Not only did the error rates drop to 2.5% but the commands would also be rejected if there similarity was not close enough. In saying this, the overall effectiveness of the algorithm increased dramatically and could now be deemed reliable enough to execute voice based commands under no noise conditions.

The results obtained in table Tables 6 and 7, showed the effectiveness of the rejection element added to the algorithm. Any incoming command showing too much dissimilarity compared to the command models recorded under no-noise conditions was in fact rejected almost 100% of the time. This result is good for the safety of the algorithm when put into practice but not ideal for any real world situation where industrial noise will always be present. In this case, one proposed solution to combating this situation is to record those command models under noisy conditions and relax the silence detection threshold during processing. This would still obtain those important characteristics of speech held by each utterance, but with heavier background noise effects. Error rates would be expected to drop if the noise is variable but could remain the same if it was continuous and processes correctly.

Now that a stable algorithm had been developed for the four commands further commands could be added to complete the command structure. The results shown in Table 8 display the error rate percentages of the completed algorithm under no noise conditions. A total error rate of 3.08% was recorded, indicating that there was not too much change after the addition of more commands to the vocabulary, but the chance of mistaking one command for another was slightly increased.

It is important to add, that during the course of testing, it was found that any extra effort made when uttering a command or any prolonged the utterance of the word, would cause an error or in some cases, the utterance to be confused with another. Successful tests were achieved when the utterance remained within a certain similarity boundary of those that were recorded as reference. Making sure some vocal variation was made for the same utterance when recording the models became an important task, as this expanded the similarity boundaries of each of the utterances. Thus, making the algorithm more effective and reliable for a range of utterances.

The introduction of background noise with the input utterances during the testing stage brought forward the results seen in Table 9 and 10. These showed once again the power of the rejection element and its effectiveness against dissimilarity. An extra test was performed (as seen in the table in Appendix C) to show the results without this element. A number of commands were found to be confused with each other under the presence of Background noise, re-enforcing the importance of this element within the algorithm.

5.4 Command Usability (Quantitative)

Using the voice as a command tool has its limitations, especially when the commands are restricted to a single utterance. It not only limits the number of axes that can be controlled but the number of variables in general that can be changed. The current system as described in section 4.3, allows the movement of a single axis, in a single direction per command. The distance in which this is carried out is dependent upon the desired accuracy of the system and the specific uses for the voice control. It is important that the relationship between time and accuracy be investigated to see which angle increment should be used.

5.4.1 Methodology

The main focus of this testing was to find out how much time it would take for the individual axes of the robotic arm to move at a set speed after receiving the command input. This time would vary depending on the specific angles distances used and comparing these times would allow for appropriate evaluation.

The angle increments used in the testing process were:

- 10 degrees
- 15 degrees
- 30 degrees

In order to conduct the following testing a timing mechanism must be set up in order to record the elapsed time between when the Robotic arm receives the command and when it finishes executing the command. This was done by inserting various parts of the following code into the appropriate start and end places of the command movements.

```
VAR clock clock2;          //Declare clock variable     ClkReset clock2;
//Reset clock

VAR num time;              //Declare time variable      ClkStart clock2;
//Start clock


ClkStop clock2;            //Stop clock

time := ClkRead(clock2);   //Read clock time

TPWrite ""\Num := time;    //Write time to flex pendent
```

Figure 33 - Clock initialisation and control

Each axis movement was then tested and the time recorded for each of the angles stated above. Once this first stage of testing is complete two points will be selected and navigated towards using the command system. This will test the functionality of the robot arm and how this application can apply to real world situations. It will be expected that the smaller angles will take less time to execute and create better accuracy when it comes to moving towards a designated point, but will overall take more time considering the amount of times the command must be executed.

5.4.2 Measurement and Testing

The following results were obtained from each angle increment at a set speed of 300 mm/s. Screenshots of this testing can be found in Appendix E.

Angle = 10˚

Test Number	Command 'one' time	Command 'two' time	Command 'three' time	Command 'four' time
1	0.584	0.600	0.448	0.604
2	0.584	0.604	0.446	0.608
3	0.584	0.600	0.464	0.604
4	0.588	0.600	0.448	0.604
5	0.584	0.604	0.464	0.602
Average	**≈0.585**	**≈0.602**	**≈0.454**	**≈0.604**

Table 11 - Timing for Angle increment 10 degrees

Angle = 15˚

Test Number	Command 'one' time	Command 'two' time	Command 'three' time	Command 'four' time
1	0.844	0.988	0.644	0.988
2	0.832	0.984	0.644	0.988
3	0.848	0.984	0.644	0.976
4	0.844	0.988	0.638	0.992
5	0.832	0.984	0.644	0.988
Average	**≈0.840**	**≈0.986**	**≈0.643**	**≈0.986**

Table 12 - Timing for Angle increment 15 degrees

Angle = 30˚

Test Number	Command 'one' time	Command 'two' time	Command 'three' time	Command 'four' time
1	1.588	1.876	1.156	1.852
2	1.596	1.872	1.172	1.860
3	1.576	1.876	1.176	1.852
4	1.588	1.868	1.164	1.872
5	1.576	1.872	1.172	1.872
Average	**≈1.585**	**≈1.873**	**≈1.168**	**≈1.862**

Table 13 - Timing for Angle increment 30 degrees

The robot arm can now be tested to find the amount of time it takes to reach a designated point in space using the voice controlled algorithm at the different angle increment tested above.

The point is defined as follows (in *jointtarget* form):

[90, 60, 60, 90, 0, 0]

If we consider the fastest possible time without a break, between when the process is finished and when the next command is ready, we can assume that:

- the next command is already recognised and awaiting socket connection from RobotStudio
- The time taken to make connection is less than approximately 0.1 s
- The time will equal (the number of commands required to achieve the appropriate point angle) * (Average time to execute command) for each of the axes
- We start from the initial point [0,0,0,0,0,0]
- Commands: one & nine, two & eight, three & seven, four & six will take the same amount of time to process as they are only moving in the opposite direction.

Based on the data extracted from the tables above, the following approximations were calculated and presented in the following table.

Point to point calculations

Test	Axis 1 Command 'four'	Axis 2 Command 'two'	Axis 3 Command 'one'	Axis 4 Command 'three'	Number of Commands said	Total Time (seconds)
Time at 10˚ increment	90˚ = 9 *10˚ = 9* 0.604 **= 5.436**	60˚ = 6*10˚ = 6* 0.602 **= 3.612**	60˚ = 6*10˚ = 6*0.585 **= 3.510**	90˚ = 9*10˚ = 9* 0.454 **= 4.086**	9+6+6+9 = 30 30*0.1 **= 3.000**	**= 19.644**
Time at 15˚ increment	90˚ = 6 *15˚ = 6* 0.986 **= 5.916**	60˚ = 4*15˚ = 4* 0.986 **= 3.944**	60˚ = 4*15˚ = 4*0.840 **= 3.360**	90˚ = 6*15˚ = 6* 0.643 **= 3.858**	6+4+4+6 = 20 20*0.1 **= 2.000**	**= 19.078**
Time at 30˚ increment	90˚ = 3 *30˚ = 3* 1.862 **= 5.586**	60˚ = 2*30˚ = 2* 1.873 **= 3.746**	60˚ = 2*30˚ = 2*1.585 **= 3.170**	90˚ = 3*30˚ = 3* 1.168 **= 3.504**	3+2+2+3 = 10 10*0.1 **= 1.000**	**= 17.006**

Table 14 - Point to Point time approximation

These approximations have been calculated assuming the processing time to make a connection is relatively low. This relatively low time is only possible when the command has been recorded, processed and output to RobotStudio before the previous command had finished executing.

5.4.3 Discussion of Results

It can be seen in the tables above that each increment used in the testing process varies in execution time. Obviously, without looking at the results, the further the arm has to travel, the longer it is going to take to finish the a designated movement. Looking at these times and applying it to a real situation, where say the user needs to move the arm towards a pre-defined point using his/her voice, the information becomes a lot more useful. This process can be seen in table 14 above where the different angle increments were applied to an actual movement towards a point and approximate completion times recorded.

From the data approximated, the largest angle (30˚) has proven to take the least time to get to the designated point, with the smallest angle (10˚) taking the most time. This was to be expected considering the smaller angle had to be processed more often than the larger angle. If the conditions changed to a more lifelike circumstance where commands take over 2 seconds to input before they are processed ready for output to RobotStudio the timing gap between the smaller angle and the larger angle would be much larger. In the long run, time is saved by increasing the angle increment, but in doing so it reduces the accuracy of the system and at 30˚ only allows 6 movement steps in either direction for axis 1. Reducing the increment to 10˚ increases the accuracy, allowing 18 steps in either direction but increases the time it takes to get to pre-defined points. In observing both of these results, operation would be more successful at lower angle increment values, especially when it comes to gaining greater control of the robotic arm. Although, depending on the operation it is being used for, the large increment could suit better i.e. for a Robotic Functionality Demonstration.

In comparison to current methods of jogging, like using the flex pendent, the voice control technique used here is essentially a step back. In saying this, taking this concept and dedicating more time and money towards it, a system which can respond in real time and following command effectively would exceed the current technologies.

It is possible to change the approach of this movement, making a command move an axis in the direction perceived until the stop command is spoken. In terms of functionality, this would be a lot more efficient and reduce the amount of commands required to move the same distance. Although, this approach would entail reducing the processing time and increasing the response time of the system to deal with real time control. Difficulties with such a system would be high but it is a possibility for future voice controlled systems.

5.5 Real time algorithm and communication response

An important factor involved in creating a system like this, is its response to command inputs in real time. Without the proper response times, operation becomes difficult especially with continual movement operation. The real time response of the algorithm and its communication with RobotStudio will be investigated and approximated below.

5.5.1 Response Methodology and Results

In order to obtain an effective approximation of the time taken to:

- Input the command
- Process the command
- Recognise the command
- Output and Communicate the command

We must consider each operation individually and determine an appropriate duration for each.

Input

The time allocated towards the input of a command can be defined by the user with the Matlab algorithm. Currently this value is set at 2 seconds after the execution of the 'enter' key to ensure the entirety of the utterance is captured. As mentioned in section 4.2.1 the utterance of a word generally last between 200 and 800 milliseconds, meaning that the input time could be reduced by up to a second or more. If this was the case, user response to the 'enter' key execution would have to be almost instantaneous and any delay, could happen to miss recording the actual utterance of a command.

Processing

The processing time involved in extracting the Mel Frequency Cepstral Coefficients can be obtained using the clock functions contained in Matlab. These functions can be set at the start and end of the processing to obtain execution times and are represented in the code below.

```
% Start timing clock
ticID = tic;

*Processing of Utterance*

% Stop timing clock
elapsedTime = toc(ticID);
```

Figure 34 - Clock timing code

The variable 'elapsedTime' contains the current value for time between stopping and starting the clock. The processing time test for a random input was completed a number of times below.

Test Number	Time (s)	
1	0.0456	
2	0.0523	
3	0.0521	**Average Time (s)**
4	0.0486	0.0497

Table 15 – Processing Time of Utterance

Recognition

The next part of the algorithm involves testing the input utterance against a number of different command models using the Euclidean distance calculations. The difficulty in timing this particular section is that, each model is tested in the command order specified in section 4.3.1, meaning that the recognition time of command 'one' will be partially faster than the recognition time of the command 'lower'. The times for both of these commands were recorded in order to determine a best and worst case scenario for the recognition timing.

In order to test these recognition times, the same clock functions above were employed.

Test Number	Time (s)		
	'one'	'lower'	
1	0.1161	0.1197	
2	0.1181	0.1187	
3	0.1180	0.1197	**Average Times (s)**
4	0.1179	0.1186	Min ('one')= 0.1175 Max ('lower') = 0.1192

Table 16 - Recognition Time of utterances

Output and Communication

Once the command had been recognised, it was time to output through the socket for RobotStudio to read and execute the command process. This process was found to be quite difficult to time considering that the client connection from RobotStudio was continually retrying to connect to the server socket created by Matlab. Every connection would occur at a different time, depending on when the server was ready for connection. Worst case scenario meant that the client would retry just before the server was ready and have to wait almost 1 second before reconnection to ensure enough time had passed to close connection before retrying.

Results from testing the connection speeds could also be obtain using the clock function in Matlab. The clock was started before connection and stopped once connection had been successful. The results can be seen in the following table.

Test Number	Time (s)	
1	0.1488	
2	0.0530	
3	0.2256	**Average Time (s)**
4	0.9218	0.3373

Table 17 - Connection Time of utterances

Timing Approximation

Each of the timing aspects explored above can now be merged together for an appropriate real time estimate of the systems response to a command.

Total = input + processing + (worst case) recognition + (worst case) connection

Total = 2 +0.0497 + 0.1192 + 0.9218

Total = 3.091 seconds (worst case) system response time

5.5.2 Discussion of Results

It can be clearly seen that the worst case scenario of approximately 3 seconds before connection is made to the RobotStudio Software can be improved. Considering an utterance only takes a matter of hundreds of milliseconds to occur, time can be majorly reduced in obtaining the sample input. A system that can continually record and buffer information, whilst searching for commands could be employed to do so and save up to a second or more on this response time. Improvement made to processing techniques can also be of minimal benefit to the system and help increase its performance. Synchronising the connections perfectly between the 2 platforms to exchange information could also be a major step in reducing the time taken to get the command and execute its relevant procedure. Employing these ideas could reduce the response time to under a second, which would drastically improve the functionality of the system and its potential for future use.

With regards to use in a Real Time system, this 3 second delay is not desirable and would require major improvement as stated in the above paragraph. At delays of this magnitude, the probability of damaging equipment or even user injury is much greater due to late reaction times. If speeds of up to 1000mm/s are being used then the robot has already travelled 3 metres before it considers the response. Luckily the movement involved with the current system is incremented and not continuous preventing such an occurrence.

For the purposes of this project, the approximated response time is satisfactory to perform the demonstrations required and meet the project objectives. If a continuous movement scheme was implemented here, then a faster response time would be necessary to allow the appropriate execution of the command ‘stop’, so the user could tell the arm to stop in a desired position.

5.6 Qualitative Analysis of Results

It is essential to the evaluation of the system that the results are looked at in a qualitative manor. This allows an overall grasp of the concept of the project and its benefits to the user. Such things like, ease of use, industrial applications and constraints will all be explored.

5.6.1 Ease of Use

An important part of the Voice Recognition and Control system is its ability to provide easier interaction between the user and the Robotic arm. Without this special form of wireless interaction, the system would be just another basic form of control. The fact that the flex pendent would no longer be required, making the operation essentially 'hands free', creates a sense of freedom for the user and allows he/she to concentrate on the task at hand, instead of fiddling around with code and buttons. In observing this system and considering its usability, a number of factors need to be considered, these are:

- Learnability – ability of user to operate the system upon first encounter
- Efficiency – ability of the user to learn from the system and perform tasks quicker over time
- Memorability – involves the ability of the user to leave the system for some time, return to it and operate it (i.e. not forget commands and constraints)
- Errors – User errors and how severe they can be
- Satisfaction – Overall ease of use and enjoyment obtained from the system

These factors will now be explored and discussed, based on observations obtained throughout the design, implementation and testing stages of the project and the results found above.

Learnability

First impressions of the idea of a voice control system, revolve around the system listening to your every command and operating based on these commands. The difficulty in creating a system with such a large vocabulary and multiple user interaction is that it requires a lot of time, money and resources. In conducting this project my aim was to develop a basic voice recognition platform that accepted a small range of commands built for interaction with the robot axes. In doing so, I could focus on improving interaction rather than developing a better recognition model. As a first time user of such a system (once it had been completed), I found it quite easy to operate and adjust the code to obtain movement in the robotic simulation.

Without considering the setup of the simulation, Matlab scripts and communication between them, a first time user would find it a little difficult to understand. The key steps in using the system could be defined as follows:

- Run the Database script to record the reference utterances for recognition (follow the command window prompts for recording).
- Run the code script to process the recorded utterances and create reference models (this step also produces graphs to examine the recordings and make sure no errors are present).
- Run the codetest script to begin the voice recognition stage of the system. After each 'enter' keypress the user will have 2 seconds to record the command utterance.

Knowing the processes involved with voice recognition and the functions used within the Matlab scripts, a proper understanding of the systems functionality can be obtained. As long as these simple steps are followed and attention is paid to command window outputs the first time usability of the system should be satisfactory.

Efficiency

As stated above, a proper understanding of the system as a whole will be a valuable contribution when it comes to improving the efficiency of the system. The user will learn over time the "in's and out's" of the system and discover how to adjust it to suit specific operations. From a user point of view, knowing the system allows much more efficient execution of commands and completion of tasks. The slightly lengthy response times on the simulator can be exploited and the next command is ready for output to the system as needed for higher efficiency and better completion times. The ability of the user to edit the angle increments explored in section 5.4 will allow for improvement in efficiency when performing various tasks, whether it be moving the axes for servicing or trying to pick up an object near a specific point. Overall the learning efficiency aspect of the system is quite high, considering the documentation provided in this report and the comments provided throughout the Matlab and RobotStudio scripts.

Memorability

An important property of this system and its functions is its ability to remain in the user's memory over time and be used at a later date. In comparison to other basic and boring methods of control, voice recognition appeals to all (as seen in the literature review). Its popularity in recent years has given it an edge over other methods of control, essentially increasing the probability that it processes will be remembered over time. The simplicity involved in the command process of the system also increases this probability and allows the user to pick up on its function, even after a long period of time. It is often those things that

are more interesting and technologically 'new' that people take the time to learn and perfect and in this case, robot voice recognition and control is one of these.

Errors

Throughout the design of this system, errors were a daily occurrence. Some took a matter of hours to fix and others a matter of minutes. Observing the current system, errors are not experienced as much as they had been previously. The most common involves the rejection of a command due to breaches in dissimilarity between the tested command and the command models. In fixing this error, the utterance needs to be repeated once again to initialise movement. Continued rejection errors could mean any of the following situations:

- Rejection limit is too low (back noise is high)
- Command models have not been recorded correctly
- Test Utterance has not been recorded correctly
- The user is not the pre-programmed user (database must be recorded again)

These errors are generally fixed quite easily by; re-entering the command, recording the models properly or focusing on proper pronunciations of the commands.

Another error experienced by the system for first time users could be timeout errors or end of retries involved in the communication between the 2 software platforms. The values of these are set by the user to around about 40 seconds each, if connection is not made within this time or a command not said for this amount of time, an error will be created and require the system to be re-initialised. Avoiding this can be done by setting the amount of retries to a high value and giving the user more time to perform mid-command operation.

At times, the system has the tendency to mistake commands for other commands, depending on how the utterance is said. If clear pronunciation is not used, the system can find it difficult recognise commands and achieve the recognition percentages found in section 5.3. Ensuring commands a clear and consistent, is a big part to the success of the voice recognition side of the algorithm.

Satisfaction

As the designer and user of this system, I obtain a great deal of satisfaction out of operating it and seeing my work come together in an integrated system. Once again focusing on the popularity of the voice recognition technology, any user operating this system would obtain a degree of satisfaction from it. Assuming some background information is known about this particular system and the errors are kept to a minimum any operation would bring a high degree of satisfaction. Considering its application to a number of different tasks, this satisfaction could vary, as there are a number of current systems on the market that are more

reliable and efficient than this particular system. A number of improvements discussed throughout chapter 5 would need to be implemented in order for this system to be competitive.

Overall the usability of the voice recognition and control system is satisfactory for simple movement activity. Its potential for success in industrial applications is moderate considering the direction this technology is taking. With greater improvements in recognition rates, noise rejection and usability it is on track to develop into a massive technology of the future.

5.6.2 Industrial Application

The testing done on this system has been based on the movement of an industrial 6 axis robot arm. As a whole, the system has proven to be effective in controlling the individual axes of the arm and given it a form of wireless control in order to make it 'smarter'. The question now becomes, what can this be used for?

The act of individually moving each axis independently is a form known as jogging. This jogging mechanism is used throughout the robotic arm industry by the user to move the arm to an estimated position to get an idea of the movement. It is also used to run through programs step by step as part of the debugging process, ensuring the movement are correct. It is clear that the method applied in this project relates to this step by step movement and could be implemented by companies as an alternate control form. Instead of using the remote flex pendant, jogging could be done using vocal commands and leave your hands free to adjust objects or use tools during the process. Currently the algorithm developed in this project would not suffice for this kind of operation due to its tendency to produce errors and occasionally mix up commands. Improvements would need to be made and the vocabulary expanded to reach this sort of application, but as it can be seen, the potential is there.

As done before by Pires (Pires 2005), voice recognition of this sort can also be used to just execute a welding process instead of individual axis control. This is a safer method, as it limits the probability of errors and can obtain higher accuracies, but is also limited to what it can do, as all movement must be pre-defined.

Improvements in the response times obtained in 5.4 and 5.5 would also need to be improved for real time control in an industrial application. Without these improvements the safety factors would remain low and deem the system unsafe.

5.6.3 Constraints

A number of factors have limited the capabilities of the voice recognition and control system developed in this project. These factors involve:

- Vocabulary size – Only 13 commands are available to use. Adding more increases the chances of recognising the wrong command.
- Recognition Process – The use of only 13 MFCC out of the possible 39 has restricted the reliability of recognition.
- Incremental Axis movement – has reduced accuracies and point to point execution times.
- Time – Designing, Implementing and Testing this system over 6 months.
- Resources – limited computational power, subject and programming knowledge.

These have all affected the overall design, implementation, testing and performance of the developed system, along with many other smaller factors. With this in mind we should understand that errors WILL occur during normal operation and steps outlined 5.6.1 should be adhered to carefully. The user will require a considerable amount of time to understand the subject matter and processes undertaken within the Matlab scripts before they learn to operate it to its potential.

It is important to state that this project was built from scratch, taking code excerpts from the VoiceBox Toolbox where I could and implemented onto the RobotStudio simulation software. Unfortunately not enough time was there to test the algorithm on Garfield itself but will be recommended for future investigation.

5.7 Chapter Summary

Chapter 5 has explored a number of testing procedures to investigate the performance of the system and judge its viability as a method of control. Error rates and Response times obtained throughout the testing were deemed appropriate and ready for implementation onto Garfield, with improvements to be explored in future research. A qualitative analysis has been completed, exploring the overall usability, its application to industry and constraints of the system.

Chapter 6

Conclusions & Further work

6.1 Chapter Overview

This chapter outlines the final result of this project, as well as recommendations for further improvement and research beyond the scope of this project.

6.2 Conclusions

The entire design process and objectives stated in earlier chapters has been executed in this project. A voice recognition and control system has been developed to make Garfield (the 6 axis robotic arm) smarter. Simulation testing has proven to show the functionality of the system and evaluate its performance as a whole, which has been deemed satisfactory.

The exploration of relevant literature relating to voice recognition and its processes has been conducted and reported with Chapter 2 of this report. Continued research was conducted and reported throughout the many stages of the project as new strategies were developed and implemented. This literature was found through access to professional databases and technical reference manuals relevant to the project. This allowed for an appropriate analysis of any past work, current technologies, techniques and relevant processes to the voice recognition field. It was found that the best way in which to develop such a system was to use the Mel Frequency Cepstral Coefficients (MFCC) contained in each utterance and compare them to a set of reference utterances. Comparison of these was done through Euclidean Distance measures and a relative error was obtained in order to distinguish the best match. It was found that TCP/IP socket messaging was an appropriate method of communication between the 2 applications (Matlab and RobotStudio) and would provide the essential link for command handling.

Following on from chapter 2 an appropriate methodology was implemented in order to address the project specifications and complete the project objectives. A risk assessment was conducted in order to address the safety issues involved with such a project and a research timeline developed to ensure the deadline was met. The design process based on relevant literature was implemented in chapter 4 using signal processing techniques and communication theory, then tested and evaluated in chapter 5.

The outcomes of the performance evaluation indicated that the recognition algorithm operated best under conditions involving no background noise. Any noise present caused high rejection rates and little command recognition. It showed that the rejection element was an important contributor to the success of the algorithm and prevented most unknown or noisy utterances from being recognised as proper commands.

The results involving the evaluation of command usability showed that larger angle increments were quicker to process and saved time on a point to point basis but lacked accuracy, whereas the smaller angle increments to longer to process on a point to point basis and were more accurate. This showed that the angle increment affected time spent manoeuvring the robotic arm and could be adjusted depending on the task that it was assigned.

The real time response of the voice recognition algorithm was deemed satisfactory for the current method of control. Any work with continuous movement would require faster system response to uphold levels of safety and protect the machine from operational damage.

A qualitative analysis of the project brought forward results based on system usability, industrial application and constraints. It was found that the project had:

- Moderate learnability, requiring the user to have a solid understanding of the processes in order to operate the system.
- High efficiency, allowing the user to increase productivity as he/she learnt to operate the system better
- Adequate Memorability, allowing the user to operate the system easily after spending a considerable amount of time away from it
- Occasional Errors, usually involved with careless use of commands and not knowing the processes within the system well enough
- Great Satisfaction, allowing the user to enjoy themselves whilst using the system

Industrial application for this project involved using it as a jogging mechanism during the testing stages of processes or maintenance of a machine. Here the user does not have to worry about coding and using the flex pendent for the jog operations and can instead, use his/her hands for more important matters. The constraints involved in the project have also been identified and potential improvement on these to be discussed in 6.3.

The entire design, implementation and testing process has been observed above. The success of the system has been demonstrated and its real-world implementation discussed. The

importance issues involving recognition error rates, timing and usability have been explored and evaluated to give an effective overview of the system. Overall, the goals of this project have been met and the system leaves room for improvement. Excerpts of the final source code can be found in Appendix D.

6.3 Further Work

The completion of this project has left a number of areas within it that could be open to further work. Improvements in these areas would not only increase the overall robustness of the system but allow it to be used in more applications and expand into others areas of robotics.

During the design stage of the project, a number of techniques have been implemented to obtain Mel Frequency Cepstral Coefficients. Considering only 13 of a possible 39 coefficients have been calculated, extra work could be done here to obtain more and improve the representation of each of the command utterances. This is because these extra coefficients represent more properties of each utterance. The recognition stage of the algorithm could also be improved through further work involving those discussed in sections 2.3.8-2.3.11 as this could not be done with the time available. This would allow for the expansion of the command vocabulary and allow control of more variables and axes. The addition of a noise rejection element to the front end of the system would allow the algorithm to work under noisy conditions (as seen in an industrial workplace) and increase its functionality. Other than these, the processing speeds could be improved with better design modularity and data processing techniques.

With regards to the RobotStudio section of the project there are a number of improvements that could be made. The first involves changing the desired movement from incremental to continuous. This would require the continuous execution of axis movement and use flags or interrupts to stop these movements. This would essentially decrease the amount of times the user speaks a command and produce greater point to point accuracy. This change would require improvements to system response as well to ensure commands were processed and executed almost immediately in real time.

Conducting further testing on the system and improving small undesirables would be a major part of future work. This would lead to being able to test the system on the real world robotic arm Garfield. This testing would give a greater sense of the potential of the technology and also greater satisfaction towards the hard work that has been put in throughout this project to get it working. All of the work above requires a lot more time and effort than expected and would be great to see implemented in future years.

References

Brookes, M 1998, *VOICEBOX: Speech Processing Toolbox for MATLAB*, $Id: rfft.m 713 2011-10-16 14:45:43Z dmb $, Department of Electrical and ELectronic Enginnering, Imperial College, Exhibiton Road, London, UK, viewed 12/08/2013

<http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html>.

Communications, N 2013, *Dragon Speech Recognition Software*, viewed 23/05/2013, <http://australia.nuance.com/dragon/index.htm>.

Engineers, Io 1997, *Towards sustainable engineering practice: engineering frameworks for sustainability*, Canberra, Australia.

Gates, B 2007, *A Robot in every home*, Scientific American 2013, viewed 16/05/2013, <file:///C:/Users/Kyle/Desktop/University/Semester%201%202013/Project/A%20Robot%20in%20Every%20Home%20%20%20Article%20%20%20Scientific%20American.htm>.

Iqbal, S, Mahboob, T & Khiyal, MSH 2011, 'Voice Recognition using HMM with MFCC for Secure ATM', *International Journal of Computer Science Issues (IJCSI)*, vol. 8, no. 6, pp. 297-303, EBSCOhost, iih, item: 73204491.

Juang, BH & Tsuhan, C 1998, 'The past, present, and future of speech processing', *Signal Processing Magazine, IEEE*, vol. 15, no. 3, pp. 24-48.

Kumar, S & Rao, PM 2011, 'Design Of An Automatic Speaker Recognition System Using MFCC, Vector Quantization And LBG Algorithm', *International Journal on Computer Science & Engineering*, vol. 3, no. 8, pp. 2942-54, EBSCOhost, a9h, item: 67367968.

Lyons, J 2009, *Mel Frequency Cepstral Coefficient (MFCC) tutorial*, Practical Cryptography, viewed 10/09/2013,

<http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/>.

Muda, L, Begam, M & Elamvazuthi, I 2010, 'Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques', *arXiv preprint arXiv:1003.4083*.

Pires, JN 2005, *Robot-by-voice: experiments on commanding an industrial robot using the human voice*, (Mechanical Engineering Department and Mechanical Engineering Research Center (A Research Center from the Portuguese Foundation for Science and Technology), University of Coimbra, Coimbra, Portugal), Emerald Group Publishing Limited,<.

Plannerer, B 2005, *An introduction to speech recognition*.

Rabiner, BHJLR 2004, *Automatic Speech Recognition – A Brief History of the Technology*

Development, Georgia Institute of Technology, Atlanta, Rutgers University and the University of California, Santa Barbara, viewed 15/07/2013

http://www.ece.ucsb.edu/Faculty/Rabiner/ece259/Reprints/354_LALI-ASRHistory-final-10-8.pdf>.

Robotics, A 2013, *Operating Manual - RobotStudio*, ABB AB Robotics Products Sweden.

Robotics, IIFo 2012, *History of Industrial Robots*, c/o VDMA Robotics + Automation.

Appendix A

Project Specification

University of Southern Queensland

FACULTY OF ENGINEERING AND SURVEYING

ENG 4111/4112 Research Project

PROJECT SPECIFICATION

FOR:	KYLE TONKIN
TOPIC:	MAKE GARFIELD (THE 6-AXIS ROBOT ARM) SMART
SUPERVISORS:	Dr. TOBIAS LOW
ENROLMENT:	ENG 4111 – S1, D, 2013
	ENG 4112 – S2, D, 2013
PROJECT AIM:	This project will look at researching and designing a smarter and safer robot arm (Garfield) through the implementation of visual control and/or voice command.
SPONSORSHIP:	UNIVERSITY OF SOUTHERN QUEENSLAND

PROGRAMME: (Issue A, 13 March 2013)

1) Research Background information on 'Garfield' (the 6-axis robot arm) and its operation as well as the principles and implementation of voice command on robot systems.
2) Create a program to process and recognise voice commands.
3) Implement an interfacing system with the robot simulation software allowing the use of voice command through TCP socket messaging between MATLAB and RPG5 (RAPID) programming.
4) Analyse the effects and reliability of voice control when implemented.
5) Achieve a robot arm responsive to voice commands.
6) Analyse the effects of these new improvements and their application to industry.

As time permits:

7) Investigate further improvements available to enhance the functionality of Garfield

AGREED _____(student) _____(supervisor)

Date: / / 2013 Date: / / 2013

Examiner/Co-examiner:_____

Appendix B

Project Management Errata

B.1 Project Timeline

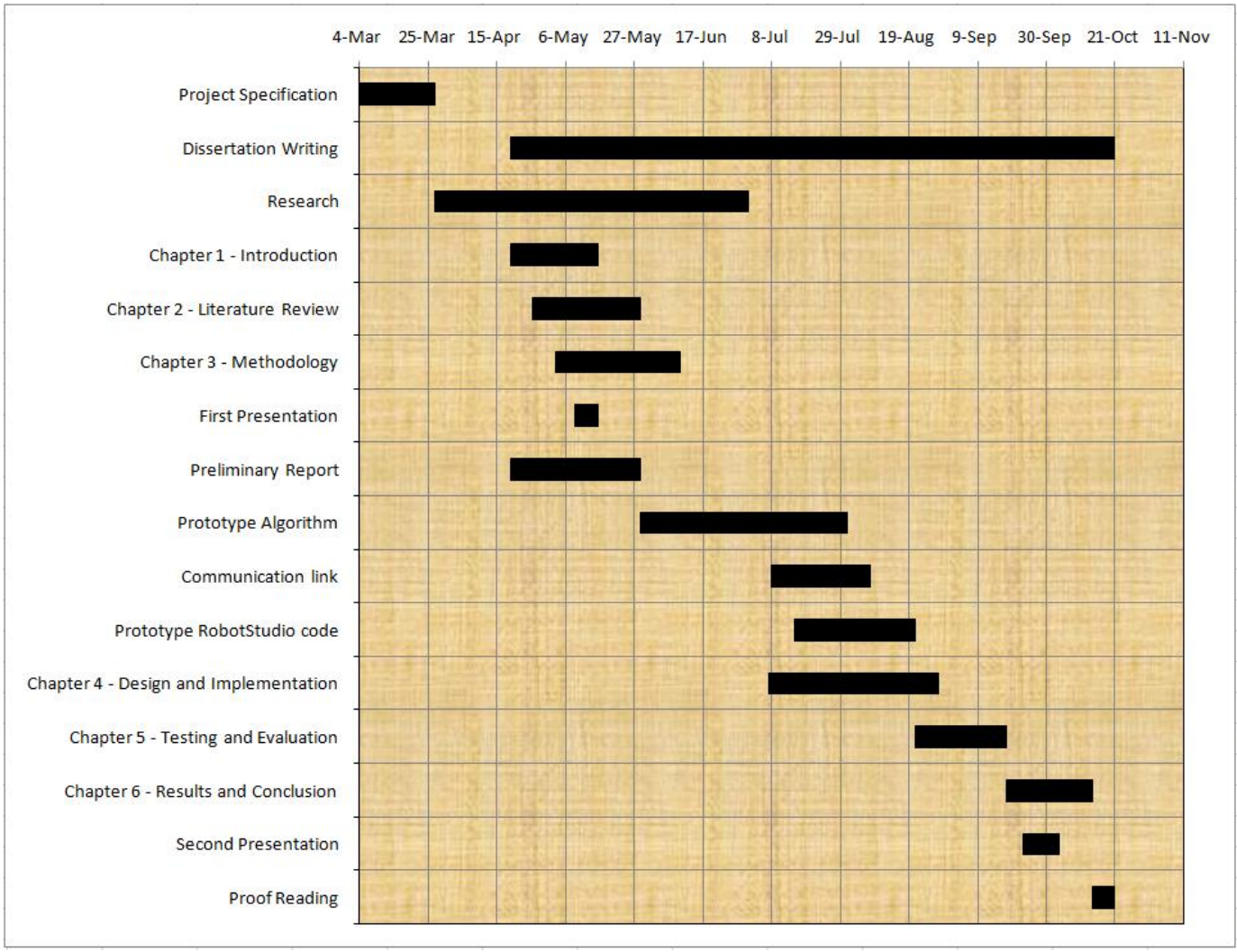


Table 18 - Research Timeline

B.2 Risk Assessment

The following table contains the different levels of risk occurrence:

Level	Risk Level	Description
1	Very Unlikely	Might occur under the right conditions
2	Unlikely	May occur at some point
3	Likely	Will occur at some point
4	Very Likely	Will certainly occur in most conditions

Table 19 - Levels of risk occurrence

The following table contains the level of severity of a risk:

Level	Risk Severity	Description
1	Insignificant	Very low impact on well-being, project completion or environment.
2	Minor	Minor impact on well-being, project completion or environment.
3	Moderate	Moderate impact on well-being, project completion or environment.
4	Major	Major impact on well-being, project completion or environment.
5	Catastrophic	Catastrophic impact on well-being, project completion or environment.

Table 20 - Risk Severity Levels

Hazard Identification

The following risks have been identified as risks to well-being, project completion and environment.

1. Working Conditions

Identified Risks

- Muscle Strain (Repetitive strain injuries, Carpel Tunnel Syndrome).
- Eye Strain.
- Breathing difficulties.
- Headaches.

Solutions

- Ergonomically designed workstation for comfort and safety, as well as correctly placing furniture at ergonomic heights.
- The use of regular breaks and exercises to prevent any further injury.
- Appropriate lighting levels should be maintained to avoid eye strain as well as appropriate placement of screen to avoid glare from windows and shutters.
- Proper ventilation throughout the work area and temperature control.

2. Stress

Identified Risks

- Long hours without much social freedom. Can lead to illness.
- Building pressure of upcoming due dates and deadlines. Can also lead to illness.

Solutions

- Create a balanced schedule with time for everything
- Eat Healthy and ensure regular sleeping patterns

3. Electric Shock

Identified Risks

- Power cords and power boards around workstation (possibly overloaded). Personal injury from this can cause burns and affect productivity.
- Electrical equipment can cause electric shock (statically or dynamically). Can also cause burns or heart attacks.

Solutions

- Ensure all equipment is standardised and power boards are not overloaded or damaged.
- Do not try to access electrical equipment i.e. computers, printers or robots unless qualified to do so.

4. Personal Injury or illness (unforeseen)

Identified Risks

- Unexpected cases of illness, injury or disease i.e. colds, broken leg or chicken pox. These will lead to a decrease in productivity and possibly project failure.

Solutions

- Avoid getting run down and stressed and ensure time is created in the schedule to allow for spare time if need be.
-

5. Equipment Failure and Data loss

Identified Risks

- The breakdown of equipment can cause data loss or even injury in some cases. This can affect project completion and also your well-being.
- Data loss can occur at any stage which can affect project completion and scheduling.

Solutions

- Document procedures in case of data loss so time can be recuperated and retain receipts and warranties to replace equipment.
- Continually back up data in case of system crashes. Weekly or even daily.

Risk Summary

The hazards identified in the previous sections have been ranked by the criteria stated in table 2 and table 3.

Hazard	Risk Level	Severity Level
Working Conditions	2	2
Stress	3	2
Electric Shock	1	5
Personal Injury or Illness	2	4
Equipment failure or Data loss	1	3

Table 21 - Risk Summary

Appendix C

Extra Results

This table contains the results obtained from Background noise testing without the rejection element.

Command	No. Times correct	No. Times incorrect	No. Times rejected	Error Rate = No. Errors/ No.Utterances *100%	
'One'	1	9	-	90%	
'Two'	5	5	-	50%	
'Three'	3	7	-	70%	
'Four'	4	6	-	60%	
'Five'	8	2	-	20%	
'Six'	5	5	-	50%	
'Seven'	4	6	-	60%	
'Eight'	7	3	-	30%	
'Nine'	2	8	-	80%	
'Stop'	8	2	-	20%	
'End'	7	3	-	30%	
'Lower'	0	10	-	100%	**Total Error**
'Higher'	0	10	-	100%	58.5%

Table 22 - Rejection element removal results (Background music)

Appendix D

Source Code

Database Building (excerpt)

```
%% %% Research Project Part I & II
% Kyle Tonkin - 2013

% Command Recording
clc; clear;

fs = 16000;
duration = 2;

% Command Initialisation
one = [];
two = [];
three = [];
four = [];
five = [];
six = [];
seven = [];
eight = [];
nine = [];
stop = [];
end1 = [];
high = [];
low = [];

%% Build database entries *THIS WAS REPEATED FOR EACH COMMAND*

% Entries for word 'one'
input ('press enter for recording of "one"')

for i = 1:10

y = wavrecord(duration*fs,fs);

one = [one y];
```

```
file = sprintf ('%s%d.wav','one',i);

wavwrite(y,file);

if i~=10
input ('press enter for next recording')
else
end

end
```

Database Processing (excerpt of command 1)

```
clc; clear;

f = (1:8000)';
fs = 16000;
duration = 2;
x = 0.5/8000:0.5/8000:0.5;

% Command Matrice Initialisation
one = [];
two = [];
three = [];
four = [];
five = [];
six = [];
seven = [];
eight = [];
nine = [];
stop = [];
end1 = [];
high = [];
low = [];

z = 1;

%% Signal Analysis

% Obtain recorded voice files from folder

*THIS WAS DONE FOR EACH RECORDED COMMAND*

one = wavread('one1.wav');
one = [one wavread('one2.wav')];
one = [one wavread('one3.wav')];
one = [one wavread('one4.wav')];
one = [one wavread('one5.wav')];
one = [one wavread('one6.wav')];
one = [one wavread('one7.wav')];
one = [one wavread('one8.wav')];
one = [one wavread('one9.wav')];
one = [one wavread('one10.wav')];
```

```
%% Pre-emphasis

% Filter coefficients
B = [1 -0.95];

% Pre-emphasis Application
Preemph1 = filter(B,1,one);

Preemph1(1,:) = 0;



%% Silence Detection
% Command 'ONE'

for k = 1:10

    i = 2;

while  abs(Preemph1(i,k))<  0.003  ||  abs(Preemph1(i+20,k))<  0.003  ||
abs(Preemph1(i+40,k))<  0.003  ||  abs(Preemph1(i+60,k))<  0.003  ||
abs(Preemph1(i+80,k))< 0.003 || abs(Preemph1(i+100,k))< 0.003
    i = i+1;
end

a = Preemph1(i:length(Preemph1),k);

if k == 1
    a1= a;
    a1(8001:length(a1))= [];
elseif k==2
    a2= a;
    a2(8001:length(a2))= [];
elseif k==3
    a3= a;
    a3(8001:length(a3))= [];
elseif k==4
    a4= a;
    a4(8001:length(a4))= [];
elseif k==5
    a5= a;
    a5(8001:length(a5))= [];
elseif k==6
    a6= a;
    a6(8001:length(a6))= [];
elseif k==7
    a7= a;
    a7(8001:length(a7))= [];
elseif k==8
    a8= a;
    a8(8001:length(a8))= [];
elseif k==9
    a9= a;
    a9(8001:length(a9))= [];
    elseif k==10
    a10= a;
    a10(8001:length(a10))= [];
end
end
```

```
Preemph1 = [a1 a2 a3 a4 a5 a6 a7 a8 a9 a10];


figure('units','normalized','outerposition',[0 0 1 1])
set(gcf,'numbertitle','off','name','Command One')
subplot(3,4,1)
plot(x,a1)
title('Utterance 1')
xlabel('Time (s)')
ylabel('Amplitude')
subplot(3,4,2)
plot(x,a2)
title('Utterance 2')
xlabel('Time (s)')
ylabel('Amplitude')
subplot(3,4,3)
plot(x,a3)
title('Utterance 3')
xlabel('Time (s)')
ylabel('Amplitude')
subplot(3,4,4)
plot(x,a4)
title('Utterance 4')
xlabel('Time (s)')
ylabel('Amplitude')
subplot(3,4,5)
plot(x,a5)
title('Utterance 5')
xlabel('Time (s)')
ylabel('Amplitude')
subplot(3,4,6)
plot(x,a6)
title('Utterance 6')
xlabel('Time (s)')
ylabel('Amplitude')
subplot(3,4,7)
plot(x,a7)
title('Utterance 7')
xlabel('Time (s)')
ylabel('Amplitude')
subplot(3,4,8)
plot(x,a8)
title('Utterance 8')
xlabel('Time (s)')
ylabel('Amplitude')
subplot(3,4,9)
plot(x,a9)
title('Utterance 9')
xlabel('Time (s)')
ylabel('Amplitude')
subplot(3,4,10)
plot(x,a10)
title('Utterance 10')
xlabel('Time (s)')
ylabel('Amplitude')

ylim([-0.2 0.2])

linkaxes
```

```
%% Frame by frame analysis
seglength = 320; % Length of frames
stepsize = seglength/2; % Frame step size
nframes = length(Preemph1)/stepsize-1;

% 320 point Hamming window
np = seglength;
h = (0.54 - 0.46*cos(2*pi*(0:np-1)/(np-1))');


dxenergy1 = [];
dxenergy2 = [];
dxenergy3 = [];
dxenergy4 = [];
dxenergy5 = [];
dxenergy6 = [];
dxenergy7 = [];
dxenergy8 = [];
dxenergy9 = [];
dxenergy10 = [];
dxenergy11 = [];
dxenergy12 = [];
dxenergy13 = [];


% Command 'ONE'

for k = 1:10
% Initialize Variables
samp1 = 1; samp2 = seglength; %Initialize frame start and end

for i = 1:nframes
% Get current frame for analysis
frame = Preemph1(samp1:samp2,k);

% Analysis on frames

% Multiply by Hamming window and compute 512 point DFT
% Keep first 257 coefficients
frame = rfft(h.*frame,512);

% Periodgram estimate of power spectrum
frame = (1/seglength).*((abs(frame)).^2);

% Mel Filter bank
frameA = melbankm(26,512,16000);

% Energy in each filterbank
energy = frameA*frame;

% Convert to log filter bank energies
energy = log(energy);

% Perform discrete cosine transform
dxenergy = dct_kyle(energy');

dxenergy = dxenergy';
```

```
% Keep 13 lower coefficients
dxenergy = dxenergy(1:13);

z = 14;
figure(z)
subplot(3,4,k)
plot(samp1:samp2-63,abs(frame))
hold on
% Step up to next frame of speech
samp1 = samp1 + stepsize;
samp2 = samp2 + stepsize;

dxenergy1 = [dxenergy1 dxenergy];
end

if k == 1
    one1 = dxenergy1;
elseif k== 2
    one2 = dxenergy1;
elseif k==3
    one3 = dxenergy1;
elseif k==4
    one4 = dxenergy1;
elseif k==5
    one5 = dxenergy1;
elseif k == 6
    one6 = dxenergy1;
elseif k== 7
    one7 = dxenergy1;
elseif k==8
    one8 = dxenergy1;
elseif k==9
    one9 = dxenergy1;
elseif k==10
    one10 = dxenergy1;
end
% Re-initialise
dxenergy1 = [];

end

linkaxes

z = z+1;

%% MFCC Command Matrices

% Command Model 'ONE'
oneDB = [one1; one2; one3; one4; one5; one6; one7; one8; one9; one10];
```

Command Recognition

```
% Voice recognition code

% Server Variables
output_port = 12345;
number_of_retries = 2;


% Enter Voice Input Loop
for m = 1:100

% Initialise Variables
% Frequency

fs = 16000;
duration = 2;
x = 0.5/8000:0.5/8000:0.5;

% Filter coefficients
B = [1 -0.95];

% Initialise 'i' increment
i=1;

%% Obtain and Process Command

% Command Input
input ('press enter for command')

ytest = wavrecord(duration*fs,fs);

%figure(1)
%plot(1/16000:1/16000:2, ytest)
%plot(1/16000:1/16000:2, Preemphtest)
%ylabel('Amplitude')
%xlabel('Time (s)')
%title('Command "One" Waveform')
%title('Command "One" Waveform (Pre-Emphasised)')

%axis tight

% Pre-emphasis of command

Preemphtest = filter(B,1,ytest);

% Set initial value of Preemphtest matrix to zero
Preemphtest(1) = 0;

%figure(2)
%plot(Preemphtest)

% Silence Detection
% Wait for clear start of signal
while abs(Preemphtest(i))< 0.003 || abs(Preemphtest(i+20))< 0.003 ||
abs(Preemphtest(i+40))< 0.003 || abs(Preemphtest(i+60))< 0.003 ||
abs(Preemphtest(i+80))< 0.003 || abs(Preemphtest(i+100))< 0.003
    i = i+1;
end
```

```
% Signal will start from the calculated position
atest = Preemphtest(i:length(Preemphtest));

% Reduce samples from 16000 to 8000 for remainder of processing
(command
% will be present within the 8000 samples)
atest(8001:length(atest))= [];

Preemphtest = atest;

%figure(3)
%plot(x, atest)
%ylabel('Amplitude')
%xlabel('Time (s)')
%title('Command "One" Waveform after silence detection')

%% Frame by frame analysis
% Initialise variables
% Frame Length
seglength = 320;
% Frame step size
stepsize = seglength/2;
% Number of frames
nframes = length(Preemphtest)/stepsize-1;
% Create Hamming window
np = seglength;
h = (0.54 - 0.46*cos(2*pi*(0:np-1)/(np-1))');

%plot(0.02/320:0.02/320:0.02,h)
%axis tight
%title('320 point Hamming Window')
%xlabel('Time (s)')
%ylabel('Amplitude')

% Initialise energy matrix
dxenergy1 = [];

%Initialize frame start and end
samp1 = 1; samp2 = seglength;

for i = 1:nframes
% Get current frame for analysis
frame = Preemphtest(samp1:samp2);

% Analysis on frames

% Multiply by Hamming window and compute 512 point DFT
% Keep first 257 coefficients
frame = rfft(h.*frame,512);

% Periodogram estimate of power spectrum
frame = (1/seglength).*((abs(frame)).^2);


% Mel Filter bank
frameA = melbankm(26,512,fs);

% Energy in each filterbank
```

```
energy = frameA*frame;

% Convert to log filter bank energies
energy = log(energy);

% Perform discrete cosine transform
dxenergy = dct_kyle(energy');

dxenergy = dxenergy';

% Keep 13 lower coefficients
dxenergy = dxenergy(1:13);

%figure(4)
%plot(samp1:samp2-63,abs(frame))
%hold on
% Step up to next frame of speech
samp1 = samp1 + stepsize;
samp2 = samp2 + stepsize;


dxenergy1 = [dxenergy1 dxenergy];
end
% Classify matrix 'ytest1'
ytest1 = dxenergy1;



%% Recognition using Euclidean Distance between MFCC coefficients


% Number of coefficients
p = 13;

% Initialise 1st command matrix
S11 = [];
% Initialise 'i' increment for loop
i = 0;

% Calculate Euchlidean distances of each of the 10 utterances
for k = 1:10
S = sum((oneDB(i+1:(k*p),:) - ytest1).^2);
S = sum(S);

i = i+13;

S11 = [S11 S];

end
S11 = S11./1000;


%% Initialise 2nd command matrix
S22 = [];
% Initialise 'i' increment for loop
i = 0;
```

```
% Calculate Euchlidean distances of each of the 10 utterances
for k = 1:10
S = sum((twoDB(i+1:(k*p),:) - ytest1).^2);
S = sum(S);

i = i+13;

S22 = [S22 S];
end
S22 = S22./1000;


%% Initialise 3rd command matrix
S33 = [];
% Initialise 'i' increment for loop
i = 0;

% Calculate Euchlidean distances of each of the 10 utterances
for k = 1:10
S = sum((threeDB(i+1:(k*p),:) - ytest1).^2);
S = sum(S);

i = i+13;

S33 = [S33 S];
end
S33 = S33./1000;


%% Initialise 4th command matrix
S44 = [];
% Initialise 'i' increment for loop
i = 0;

% Calculate Euchlidean distances of each of the 10 utterances
for k = 1:10
S = sum((fourDB(i+1:(k*p),:) - ytest1).^2);
S = sum(S);

i = i+13;

S44 = [S44 S];
end
S44 = S44./1000;


%% Initialise 5th command matrix
S55 = [];
% Initialise 'i' increment for loop
i = 0;

% Calculate Euchlidean distances of each of the 10 utterances
for k = 1:10
S = sum((fiveDB(i+1:(k*p),:) - ytest1).^2);
S = sum(S);

i = i+13;
```

```
S55 = [S55 S];
end
S55 = S55./1000;


%% Initialise 6th command matrix
S66 = [];
% Initialise 'i' increment for loop
i = 0;

% Calculate Euchlidean distances of each of the 10 utterances
for k = 1:10
S = sum((sixDB(i+1:(k*p),:) - ytest1).^2);
S = sum(S);

i = i+13;

S66 = [S66 S];
end
S66 = S66./1000;


%% Initialise 7th command matrix
S77 = [];
% Initialise 'i' increment for loop
i = 0;

% Calculate Euchlidean distances of each of the 10 utterances
for k = 1:10
S = sum((sevenDB(i+1:(k*p),:) - ytest1).^2);
S = sum(S);

i = i+13;

S77 = [S77 S];
end
S77 = S77./1000;


%% Initialise 8th command matrix
S88 = [];
% Initialise 'i' increment for loop
i = 0;

% Calculate Euchlidean distances of each of the 10 utterances
for k = 1:10
S = sum((eightDB(i+1:(k*p),:) - ytest1).^2);
S = sum(S);

i = i+13;

S88 = [S88 S];
end
S88 = S88./1000;


%% Initialise 9th command matrix
S99 = [];
```

```
% Initialise 'i' increment for loop
i = 0;

% Calculate Euchlidean distances of each of the 10 utterances
for k = 1:10
S = sum((nineDB(i+1:(k*p),:) - ytest1).^2);
S = sum(S);

i = i+13;

S99 = [S99 S];
end
S99 = S99./1000;

%% Initialise 10th command matrix
S101 = [];
% Initialise 'i' increment for loop
i = 0;

% Calculate Euchlidean distances of each of the 10 utterances
for k = 1:10
S = sum((stopDB(i+1:(k*p),:) - ytest1).^2);
S = sum(S);

i = i+13;

S101 = [S101 S];
end
S101 = S101./1000;

%% Initialise 11th command matrix
S111 = [];
% Initialise 'i' increment for loop
i = 0;

% Calculate Euchlidean distances of each of the 10 utterances
for k = 1:10
S = sum((end1DB(i+1:(k*p),:) - ytest1).^2);
S = sum(S);

i = i+13;

S111 = [S111 S];
end
S111 = S111./1000;

%% Initialise 12th command matrix
S121 = [];
% Initialise 'i' increment for loop
i = 0;

% Calculate Euchlidean distances of each of the 10 utterances
for k = 1:10
S = sum((highDB(i+1:(k*p),:) - ytest1).^2);
S = sum(S);

i = i+13;
```

```
S121 = [S121 S];
end
S121 = S121./1000;

%% Initialise 13th command matrix
S131 = [];
% Initialise 'i' increment for loop
i = 0;

% Calculate Euchlidean distances of each of the 10 utterances
for k = 1:10
S = sum((lowDB(i+1:(k*p),:) - ytest1).^2);
S = sum(S);

i = i+13;

S131 = [S131 S];
end
S131 = S131./1000;

%% Calculate closest match for each command
S1 = min(S11)
S2 = min(S22)
S3 = min(S33)
S4 = min(S44)
S5 = min(S55)
S6 = min(S66)
S7 = min(S77)
S8 = min(S88)
S9 = min(S99)
S10 = min(S101)
S11 = min(S111)
S12 = min(S121)
S13 = min(S131)

% Arrange them into a matrix
MSET = [S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13];

% Find closest of all the commands
MSETmin = min(MSET);

% Test for error
if MSETmin > 40

    fprintf('\n\nERROR\n\n')

% Determine the corresponding command and Output
elseif MSETmin == MSET(1)

    % Print command to command window
    fprintf('\n\nThe command is one\n\n')

    % Start timing clock
ticID = tic;
    % Create command string ready for output
    message = 'one';

    % Set up scoket communication
    server(message,output_port, number_of_retries)
```

```
    %s = wavread('one1.wav');
    %sound(s, 16000)
        % Stop timing clock
elapsedTime = toc(ticID);

elseif MSETmin == MSET(2)

    % Print command to command window
    fprintf('\n\nThe command is two\n\n')
    % Create command string ready for output
    message = 'two';
    % Set up scoket communication
    server(message,output_port, number_of_retries)

    %s = wavread('two1.wav');
    %sound(s, 16000)

elseif MSETmin == MSET(3)

    % Print command to command window
    fprintf('\n\nThe command is three\n\n')
    % Create command string ready for output
    message = 'three';
    % Set up scoket communication
    server(message,output_port, number_of_retries)

    %s = wavread('three1.wav');
    %sound(s, 16000)

elseif MSETmin == MSET(4)

    % Print command to command window
    fprintf('\n\nThe command is four\n\n')
    % Create command string ready for output
    message = 'four';
    % Set up scoket communication
    server(message,output_port, number_of_retries)

    %s = wavread('four1.wav');
    %sound(s, 16000)

elseif MSETmin == MSET(5)

    % Print command to command window
    fprintf('\n\nThe command is five\n\n')
    % Create command string ready for output
    message = 'five';
    % Set up scoket communication
    server(message,output_port, number_of_retries)

    %s = wavread('five1.wav');
    %sound(s, 16000)

elseif MSETmin == MSET(6)

    % Print command to command window
    fprintf('\n\nThe command is six\n\n')
```

```
    % Create command string ready for output
    message = 'six';
    % Set up scoket communication
    server(message,output_port, number_of_retries)

    %s = wavread('six1.wav');
    %sound(s, 16000)

elseif MSETmin == MSET(7)

    % Print command to command window
    fprintf('\n\nThe command is seven\n\n')
    % Create command string ready for output
    message = 'seven';
    % Set up scoket communication
    server(message,output_port, number_of_retries)

    %s = wavread('seven1.wav');
    %sound(s, 16000)

elseif MSETmin == MSET(8)

    % Print command to command window
    fprintf('\n\nThe command is eight\n\n')
    % Create command string ready for output
    message = 'eight';
    % Set up scoket communication
    server(message,output_port, number_of_retries)

    %s = wavread('eight1.wav');
    %sound(s, 16000)


elseif MSETmin == MSET(9)

    % Print command to command window
    fprintf('\n\nThe command is nine\n\n')
    % Create command string ready for output
    message = 'nine';
    % Set up scoket communication
    server(message,output_port, number_of_retries)

    %s = wavread('nine1.wav');
    %sound(s, 16000)


elseif MSETmin == MSET(10)

    % Print command to command window
    fprintf('\n\nThe command is stop\n\n')
    % Create command string ready for output
    message = 'stop';
    % Set up scoket communication
    server(message,output_port, number_of_retries)

    %s = wavread('stop1.wav');
    %sound(s, 16000)
```

```
elseif MSETmin == MSET(11)

    % Print command to command window
    fprintf('\n\nThe command is end\n\n')
    % Create command string ready for output
    message = 'end';
    % Set up scoket communication
    server(message,output_port, number_of_retries)

    %s = wavread(end11.wav');
    %sound(s, 16000)

    break;

elseif MSETmin == MSET(12)

    % Print command to command window
    fprintf('\n\nThe command is higher\n\n')
    % Create command string ready for output
    message = 'higher';
    % Set up scoket communication
    server(message,output_port, number_of_retries)

    %s = wavread('high1.wav');
    %sound(s, 16000)

elseif MSETmin == MSET(13)

    % Print command to command window
    fprintf('\n\nThe command is lower\n\n')

    % Create command string ready for output
    message = 'lower';
    % Set up scoket communication
    server(message,output_port, number_of_retries)

    %s = wavread('low1.wav');
    %sound(s, 16000)

else
    fprintf('\n\nERROR\n\n')

end

end
```

DCT Function

```
function X = dct_kyle(Data)

S_=size(Data);
N=S_(2);

 Sk(1:N)=1; Sk(1)=1/sqrt(2);
   for k=0:N-1
     X(k+1)=0;
    for i=0:N-1
        X(k+1)=X(k+1)+(Data(i+1).*cos(pi.*k.*(i+0.5)./N));
    end
     X(k+1)=Sk(k+1).*sqrt(2/N).*X(k+1);
   end
end
```

Voicebox Functions (Brookes 1998)

MELBANKM

```
function [x,mc,mn,mx]=melbankm(p,n,fs,fl,fh,w)

%MELBANKM    determine    matrix    for    a    mel/erb/bark-spaced    filterbank
[X,MN,MX]=(P,N,FS,FL,FH,W)

%

% Inputs:

%          p    number of filters in filterbank or the filter spacing in k-
mel/bark/erb [ceil(4.6*log10(fs))]

%       n   length of fft

%       fs  sample rate in Hz

%       fl  low end of the lowest filter as a fraction of fs [default = 0]

%       fh  high end of highest filter as a fraction of fs [default = 0.5]

%       w   any sensible combination of the following:

%             'b' = bark scale instead of mel

%             'e' = erb-rate scale

%             'l' = log10 Hz frequency scale

%             'f' = linear frequency scale

%

%             'c' = fl/fh specify centre of low and high filters
```

```
%              'h' = fl/fh are in Hz instead of fractions of fs
%              'H' = fl/fh are in mel/erb/bark/log10
%
%              't' = triangular shaped filters in mel/erb/bark domain (default)
%              'n' = hanning shaped filters in mel/erb/bark domain
%              'm' = hamming shaped filters in mel/erb/bark domain
%
%              'z' = highest and lowest filters taper down to zero [default]
%              'y' = lowest filter remains at 1 down to 0 frequency and
%                    highest filter remains at 1 up to nyquist freqency
%
%              'u' = scale filters to sum to unity
%
%                   's' = single-sided: do not double filters to account for
negative frequencies
%
%                 'g' = plot idealized filters [default if no output arguments
present]
%
% Note that the filter shape (triangular, hamming etc) is defined in the mel
(or erb etc) domain.
% Some people instead define an asymmetric triangular filter in the frequency
domain.
%
%                 If 'ty' or 'ny' is specified, the total power in the fft is
preserved.
%
% Outputs:  x     a sparse matrix containing the filterbank amplitudes
%                 If the mn and mx outputs are given then size(x)=[p,mx-mn+1]
%                 otherwise size(x)=[p,1+floor(n/2)]
%                  Note that the peak filter values equal 2 to account for the
power
%                 in the negative FFT frequencies.
%           mc    the filterbank centre frequencies in mel/erb/bark
%           mn    the lowest fft bin with a non-zero coefficient
%           mx    the highest fft bin with a non-zero coefficient
%                 Note: you must specify both or neither of mn and mx.
%
```

```
% References:
%
% [1] S. S. Stevens, J. Volkman, and E. B. Newman. A scale for the measurement
%     of the psychological magnitude of pitch. J. Acoust Soc Amer, 8: 185-19,
1937.
% [2] S. Davis and P. Mermelstein. Comparison of parametric representations
for
%     monosyllabic word recognition in continuously spoken sentences.
%     IEEE Trans Acoustics Speech and Signal Processing, 28 (4): 357-366, Aug.
1980.



%      Copyright (C) Mike Brookes 1997-2009
%      Version: $Id: melbankm.m 713 2011-10-16 14:45:43Z dmb $
%
%   VOICEBOX is a MATLAB toolbox for speech processing.
%   Home page: http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%   This program is free software; you can redistribute it and/or modify
%   it under the terms of the GNU General Public License as published by
%   the Free Software Foundation; either version 2 of the License, or
%   (at your option) any later version.
%
%   This program is distributed in the hope that it will be useful,
%   but WITHOUT ANY WARRANTY; without even the implied warranty of
%   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%   GNU General Public License for more details.
%
%   You can obtain a copy of the GNU General Public License from
%   http://www.gnu.org/copyleft/gpl.html or by writing to
%   Free Software Foundation, Inc.,675 Mass Ave, Cambridge, MA 02139, USA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%

% Note "FFT bin_0" assumes DC = bin 0 whereas "FFT bin_1" means DC = bin 1
```

```
if nargin < 6
    w='tz'; % default options
    if nargin < 5
        fh=0.5; % max freq is the nyquist
        if nargin < 4
            fl=0; % min freq is DC
        end
    end
end
sfact=2-any(w=='s');   % 1 if single sided else 2
wr=' ';   % default warping is mel
for i=1:length(w)
    if any(w(i)=='lebf');
        wr=w(i);
    end
end
if any(w=='h') || any(w=='H')
    mflh=[fl fh];
else
    mflh=[fl fh]*fs;
end
if ~any(w=='H')
    switch wr
                    case 'f'       % no transformation
        case 'l'
            if fl<=0
                error('Low frequency limit must be >0 for l option');
            end
            mflh=log10(mflh);       % convert frequency limits into log10 Hz
        case 'e'
            mflh=frq2erb(mflh);       % convert frequency limits into erb-rate
        case 'b'
            mflh=frq2bark(mflh);       % convert frequency limits into bark
        otherwise
```

```
            mflh=frq2mel(mflh);       % convert frequency limits into mel
    end
end
melrng=mflh*(-1:2:1)';          % mel range
fn2=floor(n/2);     % bin index of highest positive frequency (Nyquist if n is
even)
if isempty(p)
    p=ceil(4.6*log10(fs));         % default number of filters
end
if any(w=='c')              % c option: specify fiter centres not edges
if p<1
    p=round(melrng/(p*1000))+1;
end
melinc=melrng/(p-1);
mflh=mflh+(-1:2:1)*melinc;
else
    if p<1
    p=round(melrng/(p*1000))-1;
end
melinc=melrng/(p+1);
end


%
% Calculate the FFT bins corresponding to [filter#1-low filter#1-mid filter#p-
mid filter#p-high]
%
switch wr
        case 'f'
        blim=(mflh(1)+[0 1 p p+1]*melinc)*n/fs;
    case 'l'
        blim=10.^(mflh(1)+[0 1 p p+1]*melinc)*n/fs;
    case 'e'
        blim=erb2frq(mflh(1)+[0 1 p p+1]*melinc)*n/fs;
    case 'b'
        blim=bark2frq(mflh(1)+[0 1 p p+1]*melinc)*n/fs;
    otherwise
```

```
        blim=mel2frq(mflh(1)+[0 1 p p+1]*melinc)*n/fs;
end
mc=mflh(1)+(1:p)*melinc;    % mel centre frequencies
b1=floor(blim(1))+1;            % lowest FFT bin_0 required might be negative)
b4=min(fn2,ceil(blim(4))-1);    % highest FFT bin_0 required
%
% now map all the useful FFT bins_0 to filter1 centres
%
switch wr
        case 'f'
        pf=((b1:b4)*fs/n-mflh(1))/melinc;
    case 'l'
        pf=(log10((b1:b4)*fs/n)-mflh(1))/melinc;
    case 'e'
        pf=(frq2erb((b1:b4)*fs/n)-mflh(1))/melinc;
    case 'b'
        pf=(frq2bark((b1:b4)*fs/n)-mflh(1))/melinc;
    otherwise
        pf=(frq2mel((b1:b4)*fs/n)-mflh(1))/melinc;
end
%
%  remove any incorrect entries in pf due to rounding errors
%
if pf(1)<0
    pf(1)=[];
    b1=b1+1;
end
if pf(end)>=p+1
    pf(end)=[];
    b4=b4-1;
end
fp=floor(pf);                   % FFT bin_0 i contributes to filters_1 fp(1+i-
b1)+[0 1]
pm=pf-fp;                       % multiplier for upper filter
k2=find(fp>0,1);    % FFT bin_1 k2+b1 is the first to contribute to both upper
and lower filters
```

```
k3=find(fp<p,1,'last');  % FFT bin_1 k3+b1 is the last to contribute to both
upper and lower filters

k4=numel(fp); % FFT bin_1 k4+b1 is the last to contribute to any filters

if isempty(k2)

    k2=k4+1;

end

if isempty(k3)

    k3=0;

end

if any(w=='y')          % preserve power in FFT

    mn=1; % lowest fft bin required (1 = DC)

    mx=fn2+1; % highest fft bin required (1 = DC)

    r=[ones(1,k2+b1-1)   1+fp(k2:k3)   fp(k2:k3)   repmat(p,1,fn2-k3-b1+1)];   %
filter number_1

    c=[1:k2+b1-1 k2+b1:k3+b1 k2+b1:k3+b1 k3+b1+1:fn2+1]; % FFT bin1

    v=[ones(1,k2+b1-1) pm(k2:k3) 1-pm(k2:k3) ones(1,fn2-k3-b1+1)];

else

    r=[1+fp(1:k3) fp(k2:k4)]; % filter number_1

    c=[1:k3 k2:k4]; % FFT bin_1 - b1

    v=[pm(1:k3) 1-pm(k2:k4)];

    mn=b1+1; % lowest fft bin_1

    mx=b4+1;  % highest fft bin_1

end

if b1<0

    c=abs(c+b1-1)-b1+1;     % convert negative frequencies into positive

end

% end

if any(w=='n')

    v=0.5-0.5*cos(v*pi);      % convert triangles to Hanning

elseif any(w=='m')

    v=0.5-0.46/1.08*cos(v*pi);  % convert triangles to Hamming

end

if sfact==2  % double all except the DC and Nyquist (if any) terms

    msk=(c+mn>2) & (c+mn<n-fn2+2);  % there is no Nyquist term if n is odd

    v(msk)=2*v(msk);

end
```

```
%
% sort out the output argument options
%
if nargout > 2
    x=sparse(r,c,v);
    if nargout == 3     % if exactly three output arguments, then
        mc=mn;          % delete mc output for legacy code compatibility
        mn=mx;
    end
else
    x=sparse(r,c+mn-1,v,p,1+fn2);
end
if any(w=='u')
    sx=sum(x,2);
    x=x./repmat(sx+(sx==0),1,size(x,2));
end
%
% plot results if no output arguments or g option given
%
if ~nargout || any(w=='g') % plot idealized filters
    ng=201;     % 201 points
    me=mflh(1)+(0:p+1)'*melinc;
    switch wr
                case 'f'
            fe=me; % defining frequencies
            xg=repmat(linspace(0,1,ng),p,1).*repmat(me(3:end)-me(1:end-
2),1,ng)+repmat(me(1:end-2),1,ng);
        case 'l'
            fe=10.^me; % defining frequencies
            xg=10.^(repmat(linspace(0,1,ng),p,1).*repmat(me(3:end)-me(1:end-
2),1,ng)+repmat(me(1:end-2),1,ng));
        case 'e'
            fe=erb2frq(me); % defining frequencies
            xg=erb2frq(repmat(linspace(0,1,ng),p,1).*repmat(me(3:end)-
me(1:end-2),1,ng)+repmat(me(1:end-2),1,ng));
        case 'b'
```

```
            fe=bark2frq(me); % defining frequencies

            xg=bark2frq(repmat(linspace(0,1,ng),p,1).*repmat(me(3:end)-
me(1:end-2),1,ng)+repmat(me(1:end-2),1,ng));

        otherwise

            fe=mel2frq(me); % defining frequencies

            xg=mel2frq(repmat(linspace(0,1,ng),p,1).*repmat(me(3:end)-
me(1:end-2),1,ng)+repmat(me(1:end-2),1,ng));

    end


    v=1-abs(linspace(-1,1,ng));

    if any(w=='n')

        v=0.5-0.5*cos(v*pi);      % convert triangles to Hanning

    elseif any(w=='m')

        v=0.5-0.46/1.08*cos(v*pi);  % convert triangles to Hamming

    end

    v=v*sfact;  % multiply by 2 if double sided

    v=repmat(v,p,1);

    if any(w=='y')  % extend first and last filters

        v(1,xg(1,:)<fe(2))=sfact;

        v(end,xg(end,:)>fe(p+1))=sfact;

    end

    if any(w=='u') % scale to unity sum

        dx=(xg(:,3:end)-xg(:,1:end-2))/2;

        dx=dx(:,[1 1:ng-2 ng-2]);

        vs=sum(v.*dx,2);

        v=v./repmat(vs+(vs==0),1,ng)*fs/n;

    end

    plot(xg',v','b');

    set(gca,'xlim',[fe(1) fe(end)]);

    xlabel(['Frequency (' xticksi 'Hz)']);

end
```

RFFT.M

```
function y=rfft(x,n,d)
%RFFT     Calculate the DFT of real data Y=(X,N,D)
% Data is truncated/padded to length N if specified.
%   N even: (N+2)/2 points are returned with
%           the first and last being real
%   N odd:  (N+1)/2 points are returned with the
%           first being real
% In all cases fix(1+N/2) points are returned
% D is the dimension along which to do the DF
%      Copyright (C) Mike Brookes 1998
%      Version: $Id: rfft.m 713 2011-10-16 14:45:43Z dmb $
%
%   VOICEBOX is a MATLAB toolbox for speech processing.
%   Home page: http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   This program is free software; you can redistribute it and/or modify
%   it under the terms of the GNU General Public License as published by
%   the Free Software Foundation; either version 2 of the License, or
%   (at your option) any later version.
%
%   This program is distributed in the hope that it will be useful,
%   but WITHOUT ANY WARRANTY; without even the implied warranty of
%   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%   GNU General Public License for more details.
%
%   You can obtain a copy of the GNU General Public License from
%   http://www.gnu.org/copyleft/gpl.html or by writing to
%   Free Software Foundation, Inc.,675 Mass Ave, Cambridge, MA 02139, USA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
s=size(x);

if prod(s)==1

    y=x

else

    if nargin <3 || isempty(d)

        d=find(s>1,1);

        if nargin<2

            n=s(d);

        end

    end

    if isempty(n)

        n=s(d);

    end

    y=fft(x,n,d);

    y=reshape(y,prod(s(1:d-1)),n,prod(s(d+1:end)));

    s(d)=1+fix(n/2);

    y(:,s(d)+1:end,:)=[];

    y=reshape(y,s);

end
```

Server.M

```
%% Socket code - Server
% SERVER Write a message over the specified port
%
% Usage - server(message, output_port, number_of_retries)
function server(message, output_port, number_of_retries)

    import java.net.ServerSocket
    import java.io.*

    if (nargin < 3)
        number_of_retries = 20; % set to -1 for infinite
    end
    retry             = 0;

    server_socket  = [];
    output_socket  = [];

    while true
```

```
        retry = retry + 1;

        try
            if    ((number_of_retries    >    0)    &&    (retry    >
number_of_retries))
                fprintf(1, 'Too many retries\n');
                break;
            end

            fprintf(1, ['Try %d waiting for client to connect to this
' ...
                        'host on port : %d\n'], retry, output_port);

            % wait for 1 second for client to connect server socket
            server_socket = ServerSocket(output_port);
            server_socket.setSoTimeout(1000);

            output_socket = server_socket.accept;

            fprintf(1, 'Client connected\n');

            output_stream   = output_socket.getOutputStream;
            d_output_stream = DataOutputStream(output_stream);

            % output the data over the DataOutputStream
            % Convert to stream of bytes
            fprintf(1, 'Writing %d bytes\n', length(message))
            d_output_stream.writeBytes(char(message));
            d_output_stream.flush;

            % clean up
            server_socket.close;
            output_socket.close;
            break;

        catch
            if ~isempty(server_socket)
                server_socket.close
            end

            if ~isempty(output_socket)
                output_socket.close
            end

            % pause before retrying
            pause(1);
        end
    end
end
```

Client.M

```
%% Socket Client
% CLIENT connect to a server and read a message
%
% Usage - message = client(host, port, number_of_retries)
```

```
function message = client(host, port, number_of_retries)

    import java.net.Socket
    import java.io.*

    if (nargin < 3)
        number_of_retries = 20; % set to -1 for infinite
    end

    retry        = 0;
    input_socket = [];
    message      = [];

    while true

        retry = retry + 1;
        if ((number_of_retries > 0) && (retry > number_of_retries))
            fprintf(1, 'Too many retries\n');
            break;
        end
        try
            fprintf(1, 'Retry %d connecting to %s:%d\n', ...
                    retry, host, port);

            % throws if unable to connect
            input_socket = Socket(host, port);

            % get a buffered data input stream from the socket
            input_stream   = input_socket.getInputStream;
            d_input_stream = DataInputStream(input_stream);

            fprintf(1, 'Connected to server\n');

            % read data from the socket - wait a short time first
            pause(0.5);
            bytes_available = input_stream.available;
            fprintf(1, 'Reading %d bytes\n', bytes_available);

            message = zeros(1, bytes_available, 'uint8');
            for i = 1:bytes_available
                message(i) = d_input_stream.readByte;
            end

            message = char(message);

            % cleanup
            input_socket.close;
            break;

        catch
            if ~isempty(input_socket)
                input_socket.close;
            end
            % pause before retrying
            pause(1);
        end
    end
end
```

RobotStudio Code

```
MODULE Socket

VAR string message;

VAR num retry_no := 0;

VAR socketdev client_socket;

VAR num speed := 100;


CONST num retry1:= 40;

CONST string one := "one";
CONST string two := "two";
CONST string three := "three";
CONST string four := "four";
CONST string five := "five";
CONST string six := "six";
CONST string seven := "seven";
CONST string eight := "eight";
CONST string nine := "nine";
CONST string stop := "stop";
CONST string end := "end";
CONST string higher := "higher";
CONST string lower := "lower";

CONST string error1 := "ERROR - Axis 1 limit";
CONST string error2 := "ERROR - Axis 2 limit";
CONST string error3 := "ERROR - Axis 3 limit";
CONST string error4 := "ERROR - Axis 4 limit";
CONST string demo := "Preparing for Demo";
CONST string demos := "Demo in Progress";
CONST string demof := "Demo Complete";

VAR num inc:= 0;
VAR num inc1:= 0;
VAR num inc2:= 0;
VAR num inc3:= 0;


PROC main()

!Execute CFG data code below if retry error occurs

!WriteCfgData "/SYS/SYS_MISC/NoOfRetry","Value",retry1;

!WarmStart;

FOR i FROM 1 to 1000 DO

next:
SocketConnect1;
SocketReceive1;

! One Command - axis 3 forward
IF message = one  THEN
```

```
TPwrite one;
IF inc2 = -234.9 THEN
inc2:= inc2 + 14.9;
ELSEIF inc2 <40 THEN
inc2:= inc2+20;
ELSEIF inc2 <50 THEN
inc2:= inc2+14.9;
ELSE
TPwrite error3;
GOTO next;
ENDIF
commandone;

! Two Command - axis 2 forward
ELSEIF message = two THEN
TPwrite two;
IF inc1 < 120 THEN
inc1:= inc1+20;
ELSEIF inc1 < 130 THEN
inc1:= inc1+15.9;
ELSE
TPwrite error2;
GOTO next;
ENDIF
commandtwo;

! Three Command - axis 4 clockwise
ELSEIF message = three THEN
TPwrite three;
IF inc3 = -199 THEN
inc3:= inc3 + 19.9;
ELSEIF inc3 <180 THEN
inc3:= inc3+30;
ELSEIF inc3<190 THEN
inc3:= inc3+19.9;
ELSE
TPwrite error4;
ENDIF
commandthree;

! Four Command - axis 1 clockwise
ELSEIF message = four THEN
TPwrite four;
IF inc = -179.9 THEN
inc:= inc + 29.9;
ELSEIF inc < 150 THEN
inc:= inc+30;
ELSEIF inc < 170 THEN
inc:= inc+29.9;
ELSE
TPwrite error1;
GOTO next;
ENDIF
commandfour;

! Five Command - Axis Limit demo
ELSEIF message = five THEN
TPwrite five;
TPwrite demo;
commandfive;
```

```
! Six Command - axis 1 anticlockwise
ELSEIF message = six THEN
TPwrite six;
IF inc = 179.9 THEN
inc:= inc - 29.9;
ELSEIF inc > -150 THEN
inc:= inc-30;
ELSEIF inc > -170 THEN
inc:= inc-29.9;
ELSE
TPwrite error1;
GOTO next;
ENDIF
commandsix;

! Seven Command - axis 4 anticlockwise
ELSEIF message = seven THEN
TPwrite seven;
IF inc3 = 199 THEN
inc3:= inc3 - 19.9;
ELSEIF inc3 >-180 THEN
inc3:= inc3-30;
ELSEIF inc3>-190 THEN
inc3:= inc3-19.9;
ELSE
TPwrite error4;
ENDIF
commandseven;

! Eight Command - axis 2 backward
ELSEIF message = eight THEN
TPwrite eight;
IF inc1 = 135.9 THEN
inc1:= inc1 - 15.9;
ELSEIF inc1 > -50 THEN
inc1:= inc1-20;
ELSE
TPwrite error2;
GOTO next;
ENDIF
commandeight;

! Nine Command - axis 3 backward
ELSEIF message = nine THEN
TPwrite nine;
IF inc2 = 54.9 THEN
inc2:= inc2 - 14.9;
ELSEIF inc2 > -220 THEN
inc2:= inc2 - 20;
ELSEIF inc2 > -230 THEN
inc2:= inc2 - 14.9;
ELSE
TPwrite error3;
GOTO next;
ENDIF
commandnine;

! Stop Command - Return to Home position
ELSEIF message = stop THEN
TPwrite stop;
commandstop;
```

```
! End Command - return to home and exit loop
ELSEIF message = end THEN
TPwrite end;
commandend;
break;

! Higher Command - increase speed
ELSEIF message = higher THEN
TPwrite higher;
commandhigher;

! Lower Command - decrease speed
ELSEIF message = lower THEN
TPwrite lower;
commandlower;

! ERROR
ELSE
TPwrite "error";
ExitCycle;
ENDIF


ENDFOR

ENDPROC


PROC SocketConnect1()

SocketCreate client_socket;

SocketConnect client_socket, "139.86.166.26", 1234 \Time:=1;

ERROR
IF ERRNO = ERR_SOCK_TIMEOUT THEN

IF retry_no < 20 THEN

WaitTime 1;

retry_no := retry_no + 1;

TPwrite ""\Num:=retry_no;
RETRY;
ELSE
RAISE;
ENDIF
ENDIF
ENDPROC



PROC SocketReceive1()

SocketReceive client_socket \Str := message;

SocketClose client_socket;

retry_no := 0;
```

```
ENDPROC


PROC commandone()

MoveAbsj
[[inc,inc1,inc2,inc3,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v10,\V:=speed, fine, tool0;

ENDPROC


PROC commandtwo()

MoveAbsj
[[inc,inc1,inc2,inc3,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v10,\V:=speed, fine, tool0;

ENDPROC


PROC commandthree()

MoveAbsj
[[inc,inc1,inc2,inc3,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v10,\V:=speed, fine, tool0;

ENDPROC


PROC commandfour()

MoveAbsj
[[inc,inc1,inc2,inc3,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v10,\V:=speed, fine, tool0;


ENDPROC


PROC commandfive()

MoveAbsj        [[0,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v10,\V:=speed, fine, tool0;

TPwrite demos;

MoveAbsj      [[179,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v1000, fine, tool0;
MoveAbsj     [[-179,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v1000, fine, tool0;
MoveAbsj [[0,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000,
fine, tool0;
MoveAbsj   [[0,135,-150,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v1000, fine, tool0;
MoveAbsj      [[0,-60,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v1000, fine, tool0;
MoveAbsj [[0,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000,
fine, tool0;
```

```
MoveAbsj       [[0,0,54,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v1000, fine, tool0;
MoveAbsj    [[0,60,-234,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v1000, fine, tool0;
MoveAbsj [[0,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000,
fine, tool0;
MoveAbsj      [[0,0,0,200,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v1000, fine, tool0;
MoveAbsj     [[0,0,0,-200,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v1000, fine, tool0;
MoveAbsj [[0,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v1000,
fine, tool0;

TPwrite demof;
ENDPROC


PROC commandsix()

MoveAbsj
[[inc,inc1,inc2,inc3,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v10,\V:=speed, fine, tool0;

ENDPROC


PROC commandseven()

MoveAbsj
[[inc,inc1,inc2,inc3,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v10,\V:=speed, fine, tool0;

ENDPROC


PROC commandeight()

MoveAbsj
[[inc,inc1,inc2,inc3,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v10,\V:=speed, fine, tool0;

ENDPROC


PROC commandnine()

MoveAbsj
[[inc,inc1,inc2,inc3,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v10,\V:=speed, fine, tool0;

ENDPROC

PROC commandstop()

MoveAbsj        [[0,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v10,\V:=speed, fine, tool0;

! Re-initialise angle variables
inc:= 0;
inc1:= 0;
inc2:= 0;
```

```
inc3:= 0;

ENDPROC


PROC commandend()

MoveAbsj         [[0,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
v10,\V:=speed, fine, tool0;

ENDPROC


PROC commandhigher()

IF speed = 1000 THEN
TPwrite "Speed at maximum";
ELSEIF speed <1000 THEN
speed := speed + 100;
TPwrite "Speed increased to " \Num:=speed ;
ENDIF

ENDPROC


PROC commandlower()

IF speed = 100 THEN
TPwrite "Speed at minimum";
ELSEIF speed >100 THEN
speed := speed - 100;
TPwrite "Speed decreased to " \Num:=speed ;
ENDIF

ENDPROC

ENDMODULE
```

Appendix E

Full code source file contained on submitted CD’s.

Screenshots for Timing results can be found on submitted CD’s.

Simulation video also available on submitted CD’s.