LONDON'S GLOBAL UNIVERSITY

Development and Evaluation of Multichannel Nano-Fibre Piezoelectric Acoustic Transducers: Leveraging Neural Networks and Exploring Novel Structural Fabrications

Amirbahador Moineddini

Supervised by:
Professor Wenhui Song
Dr Hamid Rassoulian

Thesis submitted in partial fulfilment
of the requirements for the degree of
Masters in philosophy
of
University College London

Division of Surgery & Interventional Science
University College London
September 2023

Acknowledgement:

I would like to thank my supervisors Professor Song and Dr. Hamid Rassoulian as well as my peers in the team. Special thanks to Dr. Jinke Chang for his support in completing the project. I would like also to thank the supporting staff at UCL division of Surgery and Interventional sciences at Royal Free Hospital for the training and their support throughout this project.

I would like to extend my gratitude to my friends and colleagues at Division of Surgery at the Royal Free Hospital for their support and time throughout this project and making the past few years very memorable. Special thanks to Patricia Santos, Kelly Bokea, Alex Gray, Christina Christodoulou, Dr Lei Wu and Dr Naheem Yaqub.

Abstract:

This MPhil research project focuses on developing a test platform for the characterization of a novel class of acoustic sensors developed by the Division of Surgery and Interventional Sciences at UCL. Additionally, the project explores fabrication methods and materials to enhance the performance of these sensors. To achieve this objective, an automated test instrument (testbed) was designed and manufactured in the lab for precise and automated data collection. Furthermore, a data analysis algorithm was developed to standardize data collection and automate the testing and analysis of data collected from the devices. The hardware and software for this project were tightly integrated with bioinspired piezoelectric nanocomposite nanofiber-based acoustic sensors, showing promising results for the next generation of self-powered cochlear implants.

The testbed developed in this project serves as a normalized test platform that enables testing various sensors, recording data, and analysing the performance of different iterations of these sensors using a new neural network algorithm for speech and spatial recognition (sound source localisation). The device can automatically collect and process data from multiple sensor channels and train neural networks for testing these devices. These acoustic sensors have been systematically characterized and demonstrated high-frequency selectivity and multifunctional capabilities in speech recognition and localization. This is attributed to the specific geometry of the sensors electrodes and the piezoelectric properties of highly aligned radially polymer nanofibers made of poly(vinylidene fluoride-co-trifluoroethylene) (PVDF-TrFE) doped with Barium Titanate ($BaTiO_3$) nanoparticles.

Moreover, using the testbed, it was demonstrated that a single multichannel asymmetrical device could localize sound sources, in contrast to human hearing, which requires both ears for localization. This attribute results from the asymmetrical

nature of the sensor combined with the design of the neural network used for the sensors.

Additionally, this report proposes a new method of fabricating the sensors using Electrohydrodynamic direct printing instead of the electrospinning process currently used. This method offers more control over the diameter, orientation, and placement of the fibres on the electrode, making the fabrication process slower but more repeatable. Finally, the report discusses how the fibre structure could be modified using coaxial printing to improve the endurance and performance of the sensors.

Contents

List of Figures

List of Tables

Abbreviations:

ITD	Interaural Time Difference
ILD	Interaural Level Difference
MSO	Medical Superior Olive
HRTF	Head-Related Transfer Function
PZT	Lead Zirconate Titanate
PVDF	poly(vinylidene fluoride)
GaN	Gallium Nitride
PVDF-TrFE	poly(vinylidene fluoride-co-trifluoroethylene)
CI	Cochlear Implant
BTO	Barium Titanate-$BaTiO_3$
CNN	Convolutional Neural Network
EHDP	Electrohydrodynamic Printing

Chapter 1

1. Introduction

1.1. Background

This research focuses on designing an instrument to test the performance and capability of a new bioinspired cochlear implant. The new device uses composite piezo nano-fibres to fabricate a new class of self-powered cochlear implant, which rethinks the architecture of the implant. The device offers a higher frequency selectivity than the current generation of implants available to patients suffering from extreme hearing loss either due to birth defects or serious injuries to the inner ear. This research will also take a brief look a more suitable fabrication method for depositing the fibres on the electrode and for controlling the composite structure of the fibre.

It is crucial for scientists to continue researching and developing new cochlear implants around the world because while these implants improve the lives of countless people around the world and they are still far from perfect. These devices are implanted in adults and children with severe hearing loss who will not see any benefit from using hearing aid [1]. Moreover, the damage to the inner ear and hearing loss which lead to the need for cochlear implant could cause severe tinnitus and the cochlear implants could significantly reduce the severity of the tinnitus and improve quality of life for users. [2]

British Cochlear Implant Group (BCIG) has been publishing a yearly report for the past 12 years (excluding 2012-2013) where they collect all the cochlear implant operations in Britain between the 1st of April of each year to the 31st of March of the next year. In the report of the implants between April of 2021 to March of 2022 claims that 868 new adults and 440 new children received cochlear implants across Britain [3]. The chart in Figure 1 summarises all the operations carried across 10 years sorting

them by the type of operation. As shown in the graph, there has been an increase in the number of operations each year until post 2020 where the number of operations were drastically reduced due to the COVID 19 pandemic.

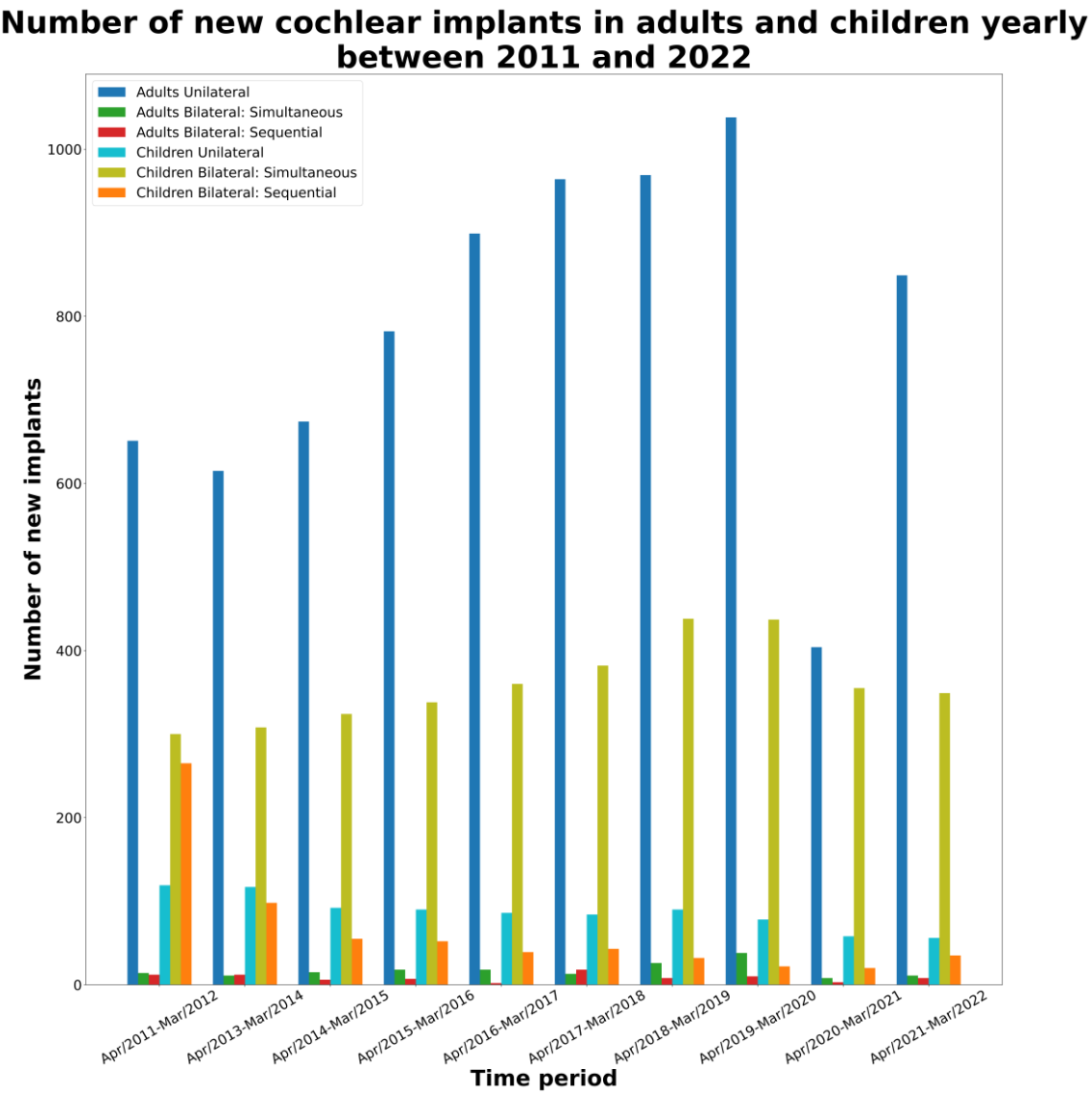


Figure 1: Illustrates the number of people who have received a surgery to get cochlear implant over 10 years. The database split the data by the age group "(Some centres may class under 19 as 'children'; others may use under 18)" and whether they received unilateral operation, simultaneous bilateral and sequential bilateral operation. The data shows that there is an overall increasing in cochlear implant operations despite of the delays and cancelation caused by the pandemic. [4]

US National Institute of Deafness and Other Communication Disorders (NIDCD) estimates that there are approximately 737,000 registered devices worldwide by the

end of 2019 and the worldwide cochlear implant market was valued at USD 1.5 billion in 2021 and expected to expand at a compounding annual growth rate of 8.71% from 2022 to 2030 [5] [6]. The enormous number of patients in need of this life changing implant, the market value and its potential growth has incentivised researchers around the world to work on making strides in development of these implants over past decades since the first single channel implant in 1972.

The current implants consist of the following main surgically implanted internal components and removable external components that need to be recharged: 1) An external microphone to pick up the sound, 2) An external speech processor to process the sound 3) An external Transmitter which is magnetically attached to an internal Receiver to transfer data from external component to the electrodes, 4) An electrode array that sits inside the cochlear to stimulate different regions of the auditory nerves directly (Figure 2). Moreover, it is important to note that the current cochlear implants do not cure deafness or hearing impairment and current implants can only give a representation of the sounds in the environment to the user; Majority of users, especially the unilateral implant users, lose directionality recognition of the sound. Additionally, all users have to spend months getting used to hearing the implant and interpreting the information and they will suffer from a condition known as the cafeteria or restaurant effect where they cannot focus on a single source of audio and targeted audio will drown in the background. [1] [7]

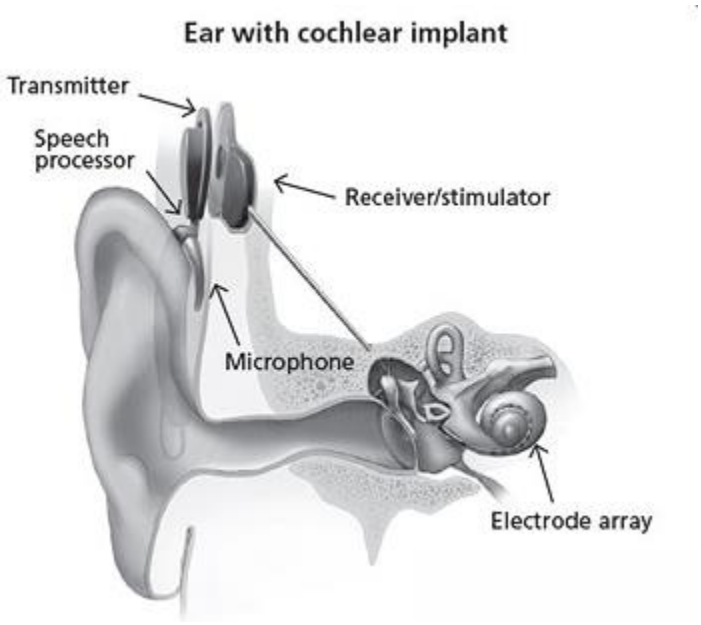


Figure 2: current cochlear implant design illustration. The microphone, speech processor and transmitter comprise of the external detachable parts which is connected to the internal implanted receiver and electrode array with a magnet.[7]

This is mostly because the cochlear implant electrode array bypasses the damaged hair cells and directly stimulates the Spiral Ganglion Neurons (SGNs). While the array has over a hundred ~~of~~ frequency channels, it can only be optimised and tuned for a few dozen channels and frequencies in the cochlear due to poor localisation and control over targeting in the cochlear during surgery, which leads to less intelligible signal and only the presentation of the sound. Consequently, cochlear implant users are unable to enjoy tunes and music [5] [8] [9]. Nevertheless, There have been major developments in all the components of cochlear implants such as increasing both the frequency channels between the implant and the SGN and the number of channels in the array by reducing the size of the electrodes in it, advances in signal processing algorithms and taking advantage of deep neural networks to reduce transient noise, which can be personalised depending on the user preferences, better transmitter and receiver interface to reduce the transmission noise. [10].

Finally, all cochlear implants are battery-powered and either use rechargeable batteries which last between 19 to 40 hours depending on the size of the battery or use disposable ones which last up to about 48 hours. Given all the limitations of the

current cochlear implants mentioned, the motivation behind this project is to take advantage of the piezo fibres to create a new generation of powerless cochlear implants inspired by cochlear itself to address all the issues.

1.2. Bilateral Hearing in humans

Bilateral hearing refers to the ability of humans (and many other species) to hear with both ears. This binaural (two-eared) hearing provides several significant advantages that enhance auditory perception [11]:

1. Localization: [12] [13]One of the most prominent benefits of bilateral hearing is sound localization. This refers to the ability to determine the direction from which a sound originates. Differences in time and intensity with which a sound reaches each ear are used to locate the sound source.
    a. Interaural Time Differences (ITD):
        - Sound waves from a source located to one side of the head will reach the nearest ear slightly before they reach the farthest ear.
        - The brain can detect these tiny differences in arrival times, using them primarily for horizontal sound localization.
        - The superior olivary complex in the brainstem plays a pivotal role in processing ITDs.
    b. Interaural Level Differences (ILD):
        - Apart from the time difference, sounds coming from one side will be slightly louder in the nearer ear than the farther ear due to the "shadowing" effect of the head.
        - ILDs are mainly used for sounds at higher frequencies, where the head is more effective at blocking or attenuating the sound.
        - ILD processing also takes place in the superior olivary complex.
    c. Coincidence Detection:
        - In the auditory system, particularly in the medial superior olive (MSO) of the brainstem, there are neurons that act as coincidence detectors.

- These neurons fire when they receive simultaneous input from both ears, playing a key role in detecting ITDs.

d. Spectral Cues:
   - The shape and size of the outer ear (pinna) introduce frequency-dependent alterations to incoming sounds, which the brain uses to help determine the elevation of a sound source.
   - This is especially useful for vertical sound localization.

e. Head-Related Transfer Function (HRTF):
   - Each individual has a unique HRTF, which is the way sounds are filtered by the body, head, and outer ear.
   - The brain learns to recognize these unique filters and uses them to aid in sound localization.

2. Central Auditory Processing:
   a. Better Hearing in Noise: Once the auditory nerve has relayed information to the brainstem, a series of complex processes, particularly in the superior olivary complex and later in the auditory cortex, further refines sound location and perception. These processes help in distinguishing sounds, understanding speech in noisy environments, and more.
   b. Improved Sound Quality: Sounds are perceived as richer and fuller when heard through both ears due to the combined auditory input.
   c. Increased Loudness: A phenomenon called "binaural summation" means that sounds are perceived as louder when heard by both ears as opposed to just one.
3. Redundancy and Fusion:
   - Binaural redundancy means that each ear sends a version of the same acoustic signal to the brain. This redundancy can enhance the signal-to-noise ratio and improve detection in noisy environments.
   - Binaural fusion refers to the brain's ability to integrate information from both ears into a single perceptual image. With two ears, if one misses some

elements of a sound due to transient noise or other interference, the other might still pick it up, ensuring a more consistent auditory experience.

4. Plasticity:
    - The auditory system can adapt to changes over time, especially with experiences or hearing loss. This plasticity ensures that the brain continues to effectively process sounds even under varying conditions.

1.3. What are piezoelectric material

Piezoelectric materials are a class of materials that have the capacity to either produce an electric charge in response to mechanical stress (the direct piezoelectric effect) or change shape in response to an applied electric field (the inverse piezoelectric effect). Due to this exceptional quality, they are useful in a variety of applications, including sensors, actuators, energy harvesting, and more. Some common piezoelectric materials include: [14]

1. Quartz - This is one of the most well-known and widely used piezoelectric materials. It is a crystalline form of silica and is often used in various electronic components.
2. Lead Zirconate Titanate (PZT) - PZT is a ceramic material that exhibits strong piezoelectric properties. It is commonly used in sensors, actuators, and transducers.
3. Polyvinylidene fluoride (PVDF) - This is a polymer with piezoelectric properties. It is flexible and can be used in a variety of applications, including flexible sensors.
4. Gallium Nitride (GaN) - GaN is a semiconductor material that also exhibits piezoelectric properties. It is often used in high-frequency electronic devices.

Table 1: Compares different piezo materials structure, density, piezoelectric coefficient Dielectric constant and offer some of the applications in the industry and research for each material.

Property	PVDF (α-phase)	PVDF (β-phase)	PZT	Quartz	Gallium
Chemical Formula	(C2H2F2)n	(C2H2F2)n	Pb(Zr,Ti)O3	SiO2	GaN
Crystal Structure	Semi-crystalline polymer Alternating trans-gauche conformation (TGTG)	Semi-crystalline polymer All Trans conformation (TTTT)	Perovskite	Trigonal	Wurtzite
Density (g/cm^3)	1.78	1.78	7.8-8.0	2.65	6.1
Piezoelectric Coefficient (d33) (pC/N)	8-10	20-30	200-600	2.3-2.5	3.1-3.3
Dielectric Constant	8-10	12	1200-1700	4.5-5.5	8.9
Applications	Low-performance sensors, packaging materials	High-performance sensors, actuators, energy harvesting	Medical devices, actuators, sensors	Oscillators, frequency standards	High-frequency electronics, LEDs

Piezoelectric materials have found various applications in the field of medicine because of these effects and used across many industries. Piezoelectric transducers are at the heart of ultrasound imaging systems. When an electric voltage is applied to these transducers, they generate ultrasonic waves. These waves travel through tissues and bounce back, creating echoes that are detected by the same transducer. These

echoes are then used to create an image of the internal structures of the body. Common piezoelectric materials used in ultrasound transducers include lead zirconate titanate (PZT) and polyvinylidene fluoride (PVDF) [15] [16]. One of the most common applications is in energy harvesting. Piezoelectric materials can convert mechanical vibrations or deformations into electrical energy. This is particularly useful in environments where there is ambient mechanical energy available.

The evolution of material research in the realm of biomedical applications has been marked by continuous advancements, and the quest to improve cochlear implant (CI) technology is a testament to this journey. There is an emergence of groundbreaking research in CI takes advantage of the piezo materials and their properties that challenges traditional CI operational principles. For example, a novel fully implantable thin film piezoelectric transducer uses a cantilever-based design PLD-PZT transducer. The choice of PLD-PZT as a material demonstrates a remarkable intersection of material science and biomedical engineering, achieving an unprecedented voltage output of 114 mV under conditions mirroring the eardrum's natural behaviour. Beyond setting new benchmarks in thin film piezoelectric transducers, this feat illuminates the immense potential of fine-tuning material properties to foster next-generation biomedical tools. The meticulously designed multi-frequency acoustic sensor, encompassing eight distinct cantilever beams, underscores the power of material optimization. Each beam, tailored to resonate at specific frequencies within the human acoustic range, exemplifies the synergy of material science and design. The prototype's compactness and sensitivity, made possible by material advancements, present a pioneering solution to challenges in fully implantable cochlear implant (FICI) applications. This research not only underscores the transformative role of materials in reshaping cochlear implantation but also charts a blueprint for leveraging material innovations in devising advanced, fully implantable biomedical devices. [17] [18] [19]

Polyvinylidene fluoride (PVDF) is a piezoelectric polymer that has gained attention for its use in various applications, including piezoelectric fibres. PVDF piezoelectric

fibres are known for their flexibility, lightweight nature, and ease of integration into a wide range of devices and structures.

PVDF, specifically the β -phase PVDF, is a popular choice for piezoelectric fibres due to its unique properties:

1. Flexibility: PVDF piezo fibres are highly flexible, making them suitable for applications where conformability to complex shapes or structures is required.
2. Lightweight: PVDF is a lightweight material, which is advantageous in applications where weight is a critical factor, such as wearable devices.
3. Biocompatibility: PVDF is biocompatible, meaning it can be used in medical applications without causing adverse reactions in the body.
4. Low Density: PVDF has a low density, which makes it an excellent choice for applications where weight reduction is important.

One of the key applications of PVDF piezo fibres is in the development of flexible and conformable sensors for various purposes, including biomedical, structural health monitoring, and wearable technology. [20] [21]

In the continuous endeavour to improve cochlear implant (CI) technology, a groundbreaking approach has emerged that challenges traditional CIs, using the mentioned properties of the piezoelectric nanofibres to create a bio-inspired, highly frequency selective and self-powered implants.

1.4. What are Convolutional Neural Networks (CNNs) and their use cases in for pattern recognition in acoustic sensing:

To test the devices for localization, traditional methods of data collection and analysis are insufficient because the variations from different angles are minuscule and undetectable with conventional mathematical models. Therefore, a more powerful tool is required to examine each data point and identify differences in data from various directions. In recent years, researchers globally have been exploring a myriad of machine learning methodologies with the objective of expediting data processing

and bolstering pattern detection within datasets. This endeavour aims to elucidate the intricate relationships among an expanding set of variables in sophisticated systems and experiments. Notably, deep learning has emerged as one of the most efficacious and flexible instruments within the machine learning domain. Neural networks are computational models inspired by the brain's neural structures, designed to recognize patterns by processing data through interconnected layers of artificial neurons, enabling tasks such as classification, regression, and clustering in diverse domains.

A *Convolutional Neural Network* (CNN) is a type of deep neural network designed for processing structured grid data, such as images. It is specifically engineered to recognize spatial hierarchies in data, which traditional multilayer perceptrons (MLPs) might struggle with due to their fully connected nature. CNNs operate by using convolutional layers to extract the spatial hierarchies of features automatically and adaptively from input data. These features, increasing in complexity across layers, are then processed through pooling and fully connected layers to make final predictions, facilitating tasks like image classification and object detection with remarkable efficiency and accuracy. [22]

The adaptability of CNNs stems from its uniquely crafted layers tailored for data extraction from given inputs. A brief overview of these layers follows: [23]

- Convolutional Layer: The cornerstone of CNNs. It employs a mathematical operation called convolution to slide a filter (or kernel) over input data (like an image) to produce a feature map, capturing spatial hierarchies and patterns in the input data.
- Pooling Layer: Used to reduce spatial dimensions while retaining significant information. The most common pooling operation is max pooling, where the maximum value from a group of values is chosen.
- Fully Connected Layer: In the end, after multiple convolutional and pooling layers, CNNs often use one or more fully connected layers to classify the extracted features into various categories or make other determinations.

- Activation Function: activation function introduces non-linearity to the model, enabling it to learn and represent more complex relationships in the data. It determines the output of an artificial neuron given a set of inputs, effectively deciding whether a particular neuron should be "activated" or not. ReLU is one of the most popular activation functions which replaces any negative values with zeros, allowing only positive values to pass through.
- Dropout: A regularization technique used in CNNs to prevent overfitting. It randomly sets a fraction of input units to zero at each update during training time.

Using Convolutional Neural Networks (CNNs) for classifying data from acoustic signals is an evolving area of research. Acoustic signals, whether they are speech, music, or environmental sounds, are typically waveforms that change over time. When represented appropriately, these waveforms can be viewed as 1D (time domain) or 2D (time-frequency domain) data, making them amenable to CNN-based approaches. CNNs already have many applications in classification and pattern recognition in acoustics. Here are some of the notable application areas where CNNs are being used: [24] [25]

1. Speech Recognition: Distinguishing spoken words or phrases.
2. Speech Emotion Recognition: Identifying emotional content in speech.
3. Environmental Sound Classification: Recognizing sounds like rain, traffic, or birds chirping.
4. Music Genre Classification: Categorizing music tracks by genre.
5. Bioacoustics Signal Classification: Analysing animal calls or underwater marine sounds.

1.5. Aim, Hypothesis and Objectives

The aim of this research is to create a new class of cochlear implants that is inspired by the human cochlear to capture and deliver more audio information to the user than

the current commercially available implants. The implant should take advantage of the piezoelectric polymeric nanofibers to be self-powered and remove the need for battery and recharging.

From the response of the devices, it is hypothesised that the asymmetrical spiral device could be used for sound localisation on it own and either one of the symmetrical circular and asymmetrical spiral devices capture all the information needed for speech recognition. To prove this hypothesis, a rotary testbed was designed to that enables to normalize and collect data from different devices at all directions to train a convolution neural network to localise and recognise simple phrases.

It is hypothesised that, by improving the current design and adding more electrodes, more audio information can be captured, so as to improve on the spatial recognition of the current generation of the device. Another hypothesis is that direct laying the fibres with higher control of the length and placement of the individual fibre on the electrode will improve the higher accuracy and the resolution of the response of the sensor.

Over the subsequent five chapters, a detailed exploration of strategies to address the specified challenges will be presented. This exploration begins with the design, instrumentation, and integration of a test platform intended to assess each device iteration for spatial and speech recognition. Subsequent sections delve into novel fabrication techniques and materials. The project culminates with an examination of the device design's influence on frequency selectivity, cellular toxicity evaluations, and in vitro studies assessing neuronal responses to the device.

Chapter 2

2. Design of the bio-inspired Implants

2.1. Background:

"Neurons at various levels in the auditory pathway are topographically arranged by their response to different frequencies. This organization, referred to as tonotopy or cochleotopy, mirrors the distribution of receptors in the cochlea, with a gradient extending between neurons that preferentially respond to high frequencies and those that respond best to low frequencies" [26].

As shown in Figure 3 Human cochlear has an auditory range between 20Hz to 20.5kHz. The current device that we are working on is most sensitive to frequencies between 80 to 500Hz due to the low stiffness of the piezoelectric fibres, which is the range of frequency that current clinical cochlear implants fail to respond. However, not all frequencies are the same and humans are more sensitive to some frequencies than others. Figure 3 illustrates a typical adult human audible range and the frequency and intensity range that a typical conversation or music uses. This means that the current device will not be able to capture the desirable audio information in a wider range of higher frequency, which results in low intelligibility. This was clear when the data collected for the speech recognition were converted back to audio. Despite that the signal still contained rich information, it was difficult to interpret the original phrase used in the speech recognition from the audio file. Therefore, to expand the range of frequency, the current design requires further improvement or re-design so

that the device can cover higher frequencies as well as by development hybrid and stiffer piezoelectric composite nanofibres.

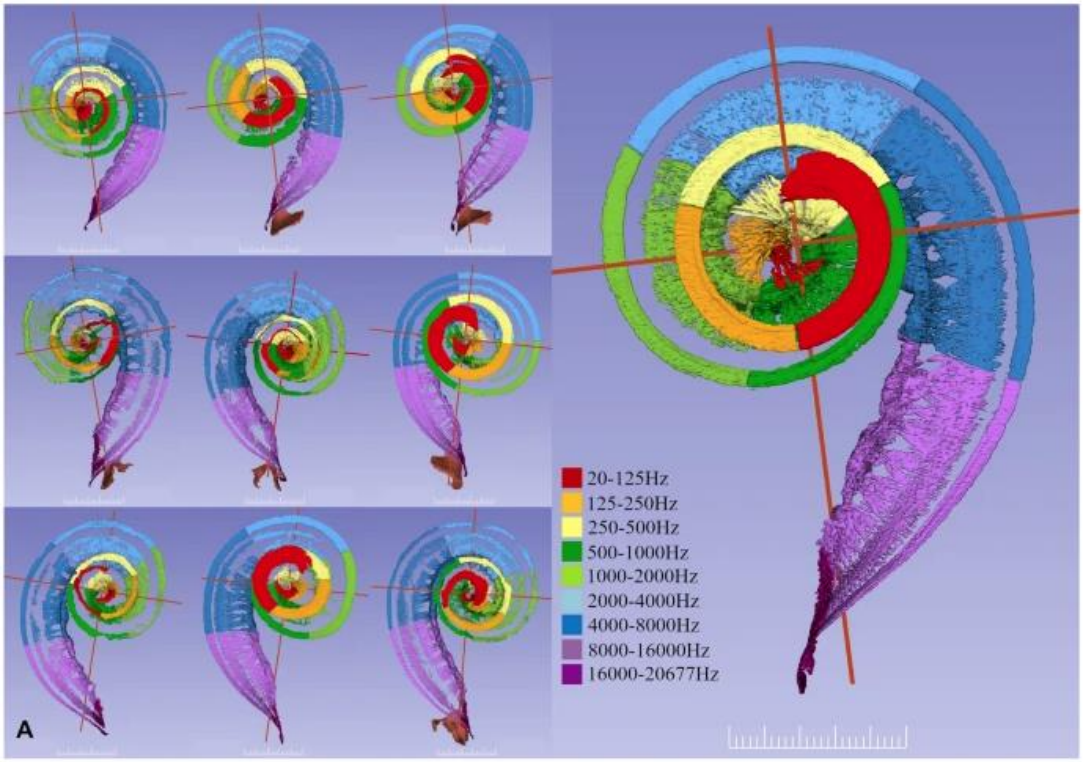


Figure 3: 3D Tonotopic mapping of the human cochlear using Synchrotron radiation phase-contrast imaging (SR-PCI) [9]. The Cochlear can cover the frequencies from 20Hz to 20.1kHz. Using the scale provided, the size of the cochlear is about 6.02 in 9.2mm.

To overcome this issue, a new nanocomposite fibre must be designed and a device that resonate with a wider range of frequencies so that the maximum amplitude of the output signal corresponds to the larger range of the audible frequency range that we use in our day-to-day life. By controlling the length, diameter and elasticity of the fibres, we could control the resonant frequency of the fibres in a wider range.

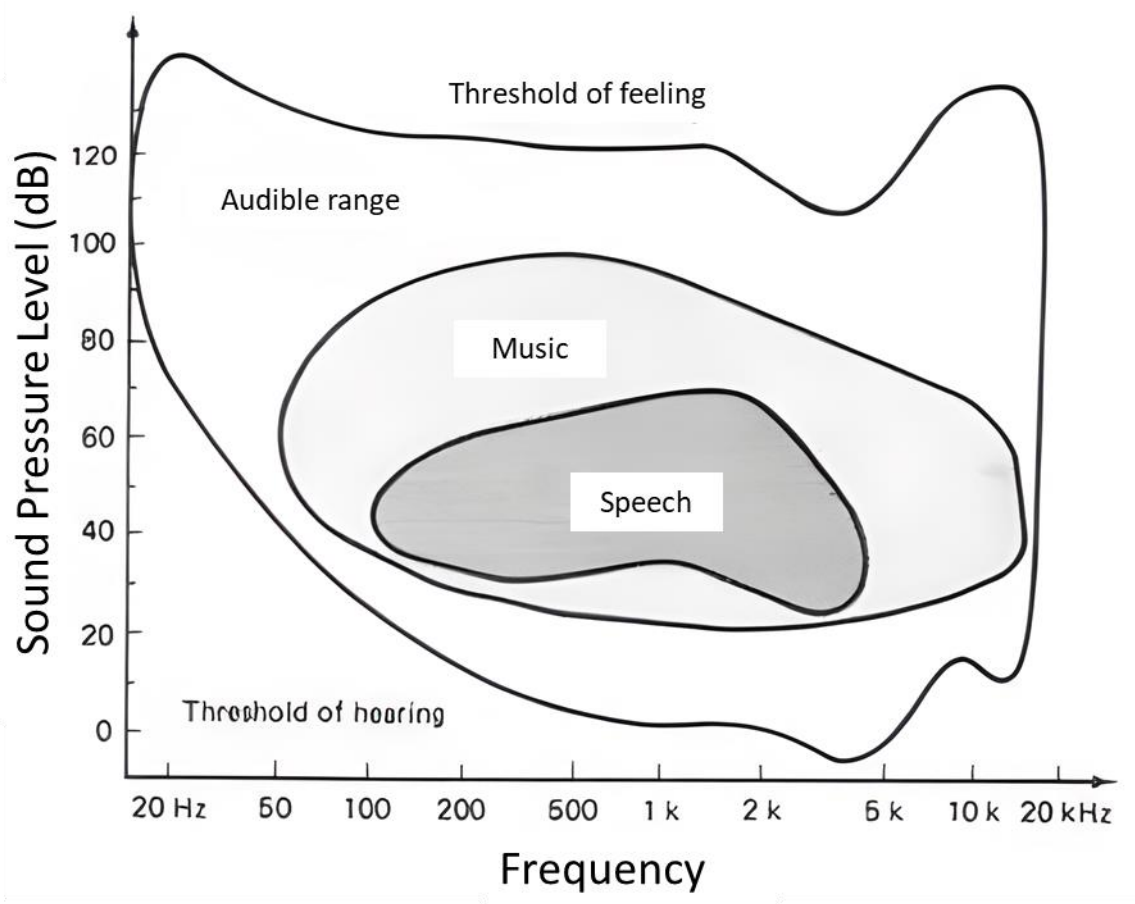


Figure 4: Average human audiogram and audible range for different frequencies and the region used for listening to music and a normal conversation [27] [28] [29]. The figure illustrates that while the range of audible frequency is between 20 to 20kHz, the intensity (sound pressure level) is variable and cochlear covers a wider range of sound pressure between the frequencies of 75 to 10kHz and is most sensitive to the frequencies between 100 to 3kHz which is the encapsulates the speech and music frequency range.

2.2. Design of the device:

This work is the improvement over the previous research carried out by Professor Song group in PVDF piezoelectric nano fibres [30] with collaboration with Dr. Jinke Chang to design the device. The hypothesis of research carried out by the group was that a device with different size channels and consequently different length of fibres between these channels will different natural frequencies. Therefore, a device with multiple channels with different dimeter (similar to the spiral structure of the

cochlear illustrated in Figure 3) will have multiple natural frequencies with that can be studied and design to match the human cochlear. To test hypotheses, two device was tested with using the vibrometer setup to measure the voltage output of each channel of an asymmetrical spiral multichannel and a symmetrical circular multichannel device in response to a sweep of frequency between 2500 to 100Hz at 1dB played by a mouth simulator perpendicular to the device from the distance of 5cm with sampling frequency of 20kHz.

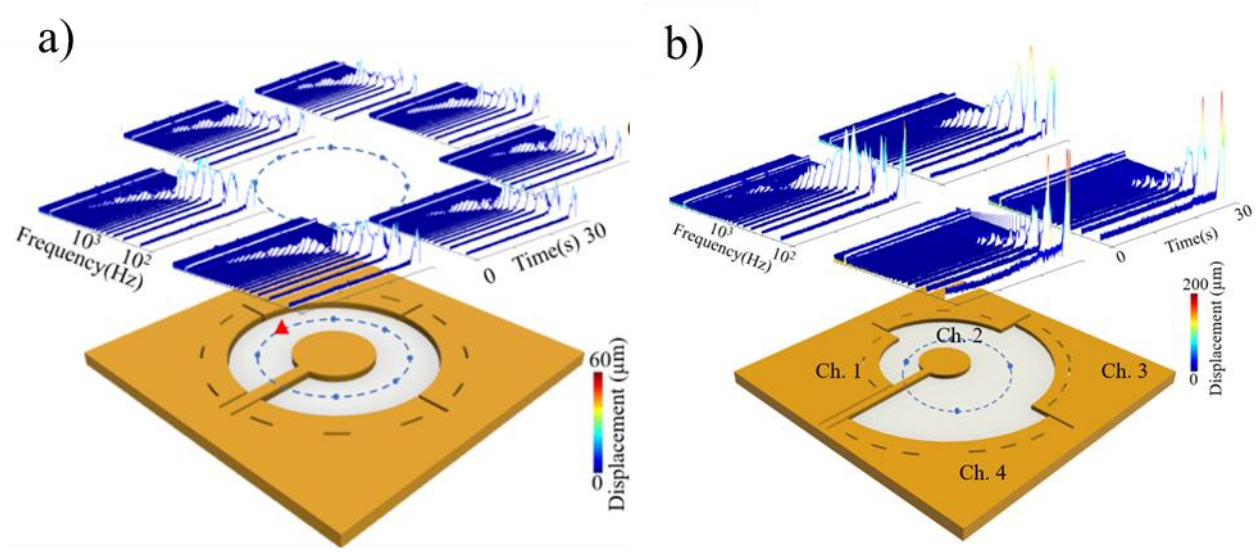


Figure 5: The design and the STFT graph of the frequency response of the devices to different sound frequency generated by the mouth simulator. a) shows the frequency response of the 7 channels on the symmetrical circular device and b) shows the frequency response of the 4 channels asymmetrical spiral device.

As shown in Figure 5, while the voltage output of the symmetrical circular device measured at 7 different positions along the sensor has very similar output, the voltage output of each channel of the asymmetrical device to the same stimuli under the same condition is different. Figure 6 shows the displace of each channel over time and voltage output of each channel over times under the above conditions.

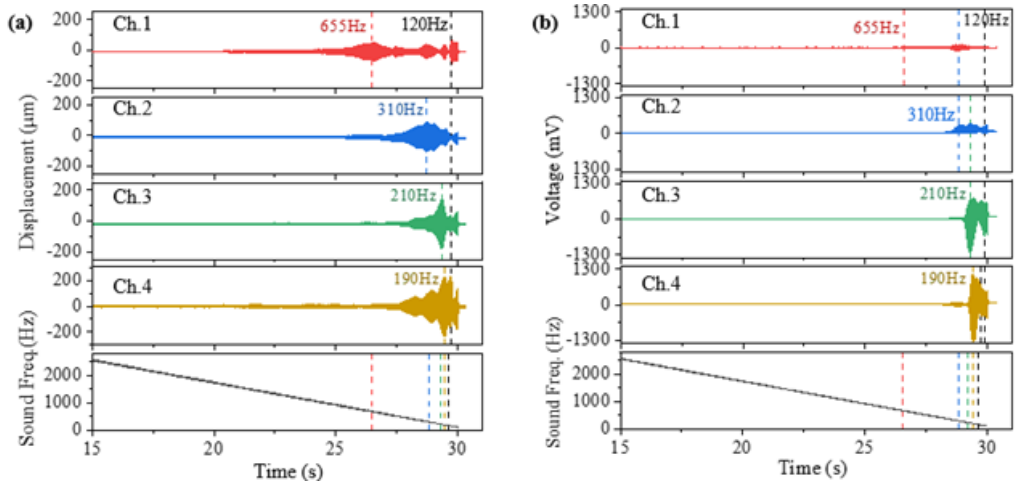


Figure 6: shows the measurement of the fibre displacement of each channel measured by the laser vibrometer and the voltage output of each channel of the asymmetrical spiral device in response to the frequency sweep by the mouth simulator. Provided by Dr Chang.

2.3. Fabrication of the devices:

The devices were fabricated by Dr. Jinke Chang using electrospinning. Electrospinning employs high voltage to electrify liquid droplets, generating a jet between the high-voltage needle extruding the liquid and the grounded electrode. The liquid droplets are provided to the system by gradually feeding the polymer solution through a syringe needle using a syringe infusion pump, which allows control over the speed of injecting the solution into the system. In this case, the PVDF-TrFE polymer, along with the BTO nanoparticles, was dissolved in a mixture of DMF and acetone[30]. To create PVDF-TrFE nanofibers, DMF is the primary solvent used to dissolve the PVDF, while acetone is added to accelerate the evaporation of the solvent, ensuring that the liquid droplets form fibres. If there is insufficient acetone in the solvent, the liquid droplets will not form a jet, causing polymer droplets to fall instead. Conversely, if there is an excessive amount of acetone in the solution, the polymer will be overly diluted, and insufficiently aligned fibres will be formed.

To get the fibres highly aligned radially on the electrode, all the terminals on the electrodes were grounded to a rotary stage in the electrospinner while the needle was attached to a high voltage source on a linear moving stage as shown in Figure 7. Both

the needle and electrodes were moving together simultaneously to make sure that the fibres are radially aligned.

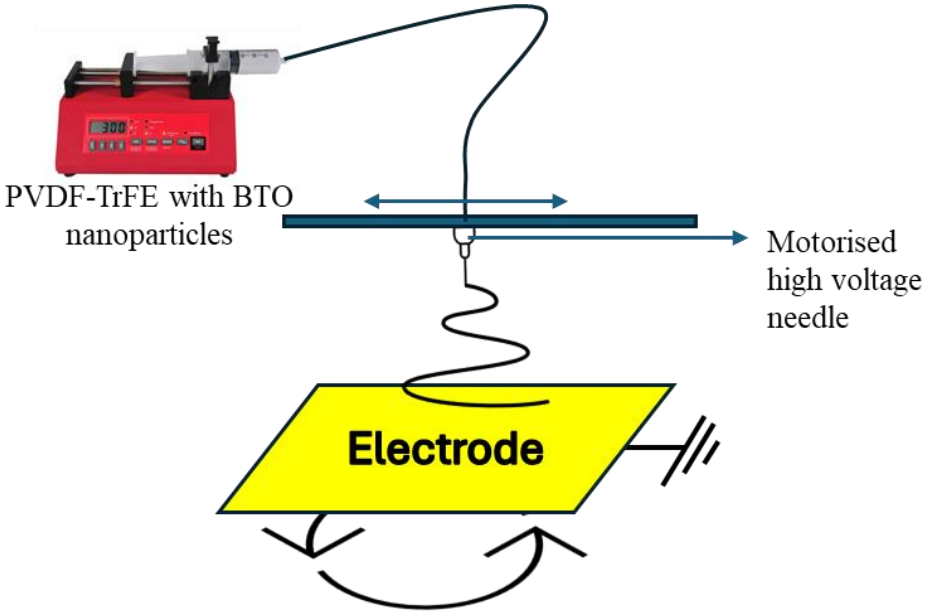


Figure 7: The fabrication stages piezoelectric sensor devices using electrospinning. The electrode is grounded and rotating around while simultaneously the high voltage needle moves side to side extruding the PVDF nanofibers.

Finally, the sensor fabricated by assembling the electrospun electrode with a mirror design electrode on top with a PET film in the middle to prevent any short circuit between the two electrodes. In the experiments, the voltage produced between the 2 electrodes is measures which is produced by the vibration of the PVDF piezoelectric fibres.

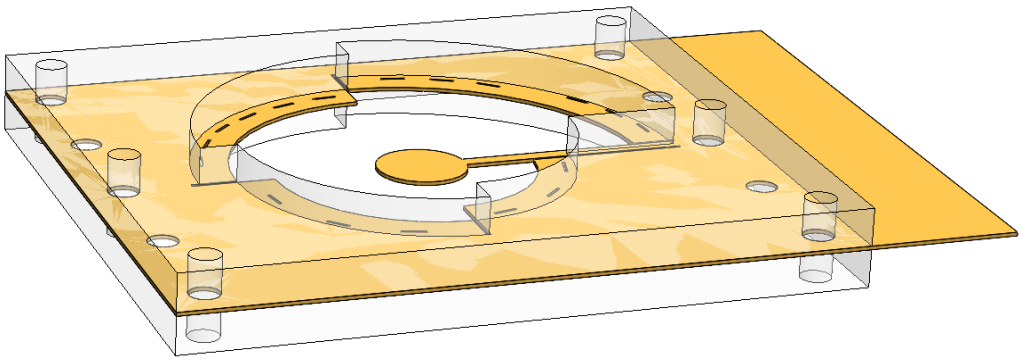


Figure 8: The final assembly of the device. The device was clamped between two acrylic cut-outs of the sensor.

As illustrated in Figure 8, the device is assembled between to acrylic parts permanently to make sure the clamping force holding the assembly together remains constant between trials. This was done as the initial testing showed that depending on clamping force holding the whole assembly together in the test bench of the vibrometer, the response of the device and voltage output changes. More details about the structure of the device and assembling is currently protected for a patent application.

Chapter 3

3. Design, instrumentation, integration and data collection of a multi-channel cantilever acoustic device

3.1. Background

In the lab, a number of bio-inspired piezoelectric acoustic sensors were designed and manufactured by electrospinning 6wt% Barium Titanate Poly(vinylidene fluoride-trifluoroethylene) (BTO/PVDF-TrFE) nanocomposite fibres on single-channel symmetrical circular (SSC) and multi-channel asymmetrical spiral radial electrodes (MAS) (Figure 9) through an EPSRC funded project. Both devices could be used in a single and multi-channel configuration, for the speech and spatial recognition testing, however, analysis of a large quantity of output data including voltage output of each device under various acoustic signals was challenging, in particular, the complexity and amount of data for multi-channel MAS devices were substantially increased.

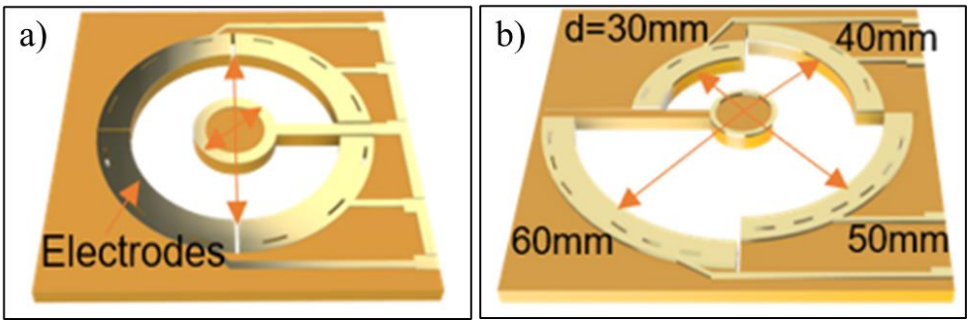


Figure 9: a) Symmetrical circular sensor b) Asymmetrical spiral sensor.

3.2. Aims and Objectives

The aim of this chapter is to develop a robotic rig for data collection and deep learning method to analyse the complex voltage signals of the devices in response to various acoustic stimulation.

To achieve the objectives set for this project, it was broken down into 5 sections starting with design and manufacturing an instrument to test the spatial recognition (localisation) using the MAS and speech recognition using both MAS and SSC by taking advantage of a deep convolutional neural network trained and tested on the voltage output from the devices that has already been fabricated in the lab for an initial benchmark and testing any future devices as mean of standardising them and evaluating their performance against each other. The initial testing for spatial recognition was carried out using the laser vibrometer setup and the mouth simulator with a modified sensor holder that allowed for tracking of the angle change for data collection.

The hypothesis for these devices is that the well aligned piezoelectric nanofibres between electrodes can produce large enough voltage consistently when stimulated by acoustic signals. Thus, the devices can be used for speech recognition. The most intriguing novelty of the multi-channel spiral device lies that, due to its asymmetrical design, the complex signals captured from such a device indicated the relative position of the stimuli source for spatial recognition. To quantify and prove the concept and performance of the device, for the first part of the project, the focus was on developing a platform that enable to collect voltage output produced by the sensors for the duration of experiments by various stimuli, such as mouth simulators and commercial speakers, while simultaneously controlling the exact position of the acoustic stimuli source relative to the sensor. Furthermore, a number of CNNs with different architectures were designed and created for analysis of the voltage data in order to determine the accuracy and precision of the spatial recognition of the MAS device and the speech recognition of the MAS and SSC device.

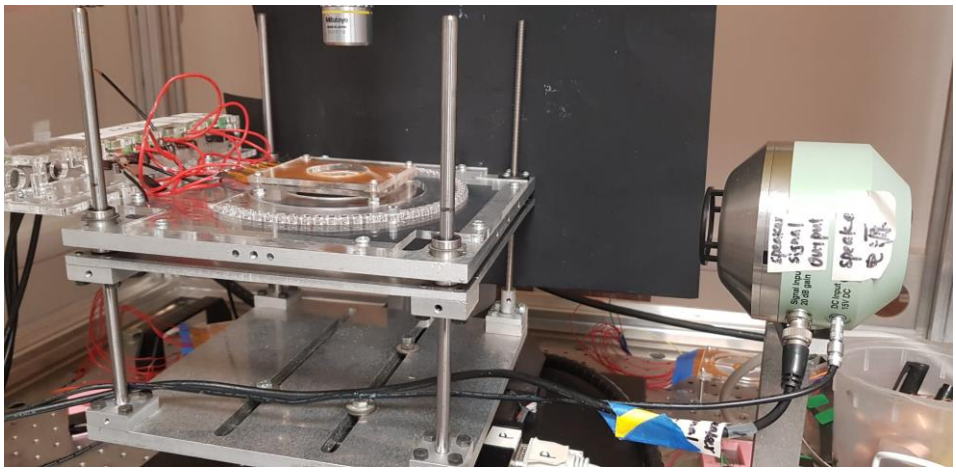


Figure 10: Initial data collection set up for directional recognition testing. The speaker is stationary and a set of holes on the sensor holder are used to place the sensor at different angles and the data are collected.

3.3. Preliminary spatial recognition testing using CNN with spiral Asymmetrical multichannel device to check the feasibility.

During initial tests, both device configurations exhibited potential for speech recognition when visually comparing the waveform of the signals produced with those from a conventional off-the-shelf microphone. Demonstrating these devices' capability to pinpoint a sound source necessitated a comprehensive experiment, given the challenge of discerning sound waveforms originating from various directions by visual inspection alone and show the merit of the experiment. Thus, prior to allocating resources to manufacture a testbed for spatial recognition, preliminary tests were conducted to validate the hypothesis that the MAS device can be used for spatial recognition to its asymmetry. This involved integrating an angle placement extension into the current sensor test stage (Figure 10) and manually gathering data from varied sensor configurations in multiple directions (at every 30°). Data collected from the spiral asymmetrical multichannel device enabled the training of a CNN that efficiently extracted features and patterns, achieving precise sound source localization. Conversely, several CNN algorithms applied to the symmetrical device data for spatial recognition proved unsuccessful because none of the CNNs trained on the asymmetrical device was unable to classify any of the test dataset correctly.

Given the positive outcomes of the experiment, the construction of a robotic rig was pursued to autonomously evaluate sensors under varied configurations, with an eye towards future device optimization. Section 2.3.1 is dedicated to the initial manual evaluations conducted on the spiral asymmetrical multichannel device for spatial recognition, along with a discussion of the findings. The code used for this section is listed in Appendix A.

3.3.1 Methodology

1) data collection and preprocessing

Initially, data were collected from each of the four channels 100 times concurrently at intervals of both 45° and 30°. Utilizing the frequency data recorded by the laser vibrometer along with the raw data, the data from each channel was segmented into 100 distinct datasets for evaluation.

Raw voltage-time series recording of the four channels response to the sweep of 1500Hz to 80Hz spitted into 100 distinct data points.

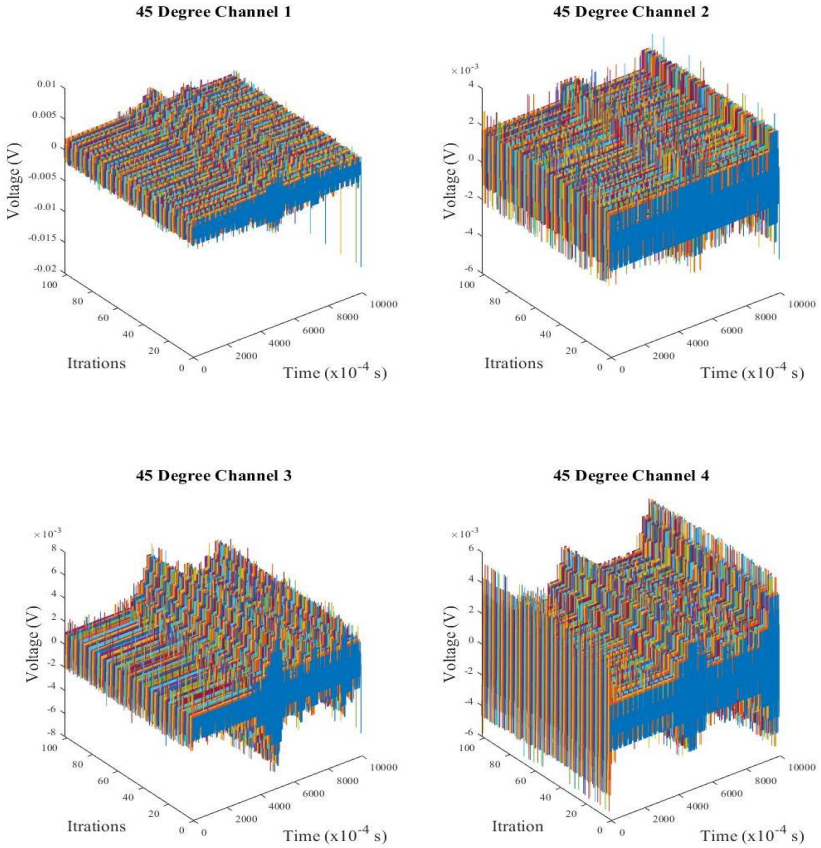


Figure 11: demonstrates raw data collected at 45º and split into individual sweep response reading by the sensors.

Acknowledging the parameters and constraints of the experiment and using the Short Time Fourier Transformation (STFT) of the raw data, a bandwidth Butterworth digital filter was constructed and subsequently applied to the signal, which was then detrended during the preprocessing phase of the experiment.

STFT plot of individual channels of a single reading recorded at 45º

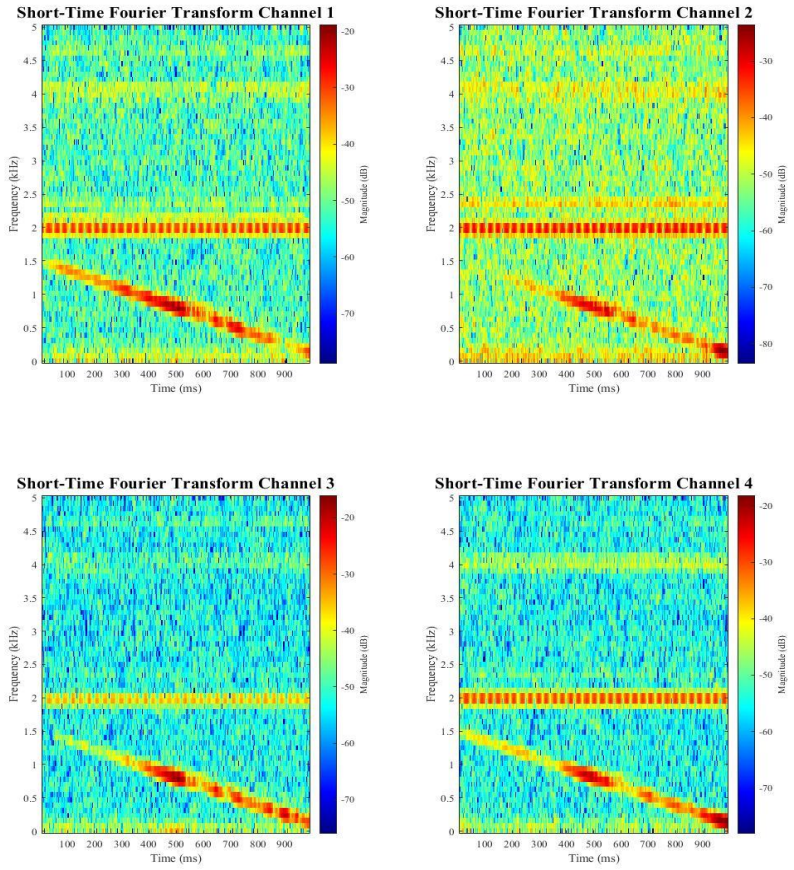


Figure 12: illustrates the STFT of plot of an individual at reading at 45º obtained from the voltage output of the device in response to a sweep from 1.5kHz to 80Hz over a second at 1dB. This plot is used for creating a bandwidth filter to reduce the noise and highlight the difference in output of different devices.

Preprocessed voltage-time series recording of the four channels response to the sweep of 1500Hz to 80Hz spitted into 100 distinct data points.

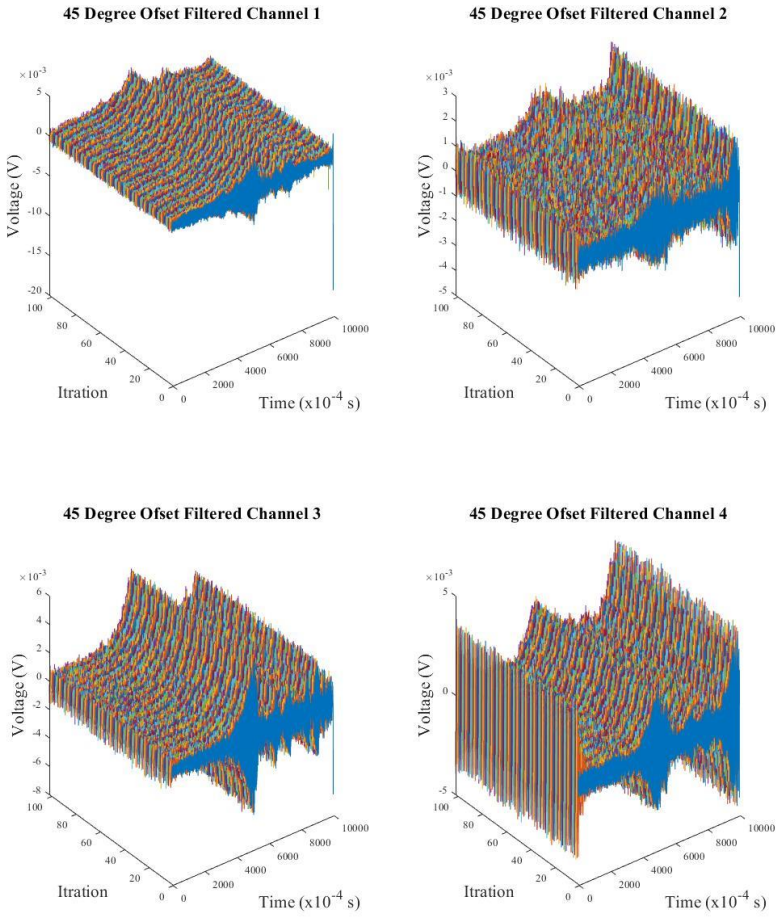


Figure 13: demonstrates filtered and detrended data at 45º using the Butterworth filter which was used in feature extraction.

2) Feature extraction

To extract the features from each data and creating a database for training and testing the CNN, each data point was first split into its individual channels, then the voltage-time signal of the channel was used to find its Wigner-Ville Distribution plot which is the representation of the time-frequency intensity of the signal. Then the plots for each channel was combined into a single black and white image and stored in the database indicating the position of where the data was recorded. In this plot, as shown in Figure 14, the darker the area, the lower the intensity of the frequency captured.

This will highlight the intensity of the response of the natural frequencies each channel relative to each channel.

3) Training, testing and validation:

Finally, the resulting data was split into the Training, Testing and Validation by 80%, 10% and 10% randomly to create a database for training and testing of the proposed CNN. This split of data is recommended by both MATLAB and TensorFlow as best practices and it draws its justification from the Pareto principle.

Figure 14 illustrates the architecture of the deep CNN used in MATLAB to classify the angles of the acoustic source. The produced figures were first downsized to 227x227 pixels and then passed through a Convolutional Layer with an 8x20pixels filter and a Rectified Linear Units (ReLU) activation function to add non-Linearity to the resulting feature map image. The image is then passed through a maxpooling layer to reduce the dimensions of the feature before feeding the features to the fully connected layer for classification. Finally, a fully connected layer (Dense layer) outputs 12 probability for each class and the label with the highest probability is outputted using a SoftMax Layer.

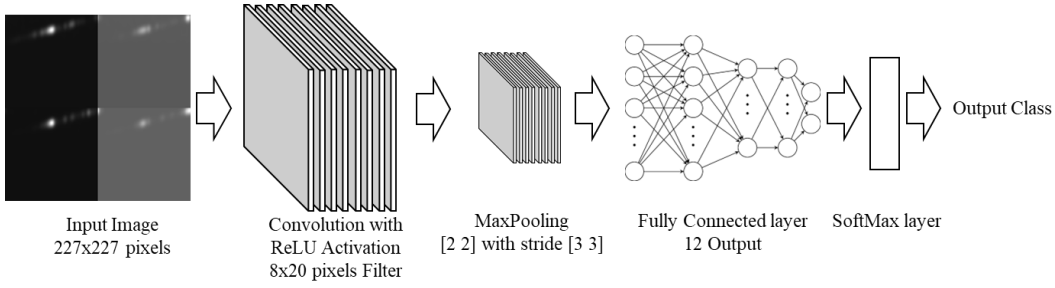


Figure 14: Initial Neural Network to architecture for classification of the angles designed and implemented in MATLAB. The CNN uses a single layer Convolution layer connected to fully connected layer for classification and a SoftMax layer at the end is used to filter the highest probability class and outputs it.

3.3.2 Results and Discussion

The results were very promising with the neural network being able to correctly classify 100% of the results (Figure 15). The limitation of this setup was that the angle placement was inaccurate and more importantly the sensor was moved manually by

hand relative to the speaker (Figure 10). The current iteration of the sensor is still sensitive to environmental change and fragile which means that the smallest alteration could give widely different results. This was apparent in the collected data and we tried to mitigate this issue by placing the sensor in an acrylic holder to protect the device and minimise alteration to the result caused by the environment and the movement of the sensor.

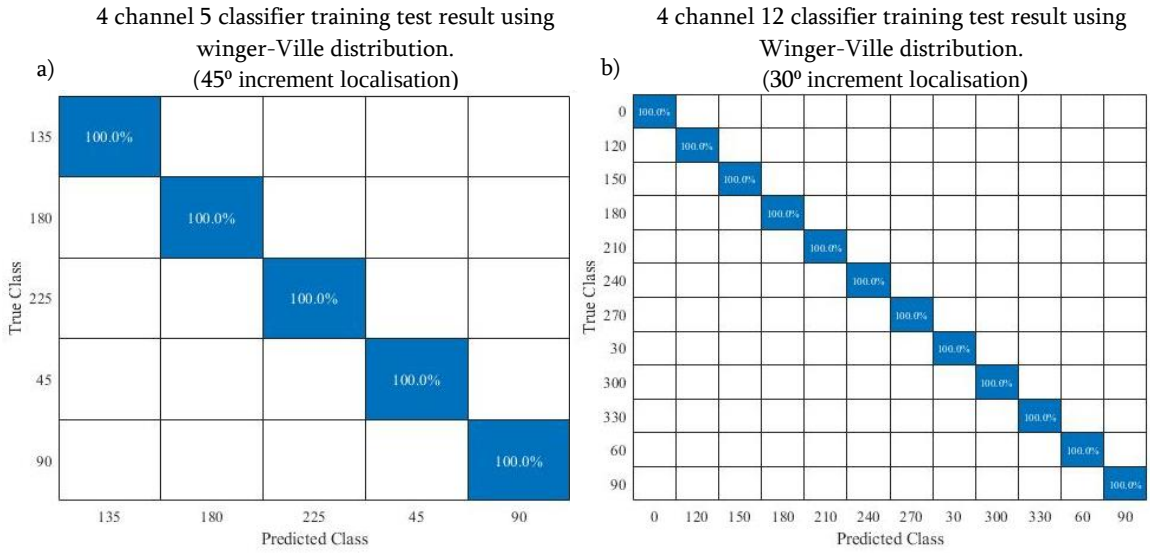


Figure 15: The resulting confusion matrix illustrating the predicted classification vs the true class for the testing data. The graph shows 100% accuracy for all both the 45º increment (5 classes) and 30º increment (12 classes).

Additionally, there was concerns that the 100% accuracy in classification might be resulting from overfitting of data as all the data for this test was collected at once over a short period of time. However, the current setup was not best suited for collected large dataset over a long period of time in different conditions because the accuracy of rotating the device was extremely limited. To address all these issues and eliminate this in future tests and automate the testing process to allow for rapid testing of new devices under different circumstances, a new instrument (Figure 22) was designed, manufactured and assembled that allows to run multiple tests with higher accuracy and close integration with the sensors to allow us to test future iterations of sensors and comparing their performance with minimal bias.

3.4. Speech Recognition

To evaluate the devices for speech recognition, both the SSC and MAS devices were used with a 2 layered CNN trained for speech recognition using some predetermined phrases, which was written in python and utilised Keras and Tensor Flow for the training. The CNN is trained on the voltage output from the devices at sampling frequency of 4 kHz. The data from the sensor is collected using an Arduino Mega, sent over to the main PC over serial communication and fed into the algorithm for processing. For this test, a portion of Shakespeare's Hamlet we chosen and broken into the 6 phrases which was played for the devices over days while their voltage output was recorded. The code used for this section is available in Appendix B (data collection Arduino), Appendix C (the main python code) and Appendix D (real-time testing).

Table 2: Phrases used to test the speech recognition as well as the their ID used for identification in the confusion matrix.

ID	Phrase
A	To be or not to be
B	That is the question
C	Whether this nobler in the mind to suffer the slings and arrows of outrageous fortune
D	Or To take Arms against a sea of troubles and by opposing end them
E	To sleep no more
F	To die

3.4.1. Methodology

1) Data processing and Preprocessing:

To expedite data gathering, audio files containing 100 repetitions per phrase, interspersed with 2-second intervals, were created. Each audio files were played 3 times over 3 days to at the device and the corresponding voltage output was used to create the dataset for training and testing. As with the prior test, data preprocessing involved filtering, detrending, segmenting into individual phrase samples, and databasing. In this particular experiment, a low-pass Butterworth filter, tailored from the experimental parameters, was utilized. With the data collector's sampling rate in mind and applying the Nyquist frequency theorem, the highest usable frequency of

the obtained data was determined, serving as the filter's cut-off frequency to eliminate high-frequency disturbances. Subsequently, the signal underwent detrending. Yet, segmenting the data into distinct intervals and filtering them presented more challenges than in the subsequent spatial recognition test where a sweep of a known frequency was employed.

The next stage of the preprocessing was to segment the collected signal into individual phrases, suitable for training, validation and initial testing of the neural network. To find the suitable points in the signal to separate the intervals, comparison was performed between the large amplitude of the signal to the much smaller amplitude from the 2-second delay and the two intervals along with the expected length of the phrase to segment the signal in the correct intervals. Finally, Using the time period for the phrase, an algorithm was developed that removed any corrupted compromised or incomplete data to ensure that the data used in training and validation was a good representation for each class and did not affect the classification negatively (Figure 16).

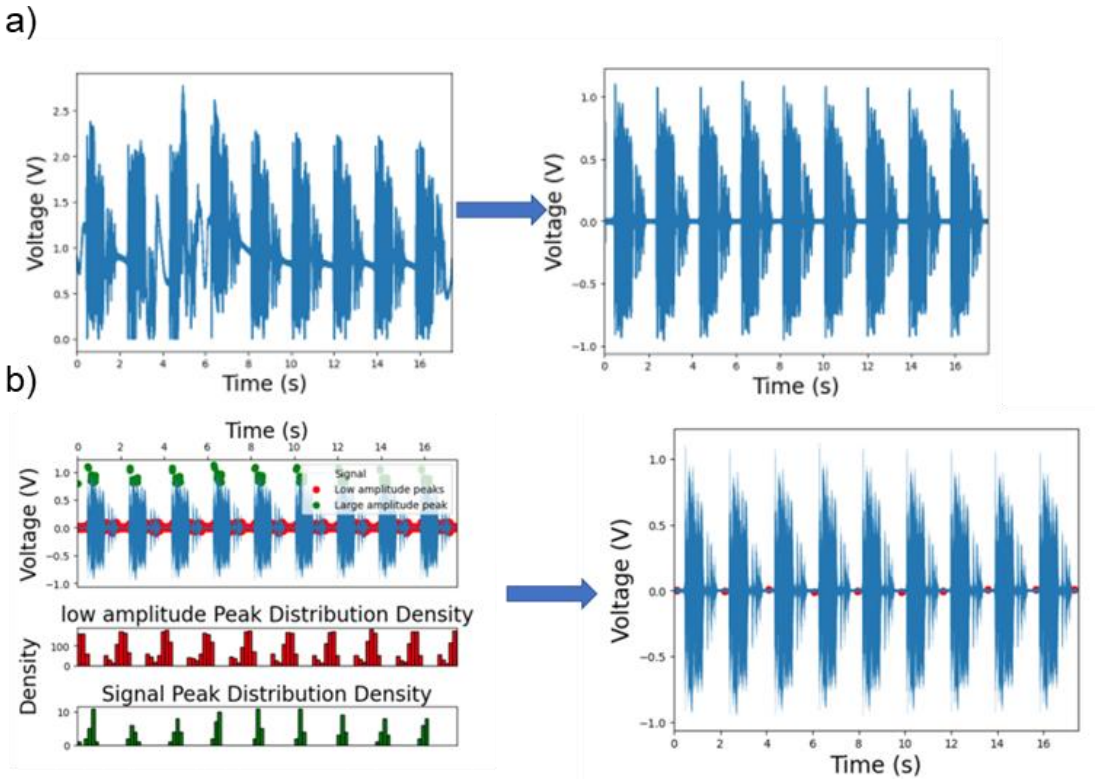


Figure 16: Preprocessing which include a) filtering and detrending the raw data and b) breaking down the data collection into individual samples. This stage helped us to save time in data collection.

2) Feature extraction

Two figures were plotted from the resulting preprocessed data for each of the phrase voltage output and were concatenated together and saved into a single PNG image that was used for training the neural network classifier. The former plot is the scatter plot of the waveform of the signal where the points with higher voltage output magnitudes are more pronounced by using larger and whiter points relative to the black background to highlight the shape of the of the waveform (Figure 17). The latter plot is the STFT of the signal to highlight the relative frequency intensity of the phrase. To ensure that the signals between different phrases are distinguishable, the amplitude of the waveform plot and the frequency for the STFT plot range (the y-axis range) of the plot were kept the same across all the plots for all the phrases (Figure 17).

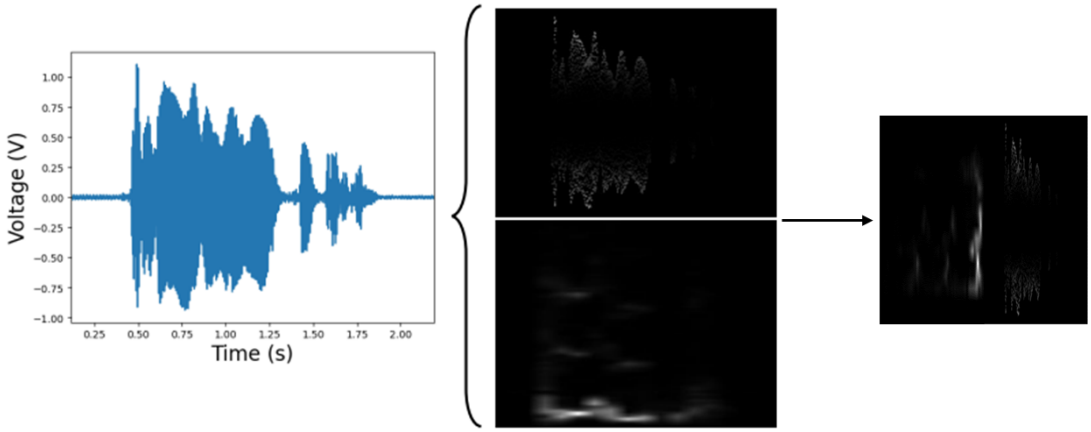


Figure 17: An example plot that was used in the CNN for training and testing. The image consists of a waveform plot of the voltage time graph and the STFT voltage output rotated 90 degrees and concatenated together into a single image. This is to preserve important information in each figure when squeezed into a square. For STFT plot, the impactful information lies in the position of the peaks regarding time and in the waveform data, the import information is in the shape of the waveform while removing the temporal information that could be used to distinguish between long and short phrases.

3) CNN architecture and classification

Figure 18 illustrates the architecture of the CNN used for the classification of the phrases. The input image of the CNN is a resized horizontal rectangular image

composed of an image of the waveform concatenated with an image of the STFT plot of the data resized into a 64x64 pixels square image as shown in the Figure 17. Therefore, given that one side of the plot was going to be squeezed, the plots was oriented in a way to ensure that the features from each plot were preserved. For the STFT data, it was crucial to preserve the timing information because the frequency range on the y-axis was constant for all the phrases where the period for each phrase and where the time of frequency intensity was different; Therefore, we rotate the STFT figures by 90 degrees to ensure any timing information was kept and the frequency data was scaled down uniformly across all the plots. In the waveform plot, the outline of all the peaks and how they are all correlated to each other were analysed. Therefore, to preserve continuity, the waveform plot was kept the same.

To speed up the processing time, two larger convolutional filters were used and each of the convolutional layers was followed by a Maxpooling step to reduce the size of the node. The 2-stage convolutional layer filter works very well in creating a feature map for extracting and comparing features to classification while light enough to train and run on any computer. However, this leads to CNN overfitting the data during training. To combat this, two dropout layers were added, one after the second convolution and the other one in the dense layer, to remove some nodes at random to discourage complexity and overfitting [31]. The final CNN architecture was achieved by a small manual adjustment through trial and error on the filter and the fully connected layers sizes to avoid over fitting while maximising the accuracy and minimising loss.

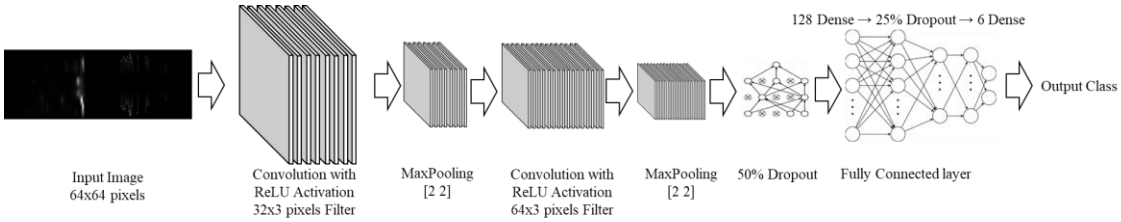


Figure 18: Initial Neural Network to architecture for speech recognition was designed and implemented in Python. The CNN uses double convolutional layers followed by Maxpooling

layers to reduce the dimension connected to fully connected layers for classification. Dropout layers were introduced within the network to prevent from over fitting the data.

3.4.2. Results and Discussion of speech recognition

For the single channel circular symmetric device, from the original dataset 300 sample points per class collected to create a total of 1800 sample dataset for training, validation and testing. To evaluate the neural network and device, 10% of the data was used. After optimising the algorithm and ensuring that the input image had enough information, we proceeded to run a blind real-time test where the algorithm chose a random phrase and look at the response of the device and algorithm. Figure 19 shows the confusion matrix for the testing data and the real-time test. The confusion matrix for the testing data illustrates that the device collects enough information to be used for speech recognition. While the real-time test was also accurate, there are slight variations and overall lower accuracy. This is because the training, validation and testing data collection and real-time testing happened on different days and the sensor tends to produce slightly different outputs depending on the environmental condition (Especially the humidity and temperature). This slight variation can be overcome by either making an environmental protection case for the sensor to reduce the environmental effects or running the test on multiple days and using a much larger dataset to train the neural network to account for this change.

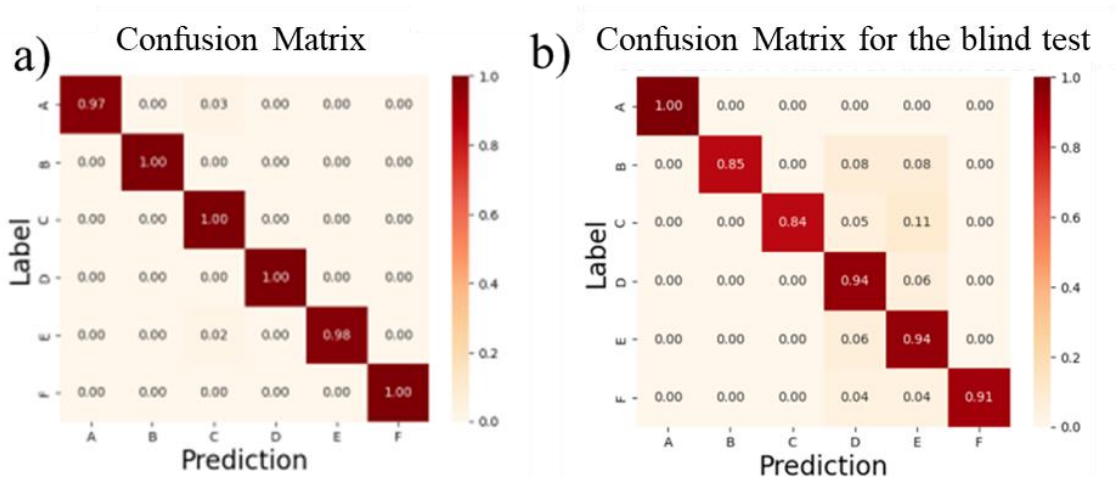


Figure 19: a) Testing Confusion Matrix illustrating high accuracy and precision for the speech recognition CNN using the 10% original data set.b) Illustrates the real time speech recognition testing show high level accuracy.

The data showed that such a piezoelectric sensor with its current simple design, can collect large amount of information from the acoustic stimuli that can be used for speech recognition. This is apparent from the confusion matrix illustrated in Figure 19 where the device was able to classify the test data collected and only miss classify 2 test samples in addition to being able to perform extremely well classifying never seen before inputs in the blind test in real-time.

Due to time limits, a smaller dataset of 1200 sample was collected from the single-channel asymmetric spiral device as opposed to the 1800 collected from the symmetrical circular device. Similar to the symmetric device, the data was randomly split into 80%, 10% and 10% for training, validation and evaluating the neural network. This was much smaller dataset compared to the one used to train the symmetrical device; however, it yielded a much more accurate model which was able to predict all the test figures correctly and efficiently. The Figure 20 below shows the result of the experiment in a confusion matrix.

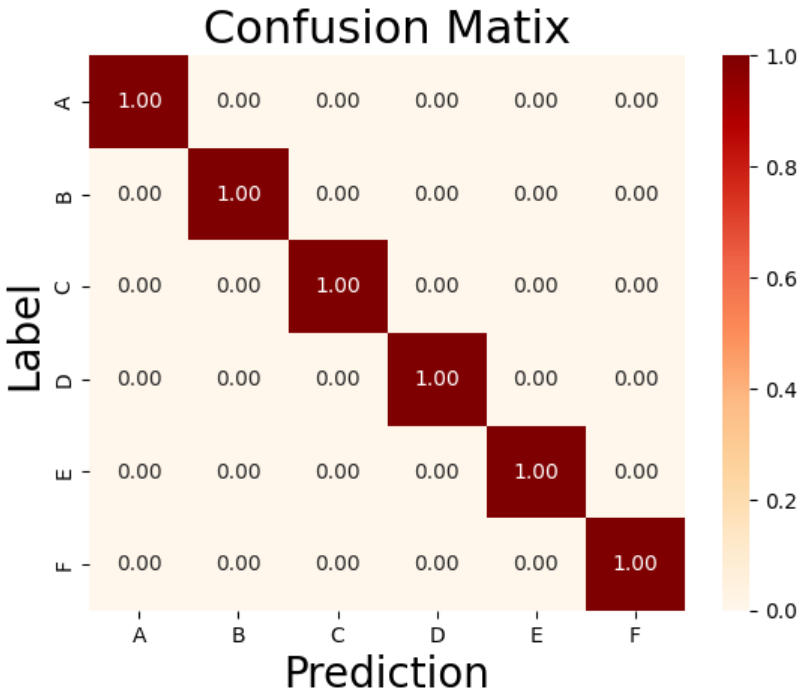


Figure 20: Illustrates the result of the classification test confusion matrix of the single channel spiral device for speech recognition. The figure shows that the device can perform speech recognition very well with 100% accuracy with 188 supports for the test.

3.5. Spatial localization Recognition

3.5.1 Instrumentation and integration for spatial recognition

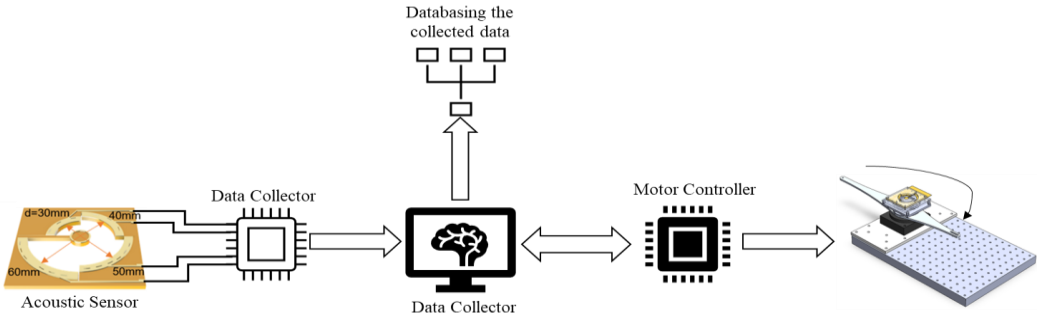


Figure 21: The architecture of the spatial data collection and testing setup. An Arduino Mega was used for data collection and Arduino Uno was used for controlling the motor. All the collection and control were coordinate through a central computer using Python. The code for this section is available in appendix B (data collection), E (stage controller), F (main python code), Appendix G (Real-time spatial recognition testing) and Appendix H (the testbed CAD Design).

The testbed is shown in show in the fire which consist of an aluminium arm to hold the sound source (mouth simulator) attached to a hollow rotary stage with sensor sitting in the middle of the stage. There is a hall effect sensor on board on the rotary stage which is used to set the home position (zero degrees) for data collection. Since the process of data collection is automated, the stage will zero itself to ensure that it is in the correct position and has not lost its bearing due to obstructions such as entangled wires. When in calibration mode, zeroing its position, it will calculate the angle it turns to reach zero position and compares it to the angle that it assumed it was. If the assumed position and actual position does not match, it will send an error and remove all the data collected since last calibration sequence.

To assess a directionality of the sensor, an automatically controlled rotatory stage was designed, manufactured and integrated with the multi-channel data collection system. To collect data, an Arduino Mega with a custom shield that takes advantage of the Arduino's analogue pins were used for data collection from the sensor. The system allows to rotate the sound source around the sensor accurately and collect data for a different range of tests so as to measure the position and corresponding voltage output of the sensor with high accuracy, realising its full potentials for precise positioning in three dimensions and overcoming the limitations of previous Platform L. The

aluminium parts were waterjet cut by UCL Institute of Making. The general overview of the architecture of how different components are connected and communicate with each other is shown in Figure 21 and the CAD design for the stage and the final stage is shown in Figure 22.

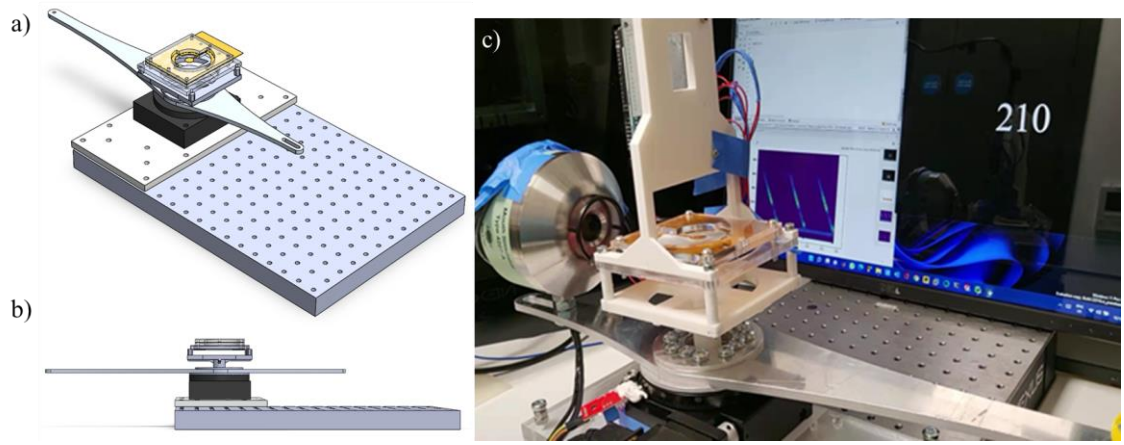


Figure 22: Solid works design and final assemble instrument to test directionality recognition. The stage moves the speaker around the sensor automatically and collects its output directly.

The device was placed at the centre of the stage and connected to an Arduino MEGA for data collection. The device had to be at the same height as the speaker and we had to ensure that none of the wires were tangled and interfered with each other as the stage was programmed to be able to move 360˚ around the sensor. To ensure that the wires do not tangle and the speaker is free to rotate about the device, the Arduino is placed on top of the sensor.

3.5.2. Methodology

1) Data processing and Preprocessing:

Analogous to the previous experiments and setups, the collected data was first processed as shown in Figure 23. As shown in step 1, initially the raw data was filtered by passing the collected data through a digital bandwidth Butterworth filter deriving from the STFT plot and removing the noise, and then detrended. As this was a known sweep frequency input signal, the breakdown of the signal into individual sweeps was a straightforward task to the speech recognition data segmentation. By plotting the STFT of the data, it is clear to see where one sweep has ended and where the next has

started, which can be used to separate them into individual intervals. By using this method in step 2, the input signal collected was segmented into individual complete intervals of sweeps. Any incomplete sweep was removed from the data set.

2) Feature extraction

Finally, in steps 3 and 4, the signal was verified by evaluating the frequency sweep range to ensure that the sweep is complete and correct data is being used for testing and then the grayscale STFT of each channel was created and concatenated together to form the final image for the CNN training, validation and testing. Similarly, to the other 2 experiments, the data was split into 80%, 10% and 10% for training, testing and validation respectively.

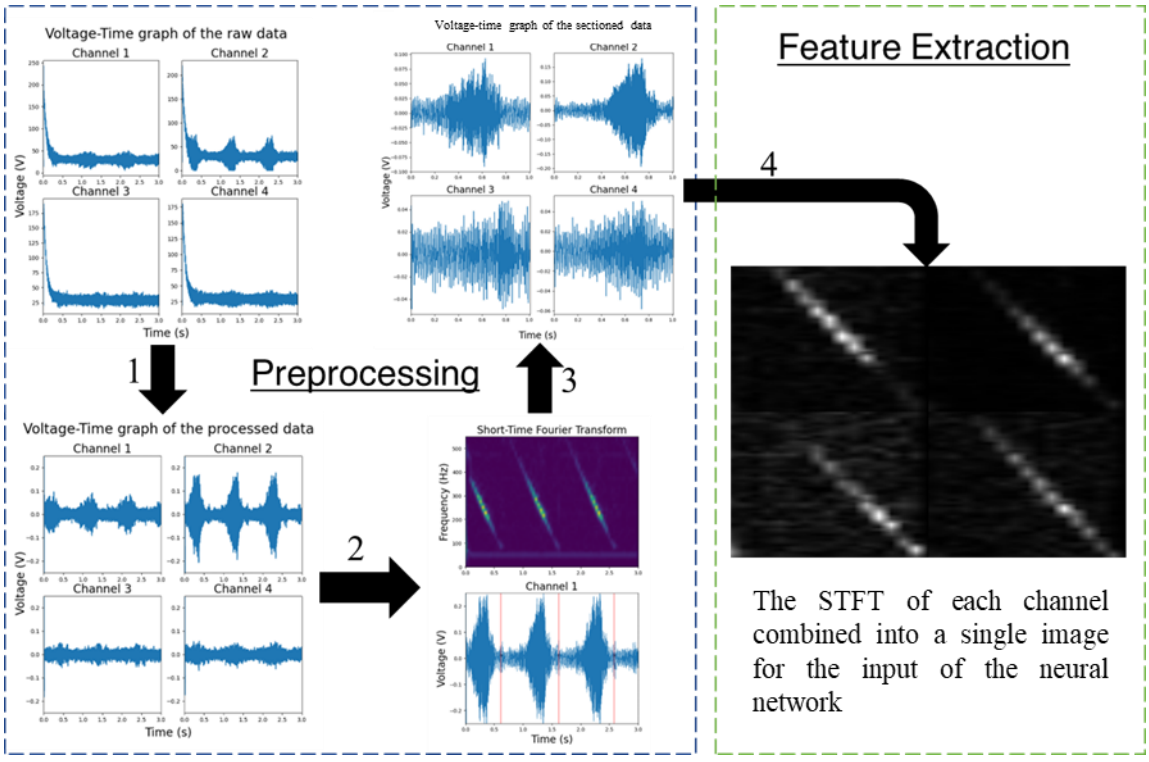


Figure 23: Typical data preprocessing and feature extraction for the directional recognition CNN. The collected data was 1) filtered and detrended and 2) the resulting data was separated using the STFT plot to determine where one sample point ended and where the next sample started. 3) the individual samples were used to make a lower resolution STFT plot and 4) the plots were combined into a single image.

3) CNN architecture and classification

Similar to the CNN used in Section 2.4.1, the neural network takes an image of the combination of 4 STFTs of the 4 channels and then downsized to 150x 150 pixels from 1220x900. This would increase the speed of training and testing. The neural network takes these images and feeds them through a network with similar architecture as the one used for speech recognition with the major difference being in the 12 output classes (the 12 angle positions) as opposed to the 6 classes for speech recognition (Figure 24).

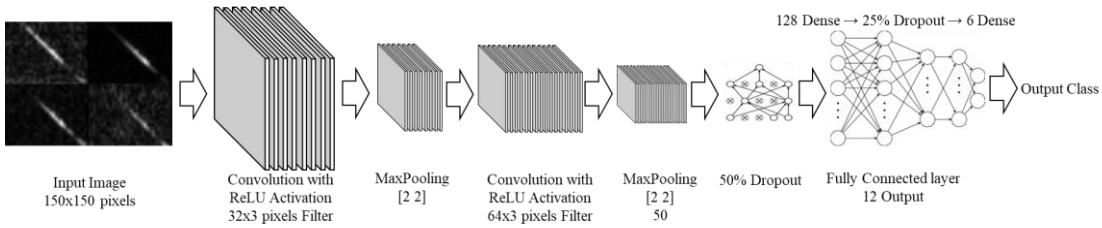


Figure 24: Initial neural network to architecture for spatial recognition classification was designed and implemented in Python. The architecture used for spatial recognition is the same as the one used for speech recognition.

3.5.3. Result and discussion of spatial recognition

1) Spatial recognition of asymmetric spiral devices:

As shown in the Figure 25, three set of experiments were conducted in 1) planar Azimuth (θ) and 2) orthogonal (ϕ) directionality recognition and 3) distance (x) recognition on the device for 3D sound source localisation. All three experiments showed a high level of accuracy. Despite the lengthy period of time for the data collection and the changes to the environmental conditions, the performance and accuracy of their cognition was reproducible. Moreover, because the data for each angle was collected over few days multiple times, the range of data was large and various in environmental changes to ensure that they were not over fitted or trained. This shows that the frequency selectivity and asymmetry of the device can be utilised for spatial recognition. The device showed 100% accuracy in distance recognition, 94% accuracy in the orthogonal (ϕ) direction and 97% accuracy in the planar (θ) direction. Because the data for all these experiments were collected over days with

plenty of variations in the data, it shows that the neural network is generalised and able to localise the sound source very well as evident by the confusion matrixes in the Figure 25.

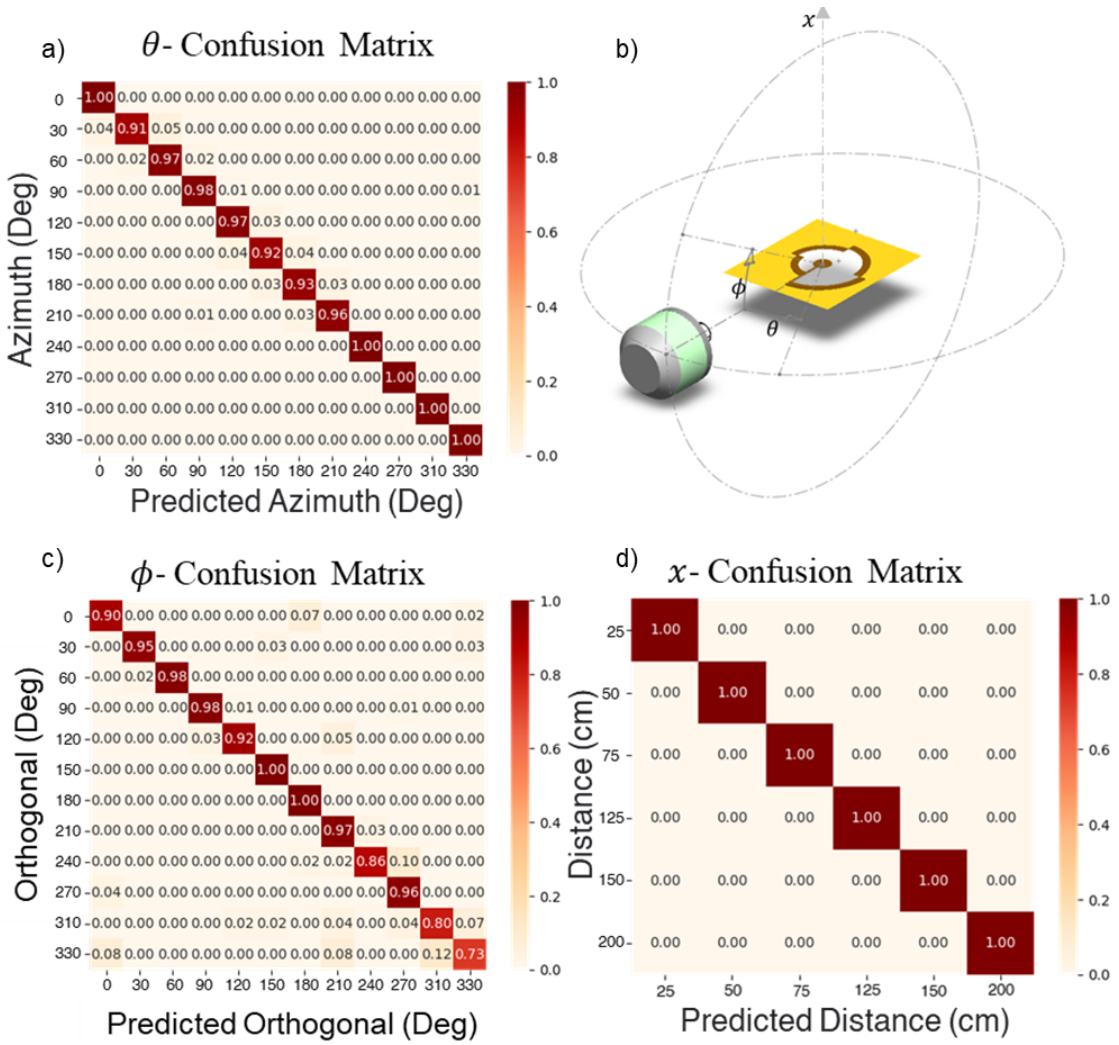


Figure 25: The spatial recognition confusion matrix results. θ indicates the 360˚ directional recognition testing in plane with the sensor. φ indicates the 360˚ directional recognition testing in the orthogonal direction to the sensor and *x* indicates the distance recognition from the sensor.

As illustrated in Figure 25a, the experiment conducted with azimuth angles to the device shows lower accuracy between 30 degrees and 180 degrees. All misclassified points are mistaken for their adjacent classes, indicating that the predictions are not random but are off by only ±30 degrees. This accuracy can be improved by collecting more data to train the model, using more sophisticated models such as YOLOv8 to YOLOv10, or employing higher resolution and RGB input images.

Figure 25c illustrates that the model is least accurate when the sound source is positioned towards the centre of the device, specifically between 330 degrees and 30 degrees on either side. One of the largest misclassifications occurs at 0 degrees, where the sound source is incorrectly identified as being at 180 degrees, directly opposite side of the device. Other misclassifications occur in adjacent classes, with the lowest accuracy observed at 330 degrees, where the sound source is misclassified as being at 0 degrees.

These findings indicate that the misclassifications are not random; rather, the device captures significant localization information. To improve accuracy, particularly at 330 degrees, a larger number of samples from this angle should be included in the training set. Additionally, similar to the azimuth experiment, increasing the dataset, using more sophisticated models such as YOLOv8 to YOLOv10, and employing higher resolution and RGB input images could significantly enhance the device's performance.

Finally, the directionality test in Figure 25d shows that the device is 100% accurate in identifying the distance of sound source. This shows that the device is extremely sensitive to the sound intensity and can identify sound source from different distances very accurately. Because the data was collected over days in different environments, it mitigated the chance of overfitting the model.

2) Electrode configuration and resolution of direction recognition

After verifying the hypothesis for spatial recognition, i.e. that the MAS device could be used for sound source localisation, the correlation between the electrode configuration and spatial recognition resolution and accuracy of the device, i.e. what is the smallest angle deviation that the device can recognise was characterised. Different data sets were collected from the different electrode channel numbers using the same device as shown in Figure 26, and the sensor performance in response to the different input configurations of feeding the collected data to the neural network was also evaluated. Figure 26a illustrates the explored configurations and the results. In

the software-processed results as depicted in Figure 26b, voltage data were gathered from each channel individually across all electrode configurations. These data were then merged during the preprocessing stage. The subsequent voltage-time series was utilized for feature extraction prior to input into the neural network. Conversely, Figure 26c displays outcomes where channel data were manually integrated, and the cumulative signal was then acquired.

The aim of this experiment was to evaluate the effects of electrode configurations and data process method on the resolution and accuracy of spatial recognition of the spiral devices. The results will render guidance on further optimisation of electrode design and data process. In other words, what is the cost-effective design of the number of the electrodes with desired resolution and accuracy at reasonable costs of manufacturing and data process and whether it is necessary to make as many as possible ultra fine electrodes along the spiral device so as to achieve higher resolution and accuracy through collecting voltage output generated from the piezo-nanofibres with nearly continuous variable length along spiral channel arrays.

The result shows that the information is preserved in the signal and can be used to spatial recognise with various resolutions and accuracy depending on the electrode configuration. As expected, the higher the number of channels used to collect the data, the higher accuracy of the neural network to pinpoint the source of the sound. From the Figure 26b and 26c, it is apparent that all configurations are very accurate with over 90% accuracy from 10 degrees and above. However, the computation and data processing could be reduced by combining channels either through software or hardware which leads to a faster response from the system for localisation.

This experiment demonstrated that the accuracy across all configurations dropped rapidly below 5 degrees increments and while the four channels configuration shows the best outcome, majority of the information is collected within a single channel, regardless of being collected manually in a single channel or post-processed by the software, which concluded that the increasing the number of electrodes will improve the accuracy and precision of the device. Therefore, the spiral device with

multichannel electrodes is capable of recognising the sound direction at high resolution and accuracy. Using only one channel could provide reasonable accuracy of spatial recognition with minimised data collection and process. Moreover, the electrodes and the fibres do not have to be a discrete constant radius and can be a continuous spiral to collect more information. This is because this experiment showed that a single channel can collect all the information with very gradual losses. Figure 26b and 26c shows that similar to 2 and 4 channel configurations, while the accuracy is lower, it is still over 90% for a single channel device.

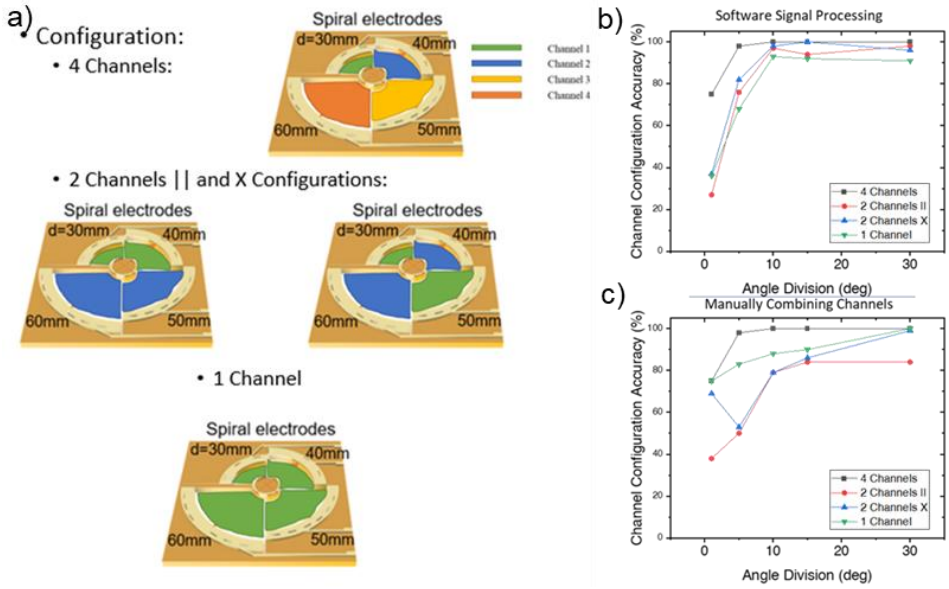


Figure 26: Different configuration of 4 channel device tested to find the correlation between the number of channels and resolution. In the figure, a) is the different configurations explored for spatial localisation, b) is the result of the experiment for the software adjusted signal to match the configuration and c) is the where the electrode connections were manually altered to match the configuration.

In summary, the sensor demonstrates significant potential, exhibiting remarkable accuracy in both speech and spatial recognition tasks. For efficiency in this research endeavour, the dataset for each classification was confined to 210 data points, with the objective of amassing all these points within a single day to mitigate environmental influences on the experimental outcomes. Future enhancements could encompass expanding the data points per class over an extended duration and rigorously controlling environmental variables. Such refinements would likely

augment the neural network's performance, leading to enhanced results in both real-time and recorded assessments.

3.6. Conclusions

As hypothesized, a single MAS device can handle both sound source localization in all directions and distances away from the device and sound pattern recognition (speech recognition), whereas the SSC device is limited to speech recognition. The sound localization capability of these devices stems from the electrode design, which incorporates frequency selectivity similar to that of a healthy human cochlea. To enhance these devices, further research is needed into the materials and fabrication methods. This research should aim to improve the consistency of device responses across different batches and allow for better control over the placement of fibres on the electrodes. Such improvements would enhance frequency selectivity and reduce the size of the devices.

To improve on the performance of the classification, there are number of techniques that can be used. Mainly a better more complicated model can be employed and retrained on the gathered to improve on the performance of the classification. Newer models such as YOLOv8, v9 and v10 have incredible practical applications for classification in real time in industry using only consumer GPUs. Additionally, as they use the RGB image input, they can use more information in classification than the model used for these experiments.

Chapter 4

4. Design, Fabrication and material improvements:

4.1. Manufacturing multichannel single micro/nano fibre device

4.1.1. Background

The acoustic sensors that we have fabricated and tested to this point, illustrated a number of desirable properties. The radially aligned piezoelectric nanofibres within the four channels were obtained by the additional local static electrical field between four pairs of electrodes and provide predictable frequency selectivity. Despite the additional local electrical field, the fabrication process gives little control over where the fibres will be positioned between the electrodes. As a result, the uniformity and thickness of nanofibres coverage across all four channels become poorly controlled, which leads to a membrane-like structure with crosstalk between channels and uneven thickness.

For enhanced precision in the fibre positioning process, Electrohydrodynamic (EHD) printing is employed. This method leverages a targeted fibre deposition via near-field electrospinning combined with electrode movement, ensuring finer control during fabrication via additive manufacturing techniques. This research delves into three distinct EHD printing techniques to craft multichannel single fibre sensors, varying in both diameter and radial properties, tailored for high-frequency selectivity. A pre-existing slicer has been developed, which offers adaptability across all three techniques.

One other limitation of the electrospinning fibres is the size limitation of the sensors. The average cochlear in an adult is about 9.20 x 6.30 x 8.00mm, whereas the sensor we have fabricated is much larger with the smallest channel coming at 30mm in two dimensions. EHD printing will potentially allow us to reduce the size of the sensors through controlling the length and the diameter of the fibre in three dimensions.

4.1.2. Aims and Objective

The primary objective of this research chapter is to design and implement a fabrication stage that facilitates the precise deposition of piezo nanofibers onto designated locations on an electrode. The stage should not only ensure control over the exact positioning of the electrode but also offer meticulous speed control during its movement. The significance of such precision arises from the relationship between the stage's movement speed and the material feed-through rate from the needle. If the movement outpaces the feed-through rate, the fibre risks elongation and thinning, potentially culminating in breakage under extreme velocities. Consequently, the uniformity and integrity of fibres across the device are contingent on this stringent speed regulation. Furthermore, by achieving granular control over fibre thickness, the research intends to elucidate the interplay between fibre diameter, its resonant frequency, and the resultant piezoelectric output, providing insights into optimizing the design and function of such devices.

4.1.3. Specialised Slicer: From design to waypoints

For precise EHD printing stage control, a method was devised to translate both simple and intricate fibre patterns intended for electrode exploration into waypoints for the printing bed (electrode). Waypoints are a set of x, y, z coordinates on the print bed that are converted to the number of steps for each stepper motor to take. This conversion subsequently determines the precise motor rotations necessary for the desired movements. The envisioned fibre designs, being substantially finer than typical 3D printing models, are not easily scalable due to their relative size ratio with the electrode. Conventional 3D printer slicers are incompatible with such designs. Moreover, ensuring the production of a continuous fibre was pivotal, given that interrupting fibre production midway in EHD printing is not feasible due to factors like viscosity and the electric field's downward pull. Meeting these specific requirements necessitated the creation of a tailored slicer. This was achieved by developing a MATLAB program that processes STL files, obtainable from standard CAD software such as SolidWorks, layering them and subsequently translating them

into a continuous set of waypoints. These waypoints guide the EHD printing stage, dictating the precise rotations required for both X and Y axis motors.

For precise EHD printing stage control, a method was devised to translate both simple and intricate fibre patterns intended for electrode exploration into waypoints for the printing bed (electrode). This translation subsequently determines the precise motor rotations necessary for the desired movements. The envisioned fibre designs, being substantially finer than typical 3D printing models, are not easily scalable due to their relative size ratio with the electrode.

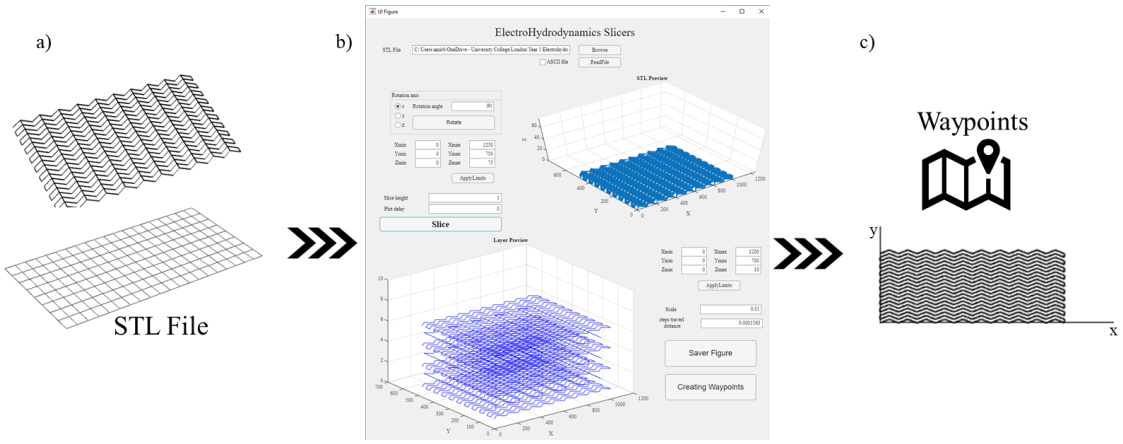


Figure 27: The process of getting the waypoints for the EHD printing. a) STL file can be obtained from CAD software, b) is the snapshot of the Slicer designed in MATLAB specially programmed for the EHD printer and c) is the waypoints generated by the program. The code for the slicer is available in Appendix I.

4.1.4. Electrohydrodynamic Printing Stage

4.1.4.1. Cartesian Axis Stage

To improve upon the fabrication, ElectroHydroDynamics printing (EHD printing) was explored which has been well studied and shows the potential to fabricate small devices very accurately. A literature review on this topic illustrates that it is possible to control the diameter of the fibre by adjusting the movement speed of the collector. This is especially useful for this project since the only way that we could adjust the resonance of the channels was by changing the length of the fibre and the fibre diameter was quite uncontrollable. The stage used for this project has 0.025μm resolution and a maximum speed of about 25mm/s which is not comparable to the

speed used in the literature, but we can compensate for this by reducing the flow rate accordingly to adjust for this. Moreover, it is expected that the minimum fibre diameter fabricated by this method is over 6x larger in diameter than fibre fabricated using electrospinning. [32]

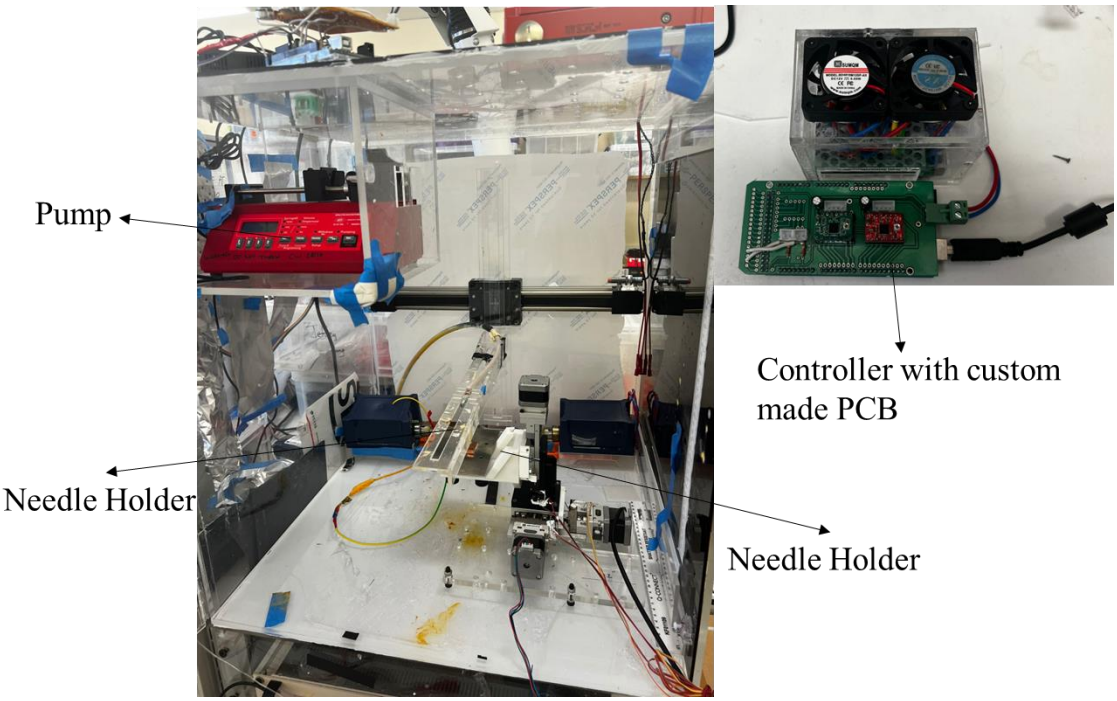


Figure 28: The cartesian EHD printer and controller designed for the Stage. The image on the left illustrates the stage as it sits in the electrospinner's enclosure and the image on the right is the custom PCB designed to control the stage. The full design CAD and code available in Appendix K and J.

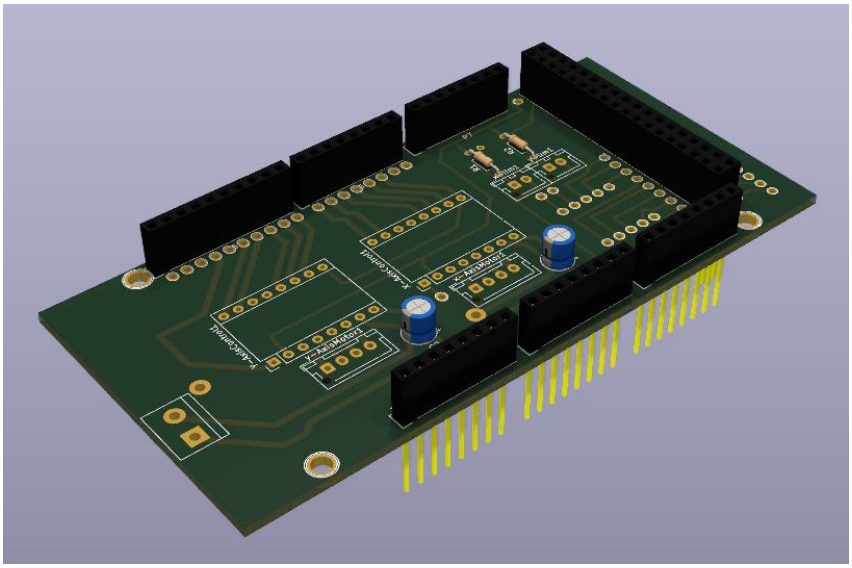


Figure 29: Show the circuit designed CAD in KiCad which was manufactured for controlling the EHD printer. The circuit is designed as an Arduino Mega shield to control stepper motors and read the data from limit switches for initial homing of the stage. The code for the Arduino was written specifically for the shield to enable controlling the EHD printing stage precisely Full schematics of the Figure available in Appendix K.

4.1.4.2 Radial Axis Stage

Given that the majority of the devices that will be explored are inspired by human cochlear and have a circular structure, it would be interesting to explore a stage that moves radially with the needle moving horizontally. In the literature, the speed explored is between 1000 to 4000 mm/minute. However, the current setup can reach the maximum speed of 1500mm/minute which is in the lower end of what the literature explores. It is expected that radial method will be faster and allow the stage to reach higher speeds which is comparable to the ones tested successfully in literature [33]. This is because the motor used to move the needed holder is larger and adjusted for higher linear speeds with a more powerful full motor controller and the stage can move radially at the maximum speed which is about 50π rad/s (1500rpm).

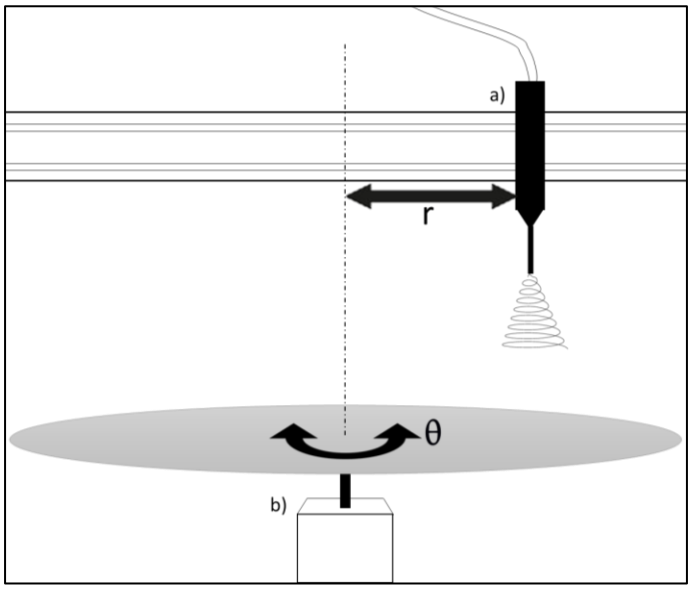


Figure 30: Diagram of a potential radial printing stage. In the figure a) shows the needle used to eject the polymer fibre controlled by a linear actuator horizontally by the amount r from the centre of the stage and b) shows the stage sitting on top of a motor that controls the stage radially by the angle θ.

4.1.4.3 Motionless Printing

Finally, there are methods to control the position of the fibre that does not include a moving stage and the process is controlled by 1) segmenting the collector electrode and using switches moving the printing location to the desired position [34] or using 2 jet defecting electrode to deflect the fibre to the desired position [35].

4.2. Exploring new Material

4.2.1 Background

There are number of limitations with the current material used in the sensors that need to be addressed. Therefore, this section is dedicated to exploring new materials with should have better mechanical properties and performance. Current Device, while having great responsive and voltage output. This was achieved largely because of the improved polarisation of the highly aligned fibres in addition of BTO nanoparticles. However, adding the BTO to PVDF undermined the fibre's mechanical properties and make the fibres brittle due to poor interface between BTO and PVDF. This is the direct result of how the BTO nanoparticles are embedded into the fibre or surface and generate cavities, acting as defects at their interface as they are being electrospun and dawn by the electric field (Figure 31).

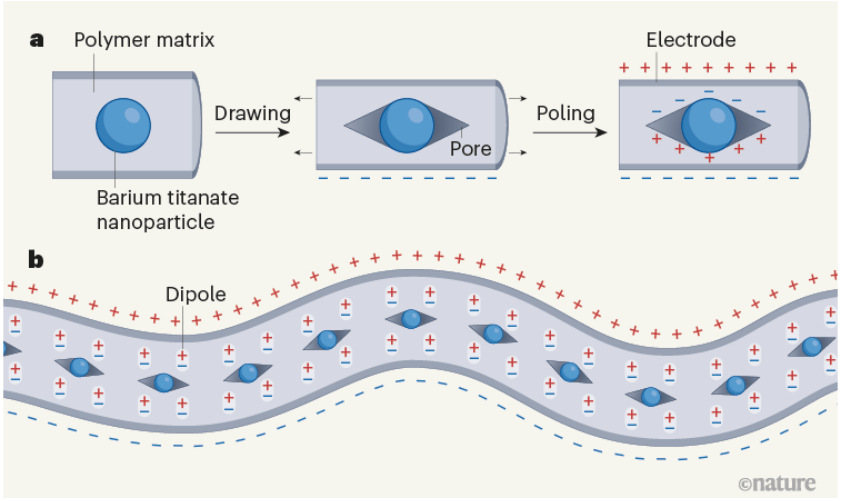


Figure 31: A single-fibre sensor with high sensitivity and flexibility [36].

4.2.2 Aims and objectives

To overcome this, we hypothesised that we could fabricate hollow PVDF fibres loaded with more BTO nanoparticles in the core. To achieve this, PVDF is going to be coaxially electrospun as the sheath with polyethylene oxide (PEO) and BTO nanoparticle aqueous solution in the core. Then the water-soluble PEO can be dissolved in water to obtain the desired core-sheath structure.

4.2.3 Methodology

The first step for achieve this goal is to optimise the electrospinning parameters of PEO and PEO/BTO and find the maximum concentration of the BTO in PEO. Before adding the BTO to the PEO and coaxially spin the fibres, first the right concentration of the PEO solution had to be determined for the most consistent fibre draw with minimal variance in the fibre diameter. For this experiment, PEO with the molecular mass of 100,000Mv and 400,000Mv was used to make a range of different concentrations of the aqueous PEO solution. PEO electrospinning is a topic that a number of researchers have explored before and there is a lot of information on the optimised condition to electrospun PEO [37] [38]. Several experiments were run to find the optimal electrospinning process for 400,000Mv PEO shown in Figure 32.

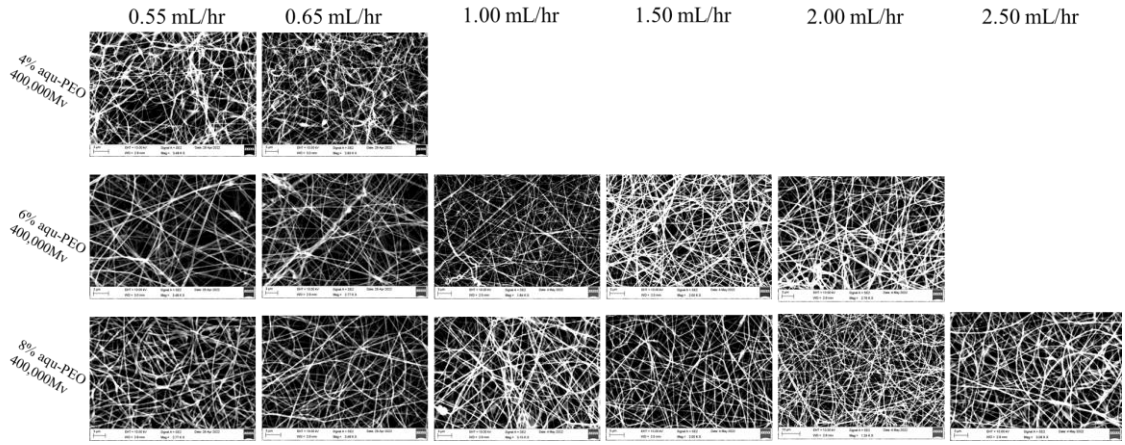


Figure 32: Optimisation result of the aqueous 400,000Mv PEO electrospinning.

4.2.4 Results and discussion

In this experiment, the optimal condition for electrospinning the PEO was explored. To find this optimal condition, the only variables that could be changed were the concentration of the PEO solution and the flow rate of the PEO solution and this is because the rest of the variables, i.e. the voltage and the distance between the collector and the needle are fixed at 15kV and 15cm respectively, since these are the variables used for the electrospinning the PVDF.

The next step was to estimate the fibre diameter under each condition. To do so, a slightly modified MATLAB code was used (available here) to automate the fibre

diameter measurement and estimate the average diameter. The results were verified by measuring the fibre diameter manually using ImageJ and then the comparison between various dimeters of PEO under different conditions and PVDF fibre diameter was plotted in Figure 33.

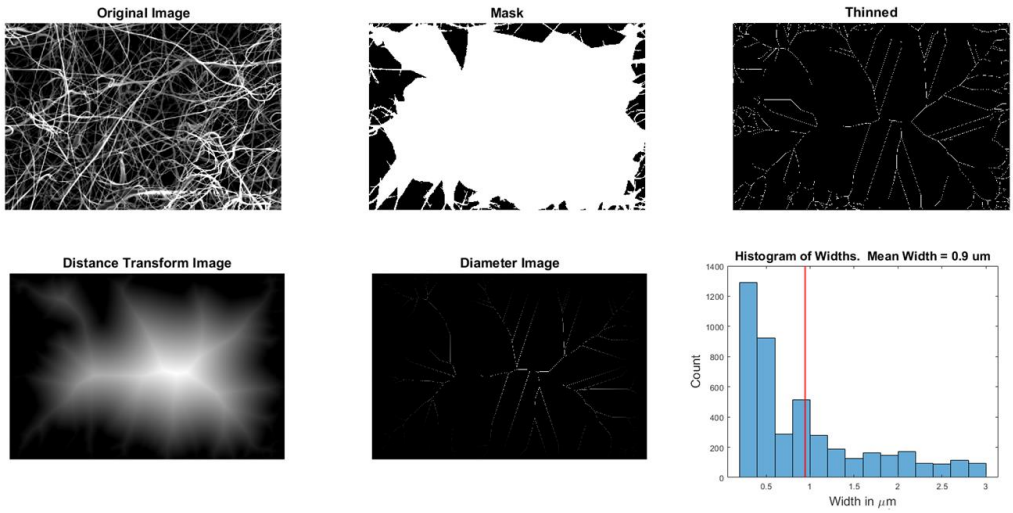


Figure 33: Program for extracting the average diameter of the fibre for PVDF and various PEO concentration and printing conditions. To automatically measure the diameter of the fibres, the program first takes in a calibration value and then uses edge detection to create a mask to get then position of each fibre. As fibres are in different distance from the detector when taking the image, the code only uses fibres with a certain predetermined intensity and only measure the diameter of those fibres to ensure a correct estimation for the mean diameter of the fibres.

The MATLAB algorithm converts the image into a binary format (1s and 0s) to distinguish the background from the fibres. It then computes the Euclidean distance transform, which calculates the distance from each pixel in the binary image to the nearest background pixel. Additionally, it generates a skeleton map, capturing the morphology of the binary image and outlining the fibres. By multiplying these two

matrices, a diameter image is produced, from which a histogram of the fibre widths can be obtained.

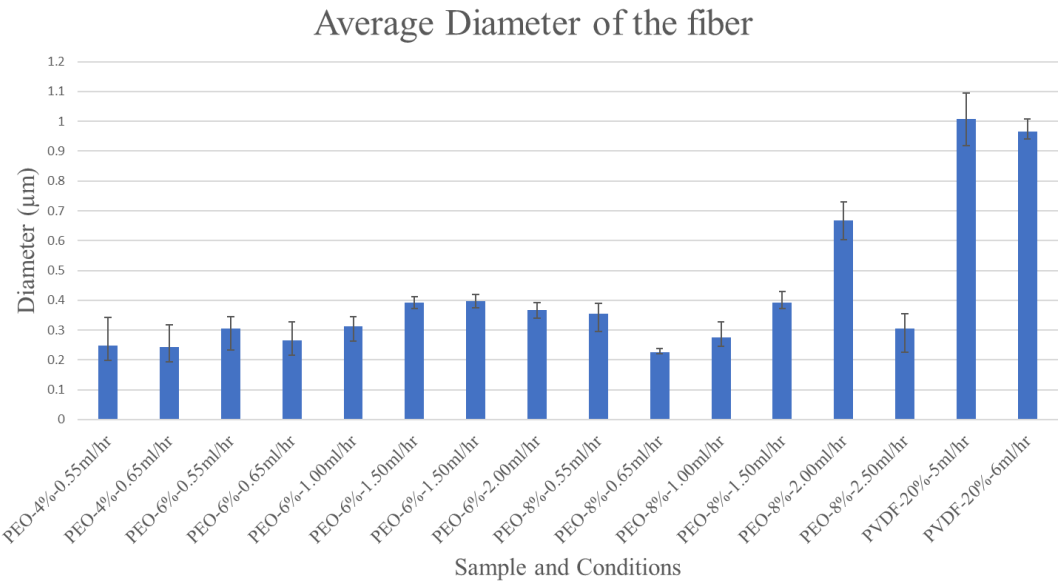


Figure 34: Average Diameter of the electrospun PVDF and PEO for different concentration of PEO and various feeding rates.

From the findings of the preliminary electrospinning trials, it was discerned that the ideal concentration for PEO stands at 6%. Concurrently, the most suitable flow rate for the PEO was identified to be 0.65mL/hr and 1.5mL/hr. The rationale behind this determination can be attributed to the fact that a concentration of 6% PEO consistently resulted in the production of uniform fibres with minimal variation in the diameter in different batch, demonstrating minimal droplet accumulation on the electrospun fibre surface. A further point of significance is the observed correlation between the flow rate and the fibre diameter. Specifically, as the flow rate elevates, there is a commensurate increase in the fibre diameter. This relationship not only solidifies our understanding of the process dynamics but also introduces a potential avenue to exercise precise control over the core diameter during coaxial electrospinning, enhancing the versatility and precision of the method.

To determine the desired inner diameter for the fibres, i.e. the desired PEO fibres dimeter, Further testing had to be carried out to test the performance of the fibres and their durability both after dissolving away the PEO and testing the devices. However, the results of these tests provided valuable insights into the concentration and flow

rate that yield the most consistent PEO fibres. Additionally, the findings demonstrated that the diameter of the PEO fibres produced is significantly smaller than that of the PVDF fibres, facilitating effective coaxial electrospinning.

Unfortunately, due to time constraints, the research into fabrication of the devices using the EHD printer and exploring new material to incorporate better performance and higher control over frequency selectivity is incomplete. For the duration of this project there was only enough time to design and manufacture the stage as well making a slicer than that enables translating simple or complex designs into a set of waypoints instructions for the stage. It will remain to see whether the device and control over the accuracy of controlling the layout of the fibres results into the desired performance improvements in the sensing capability of the devices.

Chapter 5

5. Conclusions and future of the research

5.1. Conclusion:

As detailed in Section 2, the findings from this study present a compelling case for the development of innovative auditory devices in the foreseeable future. The presented device demonstrates pronounced frequency selectivity. Continued research holds the promise of refining the structure, drawing it closer in functionality to the human cochlea. This advancement would potentially enable users to discern a broader spectrum of sounds, encompassing intricate nuances in tunes and melodies, enriching their auditory experiences.

The showcased device not only suggests a future where cochlear implants are more bio-inspired and frequency-selective but also demonstrates vast potential as an acoustic sensor tailored for detailed diagnostics and failure prevention. For instance, integrating this device within offshore turbines could be revolutionary. Systems operating sub-optimally produce distinct sounds. By harnessing acoustic sensors in conjunction with spatial recognition and a neural network akin to the one used for speech recognition—but trained specifically on malfunction noises—there's potential to remotely pinpoint both the origin and nature of a failure. Such advancements could significantly reduce diagnostic and repair times, leading to substantial cost savings.

The main finding of this research project was proving that a single multichannel asymmetrical spiral sensor fabricated by UCL division of surgery group can be used for both localisation and speech recognition as opposed to a human cochlear that needs two healthy cochlear for sound source localisation. By investing in further research in this project, an acoustic sensor could be developed that can both identify and localise sound source which could be used as a non-invasive sensor in range different industries.

5.2. limitations:

The device in its current state is too large and fragile which makes it very difficult to handle and run test on. Additionally, due to the nature of the fabrication process of the sensors, each sensor is very different to another and it response could be vastly different as the fibre lengths and thickness could change in each fabrication attempt. Finally, the limitation of the data collection, training and testing testbed was that the maximum sampling rate of the data collection (the Arduino) is relatively low and makes it difficult to capture all the data and forces the experiment to focus on a very small portion of the audio range (only up to 1.5kHz as opposed to the 20kHz that humans can hear)

5.3. Future work:

The multichannel devices show an immense potential in revolutionising and advancing the cochlear implants. Both the material and the manufacturing can be optimised through by improving the speed of the manufacturing and adapting the axial Cartesian printer for a printer method that is better fit for fabricating radial devices.

Given that the majority of the devices that will be explored are inspired by human cochlear and have a circular structure, it would be interesting to explore a stage that can be controlled to move radially with while motion of the needle can be controlled in the horizontal direction. This method should be faster and allow for the speeds that are much more comparable to the ones used in the literature. The hypothesis is that because the motor used to move the needle holder is much more powerful motor with an adjustable height and the stepper motor can move much faster radially with the maximum speed which is about 50π rad/s (1500rpm), will allow for a much faster movement of the needle relative to the desired position on the stage.

Moreover, the literature demonstrates methods of controlling the desired position and path of the fibres on the electrode without using a moving stage. There are two

interesting fabrication method illustrated: 1) segmenting the collector electrode and using switches moving the printing location to the desired position [10] 2) Using 2 jet defecting electrodes to deflect the fibre to the desired position [11].

It is imperative for the success of this project to ensure that the fabrication method must provide complete control over the position and dimeter of the fibres over the electrode. Additionally, the electrode must have a constant dimeter throughout the length of the fibre and controllable. Therefore, in addition to optimising the printing parameter, the number of devices must be fabricated with each proposed method to evaluate their consistency and how well and fast each method performed the fabrication process. Also, either one of the mechanical printing methods can be combined with the motionless printing to improve the performance. This is because the combination could allow us to draw and stretch the fibre quickly which could lead to better control over the final dimeter of the fibre given that the feed rate through the nozzle is known.

Within the material research segment detailed in Section 4, foundational efforts were dedicated to the fabrication of hollow PVDF fibres with BTO nanoparticles centrally located. Moving forward, it becomes imperative to conduct comprehensive characterizations of these resultant fibres. This will facilitate a comparative analysis with the fibres highlighted in Section 2, offering a more holistic understanding of their properties and potential applications.

Refinement of the electrode's design and layout continues to be a focal area of research. The intricacies of the design play an instrumental role not only in facilitating an increase in channel capacity but also in substantially reducing the device's footprint. For this project to achieve its intended impact, meticulous design optimization is crucial. This would ensure that even within a compact form factor, the sensor can capture an optimal amount of information.

In subsequent research, a thorough examination of neuron cell toxicity and responses to the newly developed material needs to be undertaken. Additionally, there are plans

to evaluate ganglion neuron cells' reactions to the piezoelectric fibres, as well as the device's electrodes. This analysis will involve using a patch clamp under a microscope, with the intent to observe how these cells respond when stimulated by the device in reaction to various audio signals.

6. References

[1] FDA, "What is a Cochlear Implant?," 04 02 2018. [Online]. Available: https://www.fda.gov/medical-devices/cochlear-implants/what-cochlear-implant#:~:text=back%20to%20top%5D-,Who%20uses%20cochlear%20implants,to%20benefit%20from%20cochlear%20implants..

[2] N. L. E. D. M. R. M. R. M. P. H. B. K. Thompson, "Cochlear Implantation for Unilateral and Asymmetric Hearing Loss: Long-Term Subjective Benefit.," *The Laryngoscope,* 2023.

[3] H. Cullington, "BCIG Annual UK Data Collection 01/04/2021 - 31/03/2022," British Cochlear Implant Group, Bradford, 2021.

[4] British Cochlear Implant Group, "BCIG," Yorkshire Auditory Implant Service, [Online]. Available: https://www.bcig.org.uk/annual-uk-update/. [Accessed 14 02 2023].

[5] US National Institute of Health, *NIDCD Fact Sheet - Hearing and Balance,* US National Institute of Health, 2019.

[6] "Cochlear Implant Market Size, Share & Trends Analysis Report By Type of Fitting (Unilateral Implants, Bilateral Implants), By End Use (Adult, Pediatric), By Region, And Segment Forecasts, 2022 - 2030," Grand view research, United States, 2021.

[7] MIDCD | Hearing and Balance, "Cochlear Implants Fact sheet," NIH publication, 2021.

[8] G. M. Clark, "The multiple-channel cochlear implant: the interface between sound and the central nervous system for hearing, speech, and language in deaf people—a personal perspective," *Philosophical transactions of the Royal Society of London,* vol. 361, no. Series B, Biological sciences, pp. 791-810, 2006.

[9] H. Li, L. Helpard, J. Ekeroot, S. A. Rohani, N. Zhu, H. Rask-Andersen, H. Ladak and S. Agrawal, "Three-dimensional tonotopic mapping of the human cochlea based on synchrotron radiation phase-contrast imaging," *Scientific Reports,* vol. 11, no. 1, p. 4437, 2021.

[10] H. C. Stronks, A. L. Tops, P. Hehrmann, J. J. Briaire and J. H. M. Frijns, "Personalizing Transient Noise Reduction Algorithm Settings for Cochlear Implant Users," *Ear Hear,* vol. 42, no. 6, pp. 1602-1614, 2021.

[11] W. Yost, Fundamentals of Hearing: An Introduction, Bingley: Brill, 2013.

[12] J. C. Middlebrooks and D. M. Green, "Sound localization by human listeners," *Annual Review of Psychology,* vol. 42, no. 1, pp. 135-159, 1991.

[13] B. GROTHE, M. PECKA and D. McALPINE, "Mechanisms of Sound Localization in Mammals," *Physiological Reviews,* vol. 90, no. 3, pp. 983-1012, 2010.

[14] M. Vijaya, Piezoelectric Materials and Devices Applications in Engineering and Medical Sciences, Boca Raton: CRC Press, 2013.

[15] . P. Hoskins, K. Martin and A. Thrush, Diagnostic Ultrasound, Cambridge: Cambridge Uiversity Prss , 2010.

[16] K. NIghtingale, "Acoustic Radiation Force Impulse (ARFI) Imaging: a Review," *NIH,* vol. 7, no. 4, p. 328–339, 2011.

[17] B. İlik, A. Koyuncuoglu, Ö. Sardan-Sukas and H. Külah, "Thin film piezoelectric acoustic transducer for fully implantable cochlear implants," *Sensors and Actuators A: Physical,* vol. 280, pp. 38-46, 2018.

[18] T. Inaoka, H. Shintaku, T. Nakagawa, S. Kawano, H. Ogita, T. Sakamoto, S. Hamanishi, H. Wada and J. Ito, "Piezoelectric materials mimic the function of the cochlear sensory epithelium," *PNAS,* vol. 108, no. 45, p. 18390–18395, 2011.

[19] S. Park, X. Guan, Y. Kim and et al., "PVDF-Based Piezoelectric Microphone for Sound Detection Inside the Cochlea: Toward Totally Implantable Cochlear Implants," *Sage journals Trends in Hearing,* vol. 22, pp. 1-11, 2018.

[20] R. Pelrine, R. Kornbluh and G. Kofod, "High-Strain Actuator Materials Based on Dielectric Elastomers," *Advanced Materials,* vol. 12, no. 16, pp. 1223 - 1225, 2000.

[21] A. Arrigoni, L. Brambilla, C. Bertarelli, G. Serra, M. Tommasini and C. Castiglioni, "P(VDF-TrFE) nanofibers: structure of the ferroelectric and paraelectric phases through IR and Raman spectroscopies," *RSC Advances,* vol. 10, no. 62, p. 37779–37796, 2020.

[22] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, Boston: MIT Press, 2016.

[23] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," *Nature,* vol. 521, no. 7553, pp. 436-44, 2015.

[24] A. Alsobhani, H. M. A. A. Abboodi and H. Mahdi, "Speech Recognition using Convolution Deep," in *Journal of Physics: Conference Series*, 2021.

[25] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn and D. Yu, "Convolutional Neural Networks for Speech Recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing,* vol. 22, no. 10, pp. 1533 - 1545, 2014.

[26] C. Humphries, E. Liebenthal and J. Binder, "Tonotopic organization of human auditory cortex," *NeuroImage,* vol. 50, no. 3, pp. 1202-1211, 2010.

[27] I. Gebeshuber and F. Rattay, "Coding efficiency of inner hair cells at the threshold of hearing," in *Computational Models of Auditory Function*, Amesterdam, IOS Press, 2001, pp. 5-16.

[28] H. Heffner and R. Heffner, "Hearing Ranges of laboratory animals," *PubMed,* vol. 46, no. 1, pp. 20-22, 2007.

[29] G. Varallyay, S. V. Legarth and T. Ramirez, "Music Lovers and Hearing Aids," Audiology Online, 26 02 2016. [Online]. Available: https://www.audiologyonline.com/articles/music-lovers-and-hearing-aids-16478. [Accessed 30 08 2022].

[30] V. Giuseppe , C. Jinke , M. Thomas and et al., "Bioinspired Multiresonant Acoustic Devices Based on Electrospun Piezoelectric Polymeric Nanofibers," *American Chemical Society,* vol. 12, no. 31, p. 34643–34657, 2020.

[31] G. S. Nandini, S. Kumar and K. Chidananda , "Dropout technique for image classification based on extreme learning machine," *Global Transitions Proceedings,* vol. 2, no. 1, pp. 111-116, 2021.

[32] J.-C. Wang, H. Zheng, M.-W. Chang, Z. Ahmad and J.-S. Li, "Preparation of active 3D film patches via aligned fiber electrohydrodynamic (EHD) printing," *Scientific reports,* vol. 7, no. 1, p. 43924, 2017.

[33] C. Chenhao, L. Xinlin, X. Wei and et at, "Electrohydrodynamic printing for demanding devices: A review of processing and applications," *Nanotechnology Reviews,* vol. 11, no. 1, pp. 3305-3334, 2022.

[34] T. H. Hwang, Y. J. Kim, H. Chung and W. Ryu, "Motionless Electrohydrodynamic (EHD) Printing of Biodegradable Polymer Micro Patterns," *Microelectronic Engineering,* vol. 161, pp. 43-51, 2016.

[35] I. Liashenko, J. Rosell-Llompart and A. Cabot, "Ultrafast 3D printing with submicrometer features using electrostatic jet deflection," *Nature Communications,* vol. 11, no. 1, p. 753, 2020.

[36] W. Song, "A smart sensor that can be woven into everyday life," *Nature,* vol. 603, pp. 585-586, 2022.

[37] P. Filip and P. Peer, "Characterization of Poly(Ethylene Oxide) Nanofibers—Mutual Relations between Mean Diameter of Electrospun Nanofibers and Solution Characteristics," *Processes,* vol. 7, no. 12, p. 948, 2019.

[38] H. E. Schneider, J. Steuber, W. Du, M. Mortazavi and D. Bullock, "Polyethylene Oxide Nanofiber Production by Electrospinning," *Journal of the Arkansas Academy of Science,* vol. 70, no. 1, pp. 211-, 2016.

Appendix A

MATLAB code for initial speech recognition.

A.1 Main code:

```
%% Cleaning the cache
clc
clear
close all

% Openning and sorting files
filelist = dir("*.csv");
counterBegin = 0;
counterEnd   = 1;
counter = 0;
fs = 10000;
numberofTrials = 1;
TrainingSetCounter = 1;
TestSetCounter =1;
ValicationSetCounter = 1;
TrainingSet = 0;

for i =[1,2,3,6,8,9,10,12,14,15,16,17,18,19] % size(filelist, 1)
    counter = counter +1
    name = filelist(i, 1).name;
    TF = isstrprop(name,'alpha');
    file         = readmatrix(filelist(i, 1).name);
    % Finding the Angle from file name
    for k = 1:size(TF,2)
       numNull = 0;
       if TF(k) == 0
           Anglestr(k)= name(k);
       else
           break;
       end
    end
    if length(Anglestr)>3
        Angle        = str2num(Anglestr(1:3));
        clear k;
    else
        Angle        = str2num(Anglestr);
        clear k;
    end

    % Finiding the range and the peaks
    F = file(:,2);
    t = file(:,1);
    [maxPeak,maxLoc] = findpeaks(F,t,'MinPeakDistance',1,'Threshold',1e-5);
    [minPeak,minLoc] = findpeaks(-F,t,'MinPeakDistance',3.6);

    minPeak = -1 * minPeak;

    minCorr = minPeak(minPeak>100);
    maxCorr = maxPeak(maxPeak<1600);

    if size(minCorr,1) > 0
        for countOutMin=1:size(minCorr,1)
            outliermin = find(minPeak>100);
            minLoc(outliermin) = [];
            minPeak(minPeak>100) = [];
        end
    else
        minLoc;
    end
```

```
if size(maxCorr,1) > 0
    for countOutMax=1:size(maxCorr,1)
        outliermax = find(maxPeak<1600);
        maxLoc(outliermax) = [];
        maxPeak(maxPeak<1600) = [];
    end
else
    maxLoc;
end

if minLoc(1) < maxLoc(1)
    minLoc(1) = [];
    minPeak(1) = [];
else
    minPeak;
end

if size(minLoc,1) < size(maxLoc,1)
    maxLoc(end) = [];
    maxPeak(end) = [];
else
    maxPeak;
end
clear countOutMax countOutMin minCorr maxCorr outliermin outliermax;

data_start = find(t == maxLoc(1));

% Universal Counter fror data
counterBegin = counterEnd;
counterEnd   = counterEnd + size(file(data_start:end,1),1)-1;

% Finding the trial starting point for segmentation
SamplePoint = size(minPeak,1);

Correction      = ones(size(SamplePoint));
Correction(1)   = 0;

% Spliting data into training, test and validation
TrainingSet = TrainingSet + size(minLoc,1);

% Filtering the Data
Ch_Filtered(:,1) = bandpass(file(data_start:end,3),[95 1505],fs); % Channel 1
Ch_Filtered(:,2) = bandpass(file(data_start:end,4),[95 1505],fs); % Channel 2
Ch_Filtered(:,3) = bandpass(file(data_start:end,5),[95 1505],fs); % Channel 3
Ch_Filtered(:,4) = bandpass(file(data_start:end,6),[95 1505],fs); % Channel 4

% Making the training, testing and validating set
for pointcount=1:size(minLoc)
    startPoint(pointcount,1) = find(t == maxLoc(pointcount))- data_start+1;
    endPoint(pointcount,1) = find(t == minLoc(pointcount))- data_start+1;
end
o = 1;
for n = TrainingSetCounter:TrainingSet
DataTraining(1, :) = Ch_Filtered(startPoint(o):endPoint(o),1);
DataTraining(2, :) = Ch_Filtered(startPoint(o):endPoint(o),2);  % Channel 2
DataTraining(3, :) = Ch_Filtered(startPoint(o):endPoint(o),3);  % Channel 3
DataTraining(4, :) = Ch_Filtered(startPoint(o):endPoint(o),4);  % Channel 4
```

```
    DataTrainingTarget(1, n) = Angle;

    XtestPiezo(n,1)= {DataTraining};
    clear DataTraining;
    o = o+1;
    end
    TrainingSetCounter   = TrainingSet+1;


    clear l  Anglestr Ch_Filtered Angle n i m TF;


end

%%
for i =7 % size(filelist, 1)
    counter = counter +1
    name = filelist(i, 1).name;
    TF = isstrprop(name,'alpha');
    file         = readmatrix(filelist(i, 1).name);
    % Finding the Angle from file name
    for k = 1:size(TF,2)
       numNull = 0;
       if TF(k) == 0
           Anglestr(k)= name(k);
       else
           break;
       end
    end
    Angle        = str2num(Anglestr(1:3));
    clear k;


    % Finiding the range and the peaks
    F = file(:,2);
    t = file(:,1);
    [maxPeak,maxLoc] = findpeaks(F,t,'MinPeakDistance',2);
    [minPeak,minLoc] = findpeaks(-F,t,'MinPeakDistance',3.6);

    minPeak = -1 * minPeak;

    minCorr = minPeak(minPeak>10);
    maxCorr = maxPeak(maxPeak<490);

    if size(minCorr,1) > 0
        for countOutMin=1:size(minCorr,1)
            outliermin = find(minPeak>110);
            minLoc(outliermin) = [];
            minPeak(minPeak>110) = [];
        end
    else
        minLoc;
    end

    if size(maxCorr,1) > 0
        for countOutMax=1:size(maxCorr,1)
            outliermax = find(maxPeak<490);
            maxLoc(outliermax) = [];
            maxPeak(maxPeak<490) = [];
```

```
    end
else
    maxLoc;
end

if minLoc(1) < maxLoc(1)
    minLoc(1) = [];
    minPeak(1) = [];
else
    minPeak;
end

if size(minLoc,1) < size(maxLoc,1)
    maxLoc(end) = [];
    maxPeak(end) = [];
else
    maxPeak;
end
clear countOutMax countOutMin minCorr maxCorr outliermin outliermax;

data_start = find(t == maxLoc(1));

% Universal Counter fror data
counterBegin = counterEnd;
counterEnd   = counterEnd + size(file(data_start:end,1),1)-1;

% Finding the trial starting point for segmentation
SamplePoint = size(minPeak,1);

Correction      = ones(size(SamplePoint));
Correction(1)   = 0;

% Spliting data into training, test and validation
TrainingSet = TrainingSet + size(minLoc,1);

% Filtering the Data
Ch_Filtered(:,1) = bandpass(file(data_start:end,3),[95 1505],fs); % Channel 1
Ch_Filtered(:,2) = bandpass(file(data_start:end,4),[95 1505],fs); % Channel 2
Ch_Filtered(:,3) = bandpass(file(data_start:end,5),[95 1505],fs); % Channel 3
Ch_Filtered(:,4) = bandpass(file(data_start:end,6),[95 1505],fs); % Channel 4

% Making the training, testing and validating set
for pointcount=1:size(minLoc)
    startPoint(pointcount,1) = find(t == maxLoc(pointcount))- data_start+1;
    endPoint(pointcount,1) = find(t == minLoc(pointcount))- data_start+1;
end
o = 1;
for n = TrainingSetCounter:TrainingSet

DataTraining(1, :) = Ch_Filtered(startPoint(o):endPoint(o),1); % Channel 1
DataTraining(2, :) = Ch_Filtered(startPoint(o):endPoint(o),2); % Channel 2
DataTraining(3, :) = Ch_Filtered(startPoint(o):endPoint(o),3); % Channel 3
DataTraining(4, :) = Ch_Filtered(startPoint(o):endPoint(o),4); % Channel 4

DataTrainingTarget(1, n) = Angle;

XtestPiezo(n,1)= {DataTraining};
clear DataTraining;
o = o+1;
```

```
    end
    TrainingSetCounter    = TrainingSet+1;


    clear l  Anglestr Ch_Filtered Angle n i m TF;

end
%% 100 to 1500
for i =[4, 11] % size(filelist, 1)
    counter = counter +1
    name = filelist(i, 1).name;
    TF = isstrprop(name,'alpha');
    file          = readmatrix(filelist(i, 1).name);
    % Finding the Angle from file name
    for k = 1:size(TF,2)
       numNull = 0;
       if TF(k) == 0
           Anglestr(k)= name(k);
       else
           break;
       end
    end
    Angle         = str2num(Anglestr(1:3));
    clear k;


    % Finiding the range and the peaks
    F = file(:,2);
    t = file(:,1);
    [maxPeak,maxLoc] = findpeaks(F,t,'MinPeakDistance',1,'Threshold',1e-5);
    [minPeak,minLoc] = findpeaks(-F,t,'MinPeakDistance',3.6);

    minPeak = -1 * minPeak;

    minCorr = minPeak(minPeak>100);
    maxCorr = maxPeak(maxPeak<1500);

    if size(minCorr,1) > 0
        for countOutMin=1:size(minCorr,1)
            outliermin = find(minPeak>100);
            minLoc(outliermin) = [];
            minPeak(minPeak>100) = [];
        end
    else
        minLoc;
    end

    if size(maxCorr,1) > 0
        for countOutMax=1:size(maxCorr,1)
            outliermax = find(maxPeak<1500);
            maxLoc(outliermax) = [];
            maxPeak(maxPeak<1500) = [];
        end
    else
        maxLoc;
    end

    if maxLoc(1) < minLoc(1)
        maxLoc(1) = [];
```

```
        maxPeak(1) = [];
    else
        maxPeak;
    end

    if size(maxLoc,1) < size(minLoc,1)
        minLoc(end) = [];
        minPeak(end) = [];
    else
        minPeak;
    end
    clear countOutMax countOutMin minCorr maxCorr outliermin outliermax;

    data_start = find(t == minLoc(1));

    % Universal Counter fror data
    counterBegin = counterEnd;
    counterEnd   = counterEnd + size(file(data_start:end,1),1)-1;

    % Finding the trial starting point for segmentation
    SamplePoint = size(minPeak,1);

    Correction      = ones(size(SamplePoint));
    Correction(1)   = 0;

    % Spliting data into training, test and validation
    TrainingSet = TrainingSet + size(minLoc,1);

    % Filtering the Data
    Ch_Filtered(:,1) = bandpass(file(data_start:end,3),[95 1505],fs); % Channel 1
    Ch_Filtered(:,2) = bandpass(file(data_start:end,4),[95 1505],fs); % Channel 2
    Ch_Filtered(:,3) = bandpass(file(data_start:end,5),[95 1505],fs); % Channel 3
    Ch_Filtered(:,4) = bandpass(file(data_start:end,6),[95 1505],fs); % Channel 4

    % Making the training, testing and validating set
    for pointcount=1:size(minLoc)
        startPoint(pointcount,1) = find(t == minLoc(pointcount))- data_start+1;
        endPoint(pointcount,1) = find(t == maxLoc(pointcount))- data_start+1;
    end
    o = 1;
    for n = TrainingSetCounter:TrainingSet

    DataTraining(1, :) = Ch_Filtered(startPoint(o):endPoint(o),1); % Channel 1
    DataTraining(2, :) = Ch_Filtered(startPoint(o):endPoint(o),2); % Channel 2
    DataTraining(3, :) = Ch_Filtered(startPoint(o):endPoint(o),3); % Channel 3
    DataTraining(4, :) = Ch_Filtered(startPoint(o):endPoint(o),4); % Channel 4

    DataTrainingTarget(1, n) = Angle;

    XtestPiezo(n,1)= {DataTraining};
    clear DataTraining;
    o = o+1;
    end
    TrainingSetCounter   = TrainingSet+1;

    clear l  Anglestr Ch_Filtered Angle n i m TF;


end
```

```
        minLoc(end) = [];
        minPeak(end) = [];
    else
        minPeak;
    end
    clear countOutMax countOutMin minCorr maxCorr outliermin outliermax;

    data_start = find(t == minLoc(1));

    % Universal Counter fror data
    counterBegin = counterEnd;
    counterEnd   = counterEnd + size(file(data_start:end,1),1)-1;

    % Finding the trial starting point for segmentation
    SamplePoint = size(minPeak,1);

    Correction      = ones(size(SamplePoint));
    Correction(1)   = 0;

    % Spliting data into training, test and validation
    TrainingSet = TrainingSet + size(minLoc,1);

    % Filtering the Data
    Ch_Filtered(:,1) = bandpass(file(data_start:end,3),[95 1505],fs);
    Ch_Filtered(:,2) = bandpass(file(data_start:end,4),[95 1505],fs);
    Ch_Filtered(:,3) = bandpass(file(data_start:end,5),[95 1505],fs);
    Ch_Filtered(:,4) = bandpass(file(data_start:end,6),[95 1505],fs);

    % Making the training, testing and validating set
    for pointcount=1:size(minLoc)
        startPoint(pointcount,1) = find(t == minLoc(pointcount))- data_start+1;
        endPoint(pointcount,1) = find(t == maxLoc(pointcount))- data_start+1;
    end
    o = 1;
    for n = TrainingSetCounter:TrainingSet
    DataTraining(1, :) = Ch_Filtered(startPoint(o):endPoint(o),1);  % Channel 1
    DataTraining(2, :) = Ch_Filtered(startPoint(o):endPoint(o),2);  % Channel 2
    DataTraining(3, :) = Ch_Filtered(startPoint(o):endPoint(o),3);  % Channel 3
    DataTraining(4, :) = Ch_Filtered(startPoint(o):endPoint(o),4);  % Channel 4


    DataTrainingTarget(1, n) = Angle;

    XtestPiezo(n,1)= {DataTraining};
    clear DataTraining;
    o = o+1;
    end
    TrainingSetCounter   = TrainingSet+1;


    clear l  Anglestr Ch_Filtered Angle n i m TF;


end
YtestPiezo = categorical(DataTrainingTarget');
YtestPiezo = removecats(YtestPiezo);
save('TestingData1', 'XtestPiezo', 'YtestPiezo');
```

A.2 Creating the images for CNN

```
function helperGenerateTFDfilesCombined(parentDir,dataDir,wav,truth)

        [~,~,~] = mkdir(fullfile(parentDir,dataDir));
        modTypes = unique(truth);

        for idxM = 1:length(modTypes)
            modType = modTypes(idxM);
            [~,~,~] = mkdir(fullfile(parentDir,dataDir,char(modType)));
        end

        for idxW = 1:length(truth)
           sig1 = wav{idxW}(1,:);
           sig2 = wav{idxW}(2,:);
           sig3 = wav{idxW}(3,:);
           sig4 = wav{idxW}(4,:);

           nfft = 2^nextpow2(length(sig1(1,:)));
           f = (0:(nfft/2-1))/nfft*10000;

           Z1 = fft(sig1(1,:),nfft);
           Z2 = fft(sig2(1,:),nfft);
           Z3 = fft(sig3(1,:),nfft);
           Z4 = fft(sig4(1,:),nfft);

           amp1 = abs(Z1(1:4966))./max(abs(Z1));
           amp2 = abs(Z2(1:4966))./max(abs(Z2));
           amp3 = abs(Z3(1:4966))./max(abs(Z3));
           amp4 = abs(Z4(1:4966))./max(abs(Z4));

           % Finding FFT peaks
           % Channel 1
           [maxPeak1,maxLoc1] =
               findpeaks(amp1,f(1:4966),'MinPeakDistance',3,'Threshold',1e-3);
           % Channel 2
           [maxPeak2,maxLoc2] =
               findpeaks(amp2,f(1:4966),'MinPeakDistance',3,'Threshold',1e-3);
           % Channel 3
           [maxPeak3,maxLoc3] =
               findpeaks(amp3,f(1:4966),'MinPeakDistance',3,'Threshold',1e-3);
           % Channel 4
           [maxPeak4,maxLoc4] =
               findpeaks(amp4,f(1:4966),'MinPeakDistance',3,'Threshold',1e-3);

           % Creating scatter diagram using FFT data
           s1 = scatter(maxLoc1,maxPeak1,'k','filled');
           TFD1 = getframe;

           s2 = scatter(maxLoc2,maxPeak2,'k','filled');
           TFD2 = getframe;

           s3 = scatter(maxLoc3,maxPeak3,'k','filled');
           TFD3 = getframe;

           s4 = scatter(maxLoc4,maxPeak4,'k','filled');
           TFD4 = getframe;
```

```
        % Combining the 4 images into 1
        multi1 = cat(2,TFD1.cdata,TFD2.cdata);
        multi2 = cat(2,TFD3.cdata,TFD4.cdata);
        multi  = cat(1,multi1,multi2);

        TFD = imresize(multi,[227 227]);
        TFD = rescale(TFD);
        TFD = imcomplement(TFD);

        modType = truth(idxW);

   imwrite(TFD,fullfile(parentDir,dataDir,char(modType),sprintf('%d.png',idxW)))


end
end
```

A.3 creating temp files for Training, Testing and Validation the neural network

```
clc
clear

%% Create Directory Training sets
load('TestingData1.mat');

XTrain = XtestPiezo;
YTrain = YtestPiezo;

parentDir = tempdir;
dataDir = 'TFDDatabaseFFT1';

%helperGenerateTFDfilesCombined(parentDir,dataDir,XTrain,YTrain,10e5)

helperFFTGenerateTFDfilesCombined(parentDir,dataDir,XTrain,YTrain)



%% Create Image Store database

folder = fullfile(parentDir,dataDir,{'0','30','60','90', '120', '150',...
                                     '180','210','240','270', '300', '330'});
imds = imageDatastore(folder,...

'FileExtensions','.png','LabelSource','foldernames','ReadFcn',@readTFDForSqueezeNe
t);

[imdsTrain,imdsTest,imdsValidation] = splitEachLabel(imds,0.8,0.1);
```

Appendix B:

Arduino multichannel data collection code.

GitHub link:

https://github.com/abmoineddini/MPhil_sound_localisation/tree/main/Hardware_controllers/analog_data_collection

```
/* Arduino Mega Data Collection
 * by: Amirbahador Moineddini
 * date: November 10th, 2021
 * V1 Data collection


/* Pin Setup
 * A5  Channel 1
 * A8  Channel 2
 * A12 Channel 3
 * A14 Channel 4
*/

void setup() {
  // put your setup code here, to run once:
  Serial.begin(2000000);
  pinMode(A5, INPUT);  // Channel 1
  pinMode(A8, INPUT);  // Channel 2
  pinMode(A12, INPUT); // Channel 3
  pinMode(A14, INPUT); // Channel 4

}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.print(analogRead(A5));
  Serial.print(",");
  Serial.print(analogRead(A8));
  Serial.print(",");
  Serial.print(analogRead(A12));
  Serial.print(",");
  Serial.println(analogRead(A12));

}
```

Appendix C

Python codes for data collection, preprocessing the data, plotting figures and training, testing and validation of the *Speech Recognition*.

GitHub link: https://github.com/abmoineddini/MPhil_speech_recognition

C.1. main.py code:

```
from DataCollector import *
import matplotlib.pyplot as plt
import os
import pandas as pd
from os import listdir
from DataCollector import playAudioFile
import winsound
import glob
from PreprocessorSpeechrecognition import figure_maker, Classifier,
PreprocessingMeth2
from csv import writer


dataCollection = True
CheckPort = input("Would you like to Start collecting Data : ")
if CheckPort == "y" or CheckPort == "Y" or CheckPort == "Yes" or CheckPort ==
"yes":
    COMPort = input("Please Enter COM Port: ")
    print(COMPort)
    CheckPort = input("Is that Correct?")
    Method = input("Continious or Individual Collections? ")

    while dataCollection:
        if Method == 1:
            AudioFiles = [f for f in listdir("AudioFiles/")]
            for i in AudioFiles:
                Name = i
                print(Name)
                print("here")
                Period = 30
                print("Startting to Collect Data for: ", Period, '(s)')
                CheckPeriod = input("Is that Correct?")
                if CheckPeriod == "n" or CheckPeriod == "N" or CheckPeriod ==
"no" or CheckPeriod == "No":
                    Period = input("Please Enter the Desired Time Period: ")
                    print(Period)
                    print("Starting to Collect Data for: ", Period, '(s)')
                    CheckName = input("Is that Correct?")

                AudioFileName = "AudioFiles/" + Name
                winsound.PlaySound(AudioFileName, winsound.SND_ASYNC |
winsound.SND_ALIAS)
                CSVName = Name.replace('.wav', '')
                NameTest = True
                TrainingDataDirectory = [f for f in listdir("TrainingData")]
```

```
                testNum = 1
                while NameTest:
                    CSVNameCheck = CSVName + '-Test'+ str(testNum) + '.csv'
                    if CSVNameCheck in TrainingDataDirectory:
                        print("File Already exist, Trying another name.")
                        testNum = testNum+1
                    else:
                        CSVName = CSVName + '-Test'+ str(testNum)
                        NameTest = False
                print(CSVName)
                [Channel1, Time] = collectData(COMPort, Period, CSVName)
                plt.plot(Time, Channel1)
                plt.show()
                winsound.PlaySound(None, winsound.SND_PURGE)
                DataCheck = input("Are you happy with the Data? ")
                while DataCheck == "n" or DataCheck == "N" or DataCheck ==
"no" or DataCheck == "No":
                    Nametoremove = "TrainingData/" + CSVName + ".csv"
                    os.remove(Nametoremove)


                    winsound.PlaySound(AudioFileName, winsound.SND_ASYNC |
winsound.SND_ALIAS)
                    [Channel1, Time] = collectData(COMPort, Period, CSVName)
                    plt.plot(Time, Channel1)
                    plt.show()
                    winsound.PlaySound(None, winsound.SND_PURGE)
                    DataCheck = input("Are you happy with the Data? ")

        else:
            AudioFiles = [f for f in listdir("AudioOriginal")]
            for i in AudioFiles:
                Name = i
                print(Name)
                print("here")

                AudioFileName = "AudioOriginal/" + Name
                CSVName = Name.replace('.wav', '')
                NameTest = True
                TrainingDataDirectory = [f for f in listdir("TrainingData")]
                testNum = 1

                while NameTest:
                    CSVNameCheck = CSVName + '-Test'+ str(testNum) + '.csv'
                    if CSVNameCheck in TrainingDataDirectory:
                        print("File Already exist, Trying another name.")
                        testNum = testNum+1
                    else:
                        CSVName = CSVName + '-Test'+ str(testNum)
```

```
                        NameTest = False
                print(CSVName)
                [Channel1, Time] = collectDataMet2(COMPort, CSVName,
AudioFileName)
                plt.plot(Time, Channel1)
                plt.show()
                winsound.PlaySound(None, winsound.SND_PURGE)
                DataCheck = 'y'
                while DataCheck == "n" or DataCheck == "N" or DataCheck ==
"no" or DataCheck == "No":
                    Nametoremove = "TrainingData/" + CSVName + ".csv"
                    os.remove(Nametoremove)


                    winsound.PlaySound(AudioFileName, winsound.SND_ASYNC |
winsound.SND_ALIAS)
                    [Channel1, Time] = collectDataMet2(COMPort, CSVName,
AudioFileName)
                    plt.plot(Time, Channel1)
                    plt.show()
                    winsound.PlaySound(None, winsound.SND_PURGE)
                    DataCheck = input("Are you happy with the Data? ")

        ToContinue = 'y'
        if ToContinue == "n" or ToContinue == "N" or ToContinue == "No" or
ToContinue == "no":
            dataCollection = False

StartPreprocessing = input("Should I Start the Preprocessing? ")
if StartPreprocessing == "y" or StartPreprocessing == "Y" or
StartPreprocessing == "Yes" or StartPreprocessing == "yes":

    if os.path.isdir("Figure/Training"):
        print("Adding to Figure Directory")
    else:
        os.mkdir("Figure/Training")
        os.mkdir("Figure/Testing")

    csv_files = glob.glob(os.path.join("TrainingData/", "*.csv"))

    for x in csv_files[0:]:
        dataFileName = x
        print(x)
        NAMEunprocessed = x.replace(".csv", "")
        NAMEunprocessed = NAMEunprocessed.split("\\")
        NAMEunprocessed = NAMEunprocessed[1]
        NAMEunprocessed = NAMEunprocessed.split("-")
        x = NAMEunprocessed[0]
        x = x[2:len(x)]
```

```
        check = 0
        if os.path.isfile("ProcessedData.csv"):
            print("Processed data collector Already exists")

            PDCL = pd.read_csv("ProcessedData.csv")
            print(PDCL)
            ProcessedDataChecklist = PDCL.to_numpy()
            ProcessedDataChecklist = ProcessedDataChecklist[:]

            if dataFileName in ProcessedDataChecklist:
                check = 1
                continue

        if check == 0:
            [FolderSTFTTraining, FolderSTFTTesting] = Classifier(x)

            df = pd.read_csv(dataFileName)
            data = df.to_numpy()
            print(dataFileName)

            if len(data)>40000:
                figure_maker(data, FolderSTFTTraining, FolderSTFTTesting,
dataFileName)
            else:
                PreprocessingMeth2(data, FolderSTFTTraining,
FolderSTFTTesting, dataFileName)

            if os.path.isfile("ProcessedData.csv"):
                with open("ProcessedData.csv", 'a+' ,newline='') as f_object:
                    writer_object = writer(f_object)
                    writer_object.writerow([dataFileName])
                    f_object.close()
            else:
                dict = {"File Name": [dataFileName]}
                df = pd.DataFrame(dict)
                df.to_csv("ProcessedData.csv")

print("Finished Creating Relevant Files")


StartTraining = input("Should I Start the Training? ")
while StartTraining == "n" or StartTraining == "N" or StartTraining == "no" or
StartTraining == "No":
    StartTraining = input("Should I Start the Training? ")


trainingSTFT = "Figure/Training"
testingSTFT = "Figure/Testing"
```

```
from MachineLearning import *
img_size = 64
# STFT Training
[STFTModel, acc, val_acc, loss, val_loss]= CNN_Training(trainingSTFT,
testingSTFT, 150, LearningRate=0.0001, dataType="STFT", img_size=img_size)

STFTModel.save("VoiceRecCNN")

for i in range(len(acc)):
    FileAdd = [acc[i], val_acc[i], loss[i], val_loss[i]]
    with open("AccuracyHistory.csv", 'a+', newline='') as f_object:
        writer_object = writer(f_object)
        writer_object.writerow(FileAdd)
        f_object.close()
        FileAdd = []

# Testinig Peformance
print("STFT CNN Test result")
[y_val, predictions] = TestingNetwrok(STFTModel, testingSTFT, img_size)

for i in range(len(y_val)):
    FileAdd = [y_val[i], predictions[i]]
    with open("TestingValidationCNN.csv", 'a+', newline='') as f_object:
        writer_object = writer(f_object)
        writer_object.writerow(FileAdd)
        f_object.close()
        FileAdd = []



STFTmodelName = 'STFTModel.yaml'

Save_CNN(STFTModel, Name=STFTmodelName)
```

C.2. DataCollector.py

```
import serial
import winsound
import pandas as pd

def playAudioFile(Name):
    winsound.PlaySound(Name, winsound.SND_ASYNC | winsound.SND_ALIAS )

def collectData(COMPort, Period, Name):
    arduino = serial.Serial(COMPort , 2000000, timeout=1)
    Channel1 = []

    Time = []
    #def animate(xVal, yVal):
    period = int(Period)
    period = int(period*4000/3.5)
    for i in range(period):
        line = arduino.readline()

        if line != (''):
            print(line)
            try:
                string = line.decode()
            except:
                print("ignored")
            else:
                numS = string.replace("\r\n", '')
                if numS.split(" ")[0].isdigit():
                    Channel1.append(int(numS.split(" ")[0]))
                    Time.append(i/4000/3.5)
        # print(i)
    arduino.close()
    import pandas as pd
    dict = {'Time (s)' : Time, 'Channel 1 (V)': Channel1}#,

    df = pd.DataFrame(dict)
    dataBaseName = "TrainingData/" + Name + ".csv"
    df.to_csv(dataBaseName)

    return [Channel1, Time]



import time

def collectDataMet2(COMPort, Name, AudioFileName):
    Voltage = []
    Time = []
    dataCollection = False
    dataAvailability = True
```

```
sR = 4000/3.5
i = 0
j = 0
arduino = serial.Serial(COMPort , 2000000, timeout=1)
line = arduino.readline()
time.sleep(1)
winsound.PlaySound(AudioFileName, winsound.SND_ASYNC | winsound.SND_ALIAS)
while dataAvailability:
    line = arduino.readline()
    if line != (''):
        print("start collecting")
        print(line)
        try:
            string = line.decode()
        except:
            print("ignored")
        else:
            numS = string.replace("\r\n", '')
            numS = numS.split(" ")
            numS = numS[0]
            if numS.isdigit():
                print(int(numS))
                if int(numS) < 300:
                    dataCollection = False
                else:
                    print("Trigger")
                    dataCollection = True
                    Voltage.append(int(numS))
                    Time.append(i / sR)
                    i = i + 1
                    j = 5000/2
    while dataCollection:
        line = arduino.readline()
        print(line)
        print("collecting Data")
        print(line)
        if line != (''):
            try:
                string = line.decode()
            except:
                print("ignored")
            else:
                numS = string.replace("\r\n", '')
                numS= numS.split(" ")
                numS = numS[0]
                print(numS)
                if numS.isdigit():
                    if int(numS) < 300:
```

```
                        j = j - 1
                    else:
                        pass
                    Voltage.append(int(numS))
                    print(numS)
                    Time.append(i / sR)
                    i = i + 1

                    if j < 1:
                        dataCollection = False
                        dataAvailability = False


    arduino.close()
    dict = {'Time (s)': Time, 'Channel 1 (V)': Voltage}


    df = pd.DataFrame(dict)
    dataBaseName = "TrainingData/" + Name + ".csv"
    df.to_csv(dataBaseName)
    print(dataBaseName)
    return Voltage, Time
```

C.3. preprocessingSpeechrecognition.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as sig
from SectionCutting import *
import os
from os import listdir
from os.path import isdir
from PIL import Image


def figure_maker(data, FolderSTFTTraining, FolderSTFTTesting, FileName):
    time = data[75:len(data)-1, 1]
    voltage = data[75:len(data)-1, 2]
    Fs = 4000
    detenV = sig.detrend(voltage)
    filter = sig.butter(2, [95, 1500], 'bandpass', fs=4000, output='sos')
    corrVoltage = sig.sosfilt(filter, detenV)
    corrVoltage = (corrVoltage*5)/1023
    TrainingDirectory = [f for f in listdir(FolderSTFTTraining)]
    TestDirectory = [f for f in listdir(FolderSTFTTesting)]

    countTrain = int(len(TrainingDirectory))
    countTest = int(len(TestDirectory))
    maxV = max(corrVoltage)
    normV = corrVoltage/maxV
    seperationPoints = SectionCutting(normV, time, FileName)

    N = len(seperationPoints)
    Arr = np.arange(N)
    np.random.shuffle(Arr)
    Training = Arr[:round(N*0.8)]

    for i in range(1,N-1):
        Vshow = normV[seperationPoints[i]:seperationPoints[i+1]]

        fig0 = plt.figure()
        ax0 = plt.Axes(fig0, [0., 0., 1., 1.])
        plt.style.use('dark_background')
        plt.scatter(time[seperationPoints[i]:seperationPoints[i+1]], Vshow,
c=abs(Vshow*Vshow), s=abs(Vshow))
        plt.gray()
        plt.axis('off')
        ax0.set_axis_off()
        plt.tight_layout()
        plt.show()
```

```
        fig0.savefig('ScattTestingFigure.png', bbox_inches='tight',
pad_inches=0)

        f1, t1, Zxx1 = sig.stft(Vshow, Fs, nperseg=1000)
        fig1 = plt.figure(frameon=False)
        ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
        plt.pcolormesh(t1, f1, np.abs(Zxx1), shading='gouraud', cmap='gray')
        plt.axis('off')
        plt.ylim([90, 500])
        ax1.set_axis_off()
        plt.tight_layout()
        plt.show()
        fig1.savefig('STFTTestingFigure.png', bbox_inches='tight',
pad_inches=0)

        def ProcessIm(Im1, Im2, fileName):
            image1 = Image.open(Im1)
            image1 = image1.rotate(90)

            image2 = Image.open(Im2)
            image1_size = image1.size
            new_image = Image.new('RGB', (2 * image1_size[0], image1_size[1]),
(250, 250, 250))
            new_image.paste(image1, (0, 0))
            new_image.paste(image2, (image1_size[0], 0))
            new_image.save(fileName)

        if i in Training:
            nameTrain = str(countTrain)
            countTrain += 1
            FileName = FolderSTFTTraining + "/" + nameTrain + '.png'
            print(FileName)
            ProcessIm('STFTTestingFigure.png', 'ScattTestingFigure.png',
FileName)

        else:
            nameTest = str(countTest)
            countTest += 1
            FileName = FolderSTFTTesting + "/" + nameTest + '.png'
            print(FileName)
            ProcessIm('STFTTestingFigure.png', 'ScattTestingFigure.png',
FileName)

        os.remove("STFTTestingFigure.png")
        os.remove("ScattTestingFigure.png")

def Classifier(x):
    classification = x
```

```
    FolderSTFTTraining = "Figure/Training/"+ classification
    FolderSTFTTesting = "Figure/Testing/"+ classification

    if isdir(FolderSTFTTraining):
        print("Folders Already Excits!")
    else:
        os.mkdir(FolderSTFTTraining)
        os.mkdir(FolderSTFTTesting)
    print("Folder for", classification, "made!")
    return [FolderSTFTTraining, FolderSTFTTesting]

def PreprocessingMeth2(data, FolderSTFTTraining, FolderSTFTTesting,
dataFileName):
    time = data[0:len(data)-4000, 1]
    voltage = data[0:len(data)-4000, 2]
    Fs = 4000

    TrainingDirectory = [f for f in listdir(FolderSTFTTraining)]
    TestDirectory = [f for f in listdir(FolderSTFTTesting)]
    detenV = sig.detrend(voltage)

    filter = sig.butter(2, [95, 1500], 'bandpass', fs=4000, output='sos')
    corrVoltage = sig.sosfilt(filter, detenV)
    corrVoltage = (corrVoltage*5)/1023

    maxV = max(corrVoltage)
    normV = corrVoltage/maxV

    countTrain = int(len(TrainingDirectory))
    countTest = int(len(TestDirectory))

    fig0 = plt.figure()
    ax0 = plt.Axes(fig0, [0., 0., 1., 1.])
    plt.style.use('dark_background')
    plt.scatter(time, normV, c=abs(normV * normV), s=abs(normV))
    plt.gray()
    plt.axis('off')
    ax0.set_axis_off()
    plt.tight_layout()
    plt.show()
    fig0.savefig('ScattTestingFigure.png', bbox_inches='tight', pad_inches=0)

    f1, t1, Zxx1 = sig.stft(normV, Fs, nperseg=1000)
    fig1 = plt.figure(frameon=False)
    ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
    plt.pcolormesh(t1, f1, np.abs(Zxx1), shading='gouraud', cmap='gray')
    plt.axis('off')
    plt.ylim([90, 500])
```

```
    ax1.set_axis_off()
    plt.tight_layout()
    plt.show()
    fig1.savefig('STFTTestingFigure.png', bbox_inches='tight', pad_inches=0)

    Training = [1, 3, 4]

    def ProcessIm(Im1, Im2, fileName):
        image1 = Image.open(Im1)
        image1 = image1.rotate(90)

        image2 = Image.open(Im2)
        image1_size = image1.size
        new_image = Image.new('RGB', (2 * image1_size[0], image1_size[1]),
(250, 250, 250))
        new_image.paste(image1, (0, 0))
        new_image.paste(image2, (image1_size[0], 0))
        new_image.save(fileName)


    i = randint(1, 4)
    if i in Training:
        nameTrain = str(countTrain)
        countTrain += 1
        FileName = FolderSTFTTraining + "/" + nameTrain + '.png'
        print(FileName)
        ProcessIm('STFTTestingFigure.png', 'ScattTestingFigure.png', FileName)

    else:
        nameTest = str(countTest)
        countTest += 1
        FileName = FolderSTFTTesting + "/" + nameTest + '.png'
        print(FileName)
        ProcessIm('STFTTestingFigure.png', 'ScattTestingFigure.png', FileName)

    os.remove("STFTTestingFigure.png")
    os.remove("ScattTestingFigure.png")
```

C.4. SectionCutting.py

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import scipy.signal as sig
from random import randint

def SectionCutting(V, t, FileName):
    seperation = []
    seperation.append(0)
    time = t
    voltage = V

    uplowBoundNull = 0.1
    perofMaxVal = 0.7
    binSize = 100
    perNullDen = 0.6
    perMaxDen = 0.7


    t = time[0: 70000]
    fderenV = V[0: 70000]

    maxVal = max(fderenV)

    Nullpeaks, _ = sig.find_peaks(fderenV, height=(-uplowBoundNull,
uplowBoundNull))
    # smallPeaks, _ = sig.find_peaks(fderenV, distance= 4000, height=(0.01,
0.05))
    peaks, _ = sig.find_peaks(fderenV, height=perofMaxVal*maxVal)


    f, (a0, a1, a2) = plt.subplots(3, 1, gridspec_kw={'height_ratios':
[10,3,3]})

    #plt.subplot(3, 1, 1)
    a0.plot(t, fderenV, linewidth=0.1, label="Signal")
    a0.scatter(t[Nullpeaks], fderenV[Nullpeaks], color='red', label="Low
amplitude peaks")
    a0.scatter(t[peaks], fderenV[peaks], color='green', label="Large amplitude
peak")
    a0.set_ylabel('Voltage (V)', fontsize=20)
    a0.set_xlabel('Time (s)', fontsize=20)
    a0.xaxis.tick_top()
    a0.xaxis.set_label_position('top')
    a0.legend(loc='upper right')
    a0.set_xlim([0, 17.5])
```

```
    NullpeakDensity, NullpeakRange, _ = a1.hist(Nullpeaks, bins=binSize,
facecolor='r', alpha=1, edgecolor='k', linewidth=1)
    a1.set_title('low amplitude Peak Distribution Density', fontsize=20)
    a1.set_xticks([])
    a1.set_xlim([0, 70000])

    peakDensity, peakRange, _ = a2.hist(peaks, bins=binSize, facecolor='g',
alpha=1, edgecolor='k', linewidth=1)
    a2.set_title('Signal Peak Distribution Density', fontsize=20)
    f.text(-0.025, 0.3, 'Density', va='center', rotation='vertical',
fontsize=20)
    a2.set_xticks([])
    a2.set_xlim([0, 70000])

    plt.rc('font', family='Helvetica')

    plt.subplots_adjust(left=0.1,
                        bottom=0.1,
                        right=0.9,
                        top=0.9,
                        wspace=0,
                        hspace=0.6)
    f.savefig('Figures/DataDistribtion.png', bbox_inches='tight',
pad_inches=0.25)
    plt.show()


    NullmaxDensity  = max(NullpeakDensity)
    maxDensity  = max(peakDensity)

    interestingNullPeakDis = []
    interestingPeakDis = []
    for i in range(len(peakDensity)):
        if NullpeakDensity[i]>= perNullDen* NullmaxDensity:
            interestingNullPeakDis.append(i)

    for i in range(len(peakDensity)):
        if peakDensity[i]>= perMaxDen*maxDensity:
            interestingPeakDis.append(i)

    for i in range(len(interestingNullPeakDis)):
        if interestingNullPeakDis[i] in interestingPeakDis:
            interestingNullPeakDis[i] = ''
    chekcer = True
    while chekcer:
        if '' in interestingNullPeakDis:
            interestingNullPeakDis.remove('')
        else:
            chekcer = False
```

```
    NullpeakRange = np.insert(NullpeakRange, 0, 0)


    for i in range(len(interestingNullPeakDis)):
        place = interestingNullPeakDis[i]+2
        seperation.append(int((NullpeakRange[place])))

    for i in range(5):
        for i in range(0,len(seperation)-1):
            if seperation[i]+3000 >= seperation[i+1]:
                NewSep = (seperation[i] + seperation[i+1])/2 - randint(50,
500)
                seperation[i] = ''
                seperation[i+1] = int(NewSep)

        chekcer = True
        while chekcer:
            if '' in seperation:
                seperation.remove('')
            else:
                chekcer = False

    from mutagen.wave import WAVE
    fileName = FileName.replace("TrainingData\\", '')
    AudioFileName = fileName.split('-')
    AudioFileName = "AudioOriginal/"+AudioFileName[0]+'.wav'

    audio = WAVE(AudioFileName)
    audio_info = audio.info
    length = int(audio_info.length)

    for i in range(0,len(seperation)-1):
        if seperation[i] + length*4000*0.7 >= seperation[i + 1]:
            seperation[i+1] = 0

    chekcer = True
    while chekcer:
        if 0 in seperation:
            seperation.remove(0)
        else:
            chekcer = False
    fig1 = plt.figure()
    plt.plot(t, fderenV, linewidth=0.1, label="Signal")
    plt.scatter(t[seperation], fderenV[seperation], color='red', label =
"Seperation Points")
    plt.rc('font', family='Helvetica')
    plt.xlabel('Time (s)', fontsize=20)
    plt.ylabel('Voltage (V)', fontsize=20)
    plt.xlim([0,17.5])
```

```
    fig1.savefig('Figures/SectionedPoints.png', bbox_inches='tight',
pad_inches=0.25)
    plt.show()

    displacemetData = 750
    whileChecker = True
    while whileChecker:
        if seperation[len(seperation)-1]+70000<=len(voltage):
            #print("old Seperation : ",seperation[len(seperation) - 1])
            t = time[seperation[len(seperation)-1]-displacemetData:
seperation[len(seperation)-1]+80000]
            fderenV = voltage[seperation[len(seperation)-1]-displacemetData:
seperation[len(seperation)-1]+80000]

            maxVal = max(fderenV)

            NewSection = []

            Nullpeaks, _ = sig.find_peaks(fderenV, height=(-uplowBoundNull,
uplowBoundNull))
            peaks, _ = sig.find_peaks(fderenV, height=perofMaxVal * maxVal)


            plt.plot(t, fderenV, linewidth=0.05)
            plt.scatter(t[Nullpeaks], fderenV[Nullpeaks], color='red')
            plt.scatter(t[peaks], fderenV[peaks], color='green')

            NullpeakDensity, NullpeakRange, _ = plt.hist(Nullpeaks,
bins=binSize, facecolor='r', alpha=1, edgecolor='k', linewidth=1)

            peakDensity, peakRange, _ = plt.hist(peaks, bins=binSize,
facecolor='g', alpha=1, edgecolor='k', linewidth=1)
            plt.close()

            NullmaxDensity = max(NullpeakDensity)
            maxDensity = max(peakDensity)

            interestingNullPeakDis = []
            interestingPeakDis = []
            for i in range(len(peakDensity)):
                if NullpeakDensity[i] >= perNullDen * NullmaxDensity:
                    interestingNullPeakDis.append(i)

            for i in range(len(peakDensity)):
                if peakDensity[i] >= perMaxDen * maxDensity:
                    interestingPeakDis.append(i)

            for i in range(len(interestingNullPeakDis)):
                if interestingNullPeakDis[i] in interestingPeakDis:
```

```
                interestingNullPeakDis[i] = ''
            chekcer = True
            while chekcer:
                if '' in interestingNullPeakDis:
                    interestingNullPeakDis.remove('')
                else:
                    chekcer = False

            NullpeakRange = np.insert(NullpeakRange, 0, 0)

            for i in range(len(interestingNullPeakDis)):
                place = interestingNullPeakDis[i] + 2
                NewSection.append(int((NullpeakRange[place])))

            for j in range(10):
                for i in range(len(NewSection) - 1):
                    if NewSection[i] + length * 4000*0.7 >= NewSection[i + 1]
and NewSection[i] + length * 4000*0.7 <= NewSection[i + 1]:
                        NewSection[i + 1] = 0

                chekcer = True
                while chekcer:
                    if 0 in NewSection:
                        NewSection.remove(0)
                    else:
                        chekcer = False

            for j in range(10):
                for i in range(0, len(NewSection) - 1):
                    if NewSection[i] + 2000 >= NewSection[i + 1]:
                        NewSep = (NewSection[i] + NewSection[i + 1]) / 2 -
randint(50, 100)
                        NewSection[i] = ''
                        NewSection[i + 1] = int(NewSep)

                chekcer = True
                while chekcer:
                    if '' in NewSection:
                        NewSection.remove('')
                    else:
                        chekcer = False

            for i in range(2):
                for i in range(len(NewSection) - 1):
                    if NewSection[i] + length * 4000*0.7 >= NewSection[i + 1]:
                        NewSection[i + 1] = 0

                chekcer = True
```

```
            while chekcer:
                if 0 in NewSection:
                    NewSection.remove(0)
                else:
                    chekcer = False

        plt.plot(t, fderenV, linewidth=0.05)
        plt.scatter(t[NewSection], fderenV[NewSection], color='red')
        plt.show()

        previousSep = seperation[len(seperation) - 1]
        for i in range(1,len(NewSection)):
            seperation.append(NewSection[i]+previousSep-displacemetData)
    else:
        whileChecker = False
return(seperation)
```

C.5. MachineLearning.py

```
import matplotlib.pyplot as plt
import seaborn as sns
import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dropout
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from Labeler_Data import get_data
from sklearn.metrics import classification_report,confusion_matrix
from keras.utils.vis_utils import plot_model
import pydot

import tensorflow as tf

import numpy as np

def CNN_Training(folderTraining, folderTesting, ep, LearningRate, dataType,
img_size):
    #img_size = 32
    trainData = get_data(folderTraining, img_size)

    x_train = []
    y_train = []

    for feature, label in trainData:
      x_train.append(feature)
      y_train.append(label)

    # Normalize the data
    x_train = np.array(x_train) / 255

    x_train.reshape(-1, img_size, img_size, 1)
    y_train = np.array(y_train)

    testData = get_data(folderTesting, img_size)
    x_val = []
    y_val = []

    for feature, label in testData:
        x_val.append(feature)
        y_val.append(label)

    x_val = np.array(x_val) / 255
    x_val.reshape(-1, img_size, img_size, 1)
    y_val = np.array(y_val)
```

```
    CAT = ['Or_To_Take_Arms', 'That_Is_The_Question', 'To_be_or_not_to_be',
'To_die', 'To_Sleep', 'Whether']
    num_labels = len(CAT)
    datagen = ImageDataGenerator(
            featurewise_center=False,  # set input mean to 0 over the dataset
            samplewise_center=False,  # set each sample mean to 0
            featurewise_std_normalization=False,  # divide inputs by std of
the dataset
            samplewise_std_normalization=False,  # divide each input by its
std
            zca_whitening=False,  # apply ZCA whitening
            rotation_range = 0,  # randomly rotate images in the range
(degrees, 0 to 180)
            zoom_range = 0.3, # Randomly zoom image
            width_shift_range=0.2,  # randomly shift images horizontally
(fraction of total width)
            height_shift_range=False,  # randomly shift images vertically
(fraction of total height)
            horizontal_flip = False,  # randomly flip images
            vertical_flip=False)  # randomly flip images


    datagen.fit(x_train)


    model = Sequential()
    model.add(Conv2D(32, 3,padding="same", activation="relu",
input_shape=x_train.shape[1:]))
    model.add(MaxPool2D())

    # model.add(Conv2D(64, 3, padding="same", activation="relu"))
    # model.add(MaxPool2D())


    model.add(Conv2D(64, 3, padding="same", activation="relu"))
    model.add(MaxPool2D())
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(128,activation="relu"))
    #model.add()
    model.add(Dropout(0.5))
    model.add(Dense(num_labels))

    model.summary()

    opt = Adam(lr=LearningRate)
```

```
    model.compile(optimizer = opt , loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True) , metrics =
['accuracy'])

    history = model.fit(x_train,y_train,epochs = ep , validation_data =
(x_val, y_val))

    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs_range = range(ep)

    fig = plt.figure(figsize=(15, 15))
    plt.subplot(2, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title(dataType + ' Training and Validation Accuracy')

    plt.subplot(2, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title(dataType + ' Training and Validation Loss')
    plt.show()
    Name = dataType+"AccurracyandErrorPlot.png"
    fig.savefig(Name, transparent=True, bbox_inches='tight')

    model.summary()

    #plot_model(model, to_file='model_plot.png', show_shapes=True,
show_layer_names=True)

    return model, acc, val_acc, loss, val_loss

def TestingNetwrok(model, folderTesting, img_size):
    # img_size = 32
    testData = get_data(folderTesting, img_size)
    x_val = []
    y_val = []

    for feature, label in testData:
      x_val.append(feature)
      y_val.append(label)

    x_val = np.array(x_val) / 255
```

```
    x_val.reshape(-1, img_size, img_size, 1)
    y_val = np.array(y_val)

    CAT = ['A', 'B', 'C', 'D', 'E', 'F']

    predictions = model.predict(x_val)
    predictions = np.argmax(predictions,axis=1)
    print(classification_report(y_val, predictions, target_names = CAT))

    confusion_mtx = tf.math.confusion_matrix(y_val, predictions)
    fig1 = plt.figure()
    sns.heatmap(confusion_mtx, xticklabels=CAT, yticklabels=CAT,
                annot=True, fmt='g')
    plt.rc('font', family='Helvetica')
    plt.xlabel('Prediction',fontsize=20)
    plt.ylabel('Label',fontsize=20)
    plt.show()
    return y_val, predictions

def Save_CNN(model, Name):
    json_model = model.to_json()

    with open(Name, 'w') as json_file:
        json_file.write(json_model)
```

C.6. Laber_Data.py

```
import cv2
import os
import numpy as np
labels = ['Or_To_Take_Arms', 'That_Is_The_Question', 'To_be_or_not_to_be',
'To_die', 'To_Sleep', 'Whether']
#img_size = 32
def get_data(data_dir, img_size):
    data = []
    for label in labels:
        path = os.path.join(data_dir, label)
        class_num = labels.index(label)
        for img in os.listdir(path):
            try:
                #convert BGR to RGB format
                img_arr = cv2.imread(os.path.join(path, img))[...,::-1]
                # Reshaping images to preferred size
                resized_arr = cv2.resize(img_arr, (img_size, img_size))
                data.append([resized_arr, class_num])
            except Exception as e:
                print(e)
    return np.array(data)
```

Appendix D

Python code for real-time testing of the asymmetrical multichannel sensor with the *speech Recognition*.

GitHub link: https://github.com/abmoineddini/MPhil_speech_recognition

main.py

```
import serial
import numpy as np
import time

from PreprocessorSpeechrecognition import *
import cv2
import tensorflow as tf
from tkinter import *

ws = Tk()
ws.title('Voice Recognition')
ws.geometry('800x600')
ws.config(bg='#000000')

mylabel = Label(ws,
                text="...",
                bg='#000000',
                fg='#ffffff',
                font='Times 32',
                width=50,
                height=10)

mylabel.pack()
ws.update()

#COMPort = input("Please enter the COM port: ")
COMPort = 'COM6'
arduino = serial.Serial(COMPort, 2000000, timeout=1)
model = tf.keras.models.load_model("VoiceRecCNN")

Voltage = []
Time = []
dataCollection = False
dataAvailability = False
i = 0
j = 0
arr1 = []
arr2 = []
arr3 = []
arr4 = []
counter1 = 0
counter2 = 0
counter3 = 0
counter4 = 0
FirstCounter = True
```

```
while True:
    line = arduino.readline()
    if line != (''):
        #print(line)
        try:
            string = line.decode()
        except:
            print("ignored")
        else:
            numS = string.replace("\r\n", '')

            if numS.isdigit():
                print(int(numS))
                if int(numS) < 210:
                    dataCollection=False
                    if counter1<250:
                        counter1 +=1
                        arr1.append(int(numS))
                    elif counter1>=250 and counter1<500:
                        counter1 +=1
                        counter2 +=1
                        arr1.append(int(numS))
                        arr2.append(int(numS))
                    elif counter1>=500 and counter1<750:
                        counter1 +=1
                        counter2 +=1
                        counter3 +=1
                        arr1.append(int(numS))
                        arr2.append(int(numS))
                        arr3.append(int(numS))
                    else:
                        counter1 +=1
                        counter2 +=1
                        counter3 +=1
                        counter4 +=1
                        arr1.append(int(numS))
                        arr4.append(int(numS))
                        arr2.append(int(numS))
                        arr3.append(int(numS))


                    if counter4 == 250 and counter1 > 250:
                        counter1 = 0
                        arr1 = []

                    elif counter1==250 and counter2 > 250:
                        counter2 = 0
                        arr2 = []
```

```
                elif counter2==250 and counter3 > 250:
                    counter3 = 0
                    arr3 = []

                elif counter3==250 and counter4 > 250:
                    counter4 = 0
                    arr4 = []

            else:
                dataCollection = True
                Voltage.append(int(numS))
                Time.append(i / 4000)
                i = i + 1
                j = 8000
                print("StartingDataCollection")


while dataCollection:
    line = arduino.readline()
    print(line)
    if line != (''):
        try:
            string = line.decode()
        except:
            print("ignored")
        else:
            numS = string.replace("\r\n", '')
            if numS.isdigit():
                if int(numS)<210:
                    j = j-1
                else:
                    j = 8000

                Voltage.append(int(numS))
                Time.append(i / 4000)
                i = i+1

                if j<1:
                    dataCollection=False
                    dataAvailability = True

if dataAvailability:
    SIZESLIST = [len(arr1), len(arr2), len(arr3), len(arr4)]
    maxval = max(SIZESLIST)
    maxlen = SIZESLIST.index(maxval)
    V = []
    t = []
    if FirstCounter:
        predictVal = 6
```

```
            FirstCounter = False
            dataAvailability = False

        else:
            if maxlen == 0:
                V = np.vstack([arr1, Voltage])
                tAdj = np.linspace(0, (len(arr1)-1)*0.00025, len(arr1))
                tAdjm = tAdj[len(tAdj)-1]+0.00025
                for i in range(len(Time)):
                    Time[i] = Time[i]+tAdjm
                t = np.vstack([tAdj, Time])

            elif maxlen == 1:
                V = np.vstack([arr2, Voltage])
                tAdj = np.linspace(0, (len(arr2)-1)*0.00025, len(arr2))
                tAdjm = tAdj[len(tAdj)-1]+0.00025
                for i in range(len(Time)):
                    Time[i] = Time[i]+tAdjm
                t = np.vstack([tAdj, Time])
            elif maxlen == 2:
                V = np.vstack([arr3, Voltage])
                tAdj = np.linspace(0, (len(arr3) - 1) * 0.00025, len(arr3))
                tAdjm = tAdj[len(tAdj) - 1] + 0.00025
                for i in range(len(Time)):
                    Time[i] = Time[i] + tAdjm
                t = np.vstack([tAdj, Time])
            else:
                V = np.vstack([arr4, Voltage])
                tAdj = np.linspace(0, (len(arr4)-1)*0.00025, len(arr4))
                tAdjm = tAdj[len(tAdj)-1]+0.00025
                for i in range(len(Time)):
                    Time[i] = Time[i]+tAdjm
                t = np.vstack([tAdj, Time])

            plt.plot(t, V)
            plt.show()

            TestingPreprocessing(V, t)
            img_size = 300
            V = []
            t = []
            Voltage = []
            Time = []
            i = 0
            j = 0
            dataAvailability = False
            def Preprocess(path):
                img_arr = cv2.imread(path)[..., ::-1]
```

```
                resized_arr = cv2.resize(img_arr, (img_size, img_size))
                norm_arr = np.array(resized_arr) / 255
                return norm_arr.reshape(-1, img_size, img_size, 3)


            predict = model.predict([Preprocess("TestingFigure.png")])
            print(predict)

            predictVal = np.argmax(predict)

        if  predictVal == 0:
            text = 'or To take Arms\nagainst a sea of\ntroubles and\n by
opposing end them'
        elif predictVal == 1:
            text = 'that is the question'
        elif predictVal == 2:
            text = 'to be or not to be'
        elif predictVal == 3:
            text = 'to die'
        elif predictVal == 4:
            text = 'to sleep no more'
        elif predictVal == 5:
            text = 'whether this nobler\n in the mind to suffer\n the slings
and arrows of\n outrageous fortune'
        else:
            text = '...'
        mylabel.config(text=text)
        mylabel.pack()
        ws.update()
        time.sleep(5)
        mylabel.config(text='...')
        mylabel.pack()
        ws.update()
```

* The Preprocessing Speech recognition is the same as the one in in Appendix C Section C.3.

Appendix E

Arduino code to control the turntable.

GitHub link: https://github.com/abmoineddini/MPhil_sound_localisation/tree/main/Hardware_controllers

```
/* Arduino Mega rotary stage position controller
 * by: Amirbahador Moineddini
 * date: November 10th, 2021
 * V3 rotary stage position controller


/* Pin Setup
 * Motor Controller
 * enable pin:          11
 * step pin:            10
 * direction pin:       9
 * sensor singnal pin:  3
*/

const int en = 11;
const int stp = 10;
const int dir = 9;
const int pos = 3;
const int enLED = 13;
int Homming = 0;
int angleTest = 0;
int posVal;
int ptn;
int steps;
int angleToRotate;
volatile int delayTime;


void setup() {
  // Starting pins and start serial communication
  Serial.begin(115200);
  pinMode(en, OUTPUT);
  pinMode(stp, OUTPUT);
  pinMode(dir, OUTPUT);
  pinMode(pos, INPUT);
  pinMode(enLED, OUTPUT);
  digitalWrite(en, LOW);
  digitalWrite(enLED, HIGH);
  Serial.println("Setup Complete!");
}

void loop() {
  // Homming the stage to zero degrees
  if (Homming == 0) {
    Serial.println("Homming sequence starting ...");
    posVal = digitalRead(pos);
    Serial.println(posVal);
```

```
    if (posVal == 1) {
     Serial.println("Homming 2 ...");
     digitalWrite(dir, HIGH);
     for (int i = 0; i <= 1000; i++) {
       digitalWrite(stp, HIGH);
       delay(1);
       digitalWrite(stp, LOW);
       delay(1);
     }
   }
   if (posVal == 0 && Homming == 0) {
     Serial.println("Homming 1 ...");
     digitalWrite(dir, LOW);
     while (posVal == 0) {
       digitalWrite(stp, HIGH);
       delayMicroseconds(1000);
       digitalWrite(stp, LOW);
       delayMicroseconds(1000);
       posVal = digitalRead(pos);
     }
     digitalWrite(dir, LOW);
     for (int i = 0; i <= 180; i++) {
       digitalWrite(stp, HIGH);
       delayMicroseconds(1200);
       digitalWrite(stp, LOW);
       delayMicroseconds(1200);
     }
     Homming = 1;
     Serial.println("Homming Done");
     Serial.println("Loop 2");
     ptn = 0;
     delay(5000);
   }
 } else {
   Serial.println("ready");
   String Conn = Serial.readString();
   if (Conn == "rdy") {
     Serial.println("Starting");
     while (true) {
       String val = Serial.readString();
       Serial.println(val);
       int valInt = int(val.toInt());

       if (valInt > 0 && valInt <= 360) {
         if (valInt != ptn) {
           angleToRotate = (valInt - ptn);
           ptn = valInt;
         }
```

```
    else {
        angleToRotate = 0;
      }
      if (angleToRotate > 180) {
        angleToRotate = angleToRotate - 360;
      }
      if (angleToRotate < -180) {
        angleToRotate = 360 + angleToRotate;
      }
      steps = angleToRotate * 50;
      if (angleToRotate > 0) {
        digitalWrite(dir, LOW);
        for (int i = 0; i <= steps; i++) {
          digitalWrite(stp, HIGH);
          delayMicroseconds(900);
          digitalWrite(stp, LOW);
          delayMicroseconds(900);
        }
      }
      if (angleToRotate < 0) {
        digitalWrite(dir, HIGH);
        for (int i = 0; i <= abs(steps); i++) {
          digitalWrite(stp, HIGH);
          delayMicroseconds(900);
          digitalWrite(stp, LOW);
          delayMicroseconds(900);
        }
      }
      delayTime = abs(angleToRotate) * 10;
      delay(delayTime);

      Serial.println("done");
      if (ptn == 360) {
        ptn = 0;
      }
      Serial.println(ptn);
    }
    if (valInt < 0 && valInt >= -360) {
      valInt = 360 + valInt;
      if (valInt != ptn) {
        angleToRotate = 360 - (valInt - ptn);
        ptn = valInt;
      } else {
        angleToRotate = 0;
      }
      if (angleToRotate > 360) {
        angleToRotate = -720 + angleToRotate;
      }
```

```
          if (angleToRotate > 180) {
            angleToRotate = angleToRotate - 360;
          }
          if (angleToRotate < -180) {
            angleToRotate = 360 + angleToRotate;
          }
          steps = angleToRotate * 50;
          if (angleToRotate > 0) {
            digitalWrite(dir, HIGH);
            for (int i = 0; i <= abs(steps); i++) {
              digitalWrite(stp, HIGH);
              delayMicroseconds(900);
              digitalWrite(stp, LOW);
              delayMicroseconds(900);
            }
          }
          if (angleToRotate < 0) {
            digitalWrite(dir, LOW);
            for (int i = 0; i <= abs(steps); i++) {
              digitalWrite(stp, HIGH);
              delayMicroseconds(900);
              digitalWrite(stp, LOW);
              delayMicroseconds(900);
            }
          }
          delayTime = abs(angleToRotate) * 10;
          delay(delayTime);

          Serial.println("done");
          //            if (ptn==360){
          //              ptn=0;
          //            }
          Serial.println(ptn);
        }
      }
    }
  }
}
```

Appendix F

Python codes for data collection, preprocessing the data, plotting figures and training, testing and validation of the *Spatial Recognition*.

GitHub link: https://github.com/abmoineddini/MPhil_sound_localisation

F.1. Main code:

```
from DataCollector import *
import matplotlib.pyplot as plt
import os
import pandas as pd
from os import listdir
import winsound
from DataSender import *
from datetime import date
from Preprocessor import *
from csv import writer

dataCollection = True
CheckPort = input("Would you like to Start collecting Data? ")
if CheckPort == "y" or CheckPort == "Y" or CheckPort == "Yes" or CheckPort ==
"yes":
    # Initialising the Stage and Data Collector
    COMPortMotor = input("Please Enter COM Port for the Stage : ")
    COMPortMotor  = "COM"+COMPortMotor
    print(COMPortMotor )
    currAng, MotorController = Inititialise(COMPortMotor )
    time.sleep(1)
    print("Initialisation of Stage Completed")
    COMPortCollector = input("Please Enter COM Port for Data Collector : ")
    COMPort  = "COM"+COMPortCollector
    print(COMPort)
    Increment = input("Please enter the desired Increment in degrees: ")
    RotationAngle = input("Please enter the angle you would like to cover: ")
    DirectionOfRot = input ("Please enter the Direction of Roation (0 =
                            Antliclockwise, 1 = Clockwise: ")
    Method = input("Continuous or Individual Collections? (i for individual &
                        c for Continious)")
    AutomaticTesting = input("Would you like continous automatic test ? (yN)")
    today = date.today()
    print("Today's date:", today)
    TrainingDirectoryName = "TrainingData/" + str(today)
    if os.path.isdir(TrainingDirectoryName):
        print("Adding to Figure Directory")
    else:
        os.mkdir(TrainingDirectoryName)
    while dataCollection:
        Increment = int(Increment)
        div = 2*int(int(RotationAngle)/Increment)+1
        for inc in range(0, div):
```

```
if DirectionOfRot == 1:
    if inc <= (div-1)/2:
        angle = Increment*inc
    else:
        angle = int(RotationAngle)-Increment * inc
else:
    if inc <= (div-1)/2:
        angle = Increment*inc
    else:
        angle = (Increment * inc-int(RotationAngle))
if Method == 'c':
    print("Continuous Data Collection")
    print("Sending Angles")
    currAng = AngleSet(angle, MotorController, currAng)
    # AudioFiles = [f for f in listdir("AudioFiles/")]
    Name = str(currAng)
    print(Name)
    time.sleep(1)
    Period = 10
    print("Startting to Collect Data for: ", Period, '(s)')
    NameTest = True
    TrainingDataDirectory = [f for f in
                                listdir(TrainingDirectoryName)]
    testNum = 1
    while NameTest:
        CSVNameCheck = "c_" +Name + '-Test'+ str(testNum) + '.csv'
        if CSVNameCheck in TrainingDataDirectory:
            print("File Already exist, Trying another name.")
            testNum = testNum+1
        else:
            CSVName = Name + '-Test'+ str(testNum)
            NameTest = False
    print(CSVName)
    [Channel1,Channel2, Channel3,Channel4, Time] =
collectData(COMPort, Period, CSVName, TrainingDirectoryName)

   # Plotting figures for checking the figures
    plt.plot(Time, Channel1)
    plt.plot(Time, Channel2)
    plt.plot(Time, Channel3)
    plt.plot(Time, Channel4)
    plt.show()

    DataCheck = 'y' # input("Are you happy with the Data? ")
    while DataCheck == "n" or DataCheck == "N" or DataCheck ==
             "no" or DataCheck == "No":
        Nametoremove = "TrainingData/" + CSVName + ".csv"
        os.remove(Nametoremove)
```

```
                [Channel1,Channel2, Channel3,Channel4, Time] =
            collectData(COMPort, Period, CSVName, TrainingDirectoryName)

                plt.plot(Time, Channel1)
                plt.plot(Time, Channel2)
                plt.plot(Time, Channel3)
                plt.plot(Time, Channel4)
                plt.show()
                DataCheck = 'y' #input("Are you happy with the Data? ")

            elif Method=='i':
                print("Individual Data Collection")
                print("Sending Angles")
                currAng = AngleSet(angle, MotorController, currAng)
                # AudioFiles = [f for f in listdir("AudioFiles/")]
                Name = str(currAng)
                print(Name)
                time.sleep(1)
                Period = 3
                print("Startting to Collect Data for: ", Period, '(s)')
                NameTest = True
                TrainingDataDirectory = [f for f in
                         listdir(TrainingDirectoryName)]
                testNum = 1
                while NameTest:
                    CSVNameCheck = "i_" + Name + '-Test' + str(testNum) +
'.csv'
                    if CSVNameCheck in TrainingDataDirectory:
                        print("File Already exist, Trying another name.")
                        testNum = testNum + 1
                    else:
                        CSVName = Name + '-Test' + str(testNum)
                        NameTest = False

                print(CSVName)
                [Channel1, Channel2, Channel3, Channel4, Time] =
        collectDataIndividual(COMPort, Period, CSVName,TrainingDirectoryName)

                # Plotting figures for checking the figures
                plt.plot(Time, Channel1)
                plt.plot(Time, Channel2)
                plt.plot(Time, Channel3)
                plt.plot(Time, Channel4)
                winsound.PlaySound(None, winsound.SND_PURGE)
                DataCheck = 'y'  # input("Are you happy with the Data? ")
                while DataCheck == "n" or DataCheck == "N" or DataCheck ==
            "no" or DataCheck == "No":
                    Nametoremove = "TrainingData/" + CSVName + ".csv"
```

```
                    os.remove(Nametoremove)

                    [Channel1, Channel2, Channel3, Channel4, Time] =
collectDataIndividual(COMPort, Period, CSVName, TrainingDirectoryName)

                    # Plotting figures for checking the figures
                    plt.plot(Time, Channel1)
                    plt.plot(Time, Channel2)
                    plt.plot(Time, Channel3)
                    plt.plot(Time, Channel4)
                    plt.show()
                    DataCheck = 'y'  # input("Are you happy with the Data? ")
            else:
                print("Invalid Answer!")
                Method = input("Continuous or Individual Collections? (i for
individual & c for Continious)")

            else:
                print("Invalid Answer!")
                Method = input("Continuous or Individual Collections? (i for
individual & c for Continious)")

            if AutomaticTesting == "y":
                ToContinue = 'y'
            else:
                input("would you like to Continue with collecting data? ")

        if ToContinue == "n" or ToContinue == "N" or ToContinue == "No" or
ToContinue == "no":
            dataCollection = False


StartPreprocessing = input("Should I Start the Preprocessing? ")
if StartPreprocessing == "y" or StartPreprocessing == "Y" or
StartPreprocessing == "Yes" or StartPreprocessing == "yes":

    import glob
    if os.path.isdir("Figure/Training"):
        print("Adding to Figure Directory")
    else:
        os.mkdir("Figure/Training")
        os.mkdir("Figure/Testing")
        os.mkdir("Figure/Validation")

    for i in os.listdir("TrainingData"):
        DirectoryMasterTraining = "TrainingData/"+i+"/"
        csv_files = glob.glob(os.path.join(DirectoryMasterTraining, "*.csv"))
        for x in csv_files[0:]:
            dataFileName = x
```

```
            print(x)
            NAMEunprocessed = x.replace(".csv", "")
            NAMEunprocessed = NAMEunprocessed.split("\\")
            print(NAMEunprocessed)
            print(NAMEunprocessed[1][0])

            NAMEunprocessed = NAMEunprocessed[1].split("_")
            print(NAMEunprocessed)
            if NAMEunprocessed[0] == "c":
                method = 0
                print("True")
            else:
                method = 1
            NAMEunprocessed = NAMEunprocessed[1]
            NAMEunprocessed = NAMEunprocessed.split("-")
            x = NAMEunprocessed[0]
            print(x)
            check = 0

            if os.path.isfile("Tracking/ProcessedData.csv"):
                print("Processed data collector Already exists")

                PDCL = pd.read_csv("Tracking/ProcessedData.csv")
                print(PDCL)
                ProcessedDataChecklist = PDCL.to_numpy()
                ProcessedDataChecklist = ProcessedDataChecklist[:]

                if dataFileName.split("/")[1] in ProcessedDataChecklist:
                    check = 1
                    continue

            if check == 0:
                [FolderTraining, FolderTesting, FolderValidation] =
             Classifier(x)
                print("it reaches here")
                df = pd.read_csv(dataFileName)
                data = df.to_numpy()
                print(dataFileName)

                if method==0:
                    figure_maker(data, FolderTraining, FolderTesting,
FolderValidation, dataFileName)
                    print("Making figure for continuous Data")
                else:
                    figure_makerMeth2AutoSize(data, FolderTraining,
FolderTesting, FolderValidation, dataFileName)

                if os.path.isfile("Tracking/ProcessedData.csv"):
```

```
                    with open("Tracking/ProcessedData.csv", 'a+' ,newline='')
as f_object:
                        writer_object = writer(f_object)
                        writer_object.writerow([dataFileName.split("/")[1]])
                        f_object.close()
                else:
                    dict = {"File Name": [dataFileName.split("/")[1]]}
                    df = pd.DataFrame(dict)
                    df.to_csv("Tracking/ProcessedData.csv")

print("Finished Creating Relevant Files")

labels = os.listdir("Figure/Training")
labelsInt = []
for i in labels:
    labelsInt.append(int(i))
labelsInt.sort()
labels = []
for i in labelsInt:
    labels.append(str(i))

StartTraining = input("Should I Start the Training? ")
if StartTraining == "y" or StartTraining == "Y" or StartTraining == "yes" or
StartTraining == "Yes":

    trainingSTFT = "Figure/Training"
    validationSTFT = "Figure/Validation"
    testingSTFT = "Figure/Testing"

    from MachineLearning import *

    img_size = 150
    print(labels)
    [STFTModel, acc, val_acc, loss, val_loss]= CNN_Training(trainingSTFT,
validationSTFT, 150, LearningRate=0.00005, dataType="STFT", img_size=img_size,
label = labels)
    STFTModel.save("DirectionRecCNN")
    from csv import writer
    for i in range(len(acc)):
        FileAdd = [acc[i], val_acc[i], loss[i], val_loss[i]]
        with open("Tracking/AccuracyHistory.csv", 'a+', newline='') as
        f_object:
            writer_object = writer(f_object)
            writer_object.writerow(FileAdd)
            f_object.close()
            FileAdd = []

    STFTModel = tf.keras.models.load_model("DirectionRecCNN")
```

```
    # Testinig Peformance
    print("STFT CNN Test result")
    [y_val, predictions] = TestingNetwrok(STFTModel, testingSTFT, img_size,
labels)

    for i in range(len(y_val)):
        FileAdd = [y_val[i], predictions[i]]
        with open("Tracking/TestingValidationCNN.csv", 'a+', newline='') as
f_object:
            writer_object = writer(f_object)
            writer_object.writerow(FileAdd)
            f_object.close()
            FileAdd = []
    STFTmodelName = 'STFTModel.yaml'
    Save_CNN(STFTModel, Name=STFTmodelName)

df = pd.read_csv("Tracking/TestingValidationCNN.csv")
CAT = labels
DegNum = int(labels[1])
DegNum = str(DegNum)
data = df.to_numpy()
y_val = data[:,0]
predictions = data[:,1]
fig = plt.figure()
confusion_mtx = tf.math.confusion_matrix(y_val, predictions)
print(confusion_mtx)
con_matrix = np.zeros((len(labels),len(labels)))
for i in range(len(confusion_mtx[1])):
    row = confusion_mtx[i].numpy()
    rowSum = row.sum()
    print(rowSum)
    for j in range(len(row)):
        print(len(row))
        print(row[j])
        con_matrix[i, j] = row[j] / rowSum

sns.heatmap(con_matrix, xticklabels=CAT, yticklabels=CAT,
            annot=True, fmt='.2f', cmap = "OrRd")
plt.rc('font', family='Helvetica')
plt.title('Confusion Matix', fontsize=22)
plt.xlabel('Prediction', fontsize=20)
plt.ylabel('Label', fontsize=20)
ConfunstionMatrixName = "Figures/" + DegNum + "ConfusionMatrixValidation.png"
fig.savefig(ConfunstionMatrixName, transparent=True,
bbox_inches='tight',pad_inches=0.25)
plt.show()
```

F.2.

```
import serial
import winsound
def collectData(COMPort, Period, Name, DirectoryName):
    arduino = serial.Serial(COMPort , 2000000, timeout=1)
    Channel1 = []
    Channel2 = []
    Channel3 = []
    Channel4 = []
    Time = []
    period = int(Period)
    sR = int(4000/3.5)
    period = period*sR
    for i in range(period):
        line = arduino.readline()

        if line != (''):
            print(line)
            try:
                string = line.decode()
            except:
                print("ignored")
            else:
                numS = string.replace("\r\n", '')
                vals = numS.split(" ")
                if len(vals)>3:
                    if vals[0].isdigit():
                        if vals[1].isdigit():
                            if vals[2].isdigit():
                                if vals[3].isdigit():
                                    Channel1.append(int(vals[0]))
                                    Channel2.append(int(vals[1]))
                                    Channel3.append(int(vals[2]))
                                    Channel4.append(int(vals[3]))
                                    Time.append(i/sR)

    arduino.close()
    import pandas as pd
    dict = {'Time (s)' : Time, 'Channel 1 (V)': Channel1,'Channel 2 (V)':
Channel2, 'Channel 3 (V)': Channel3, 'Channel4 (V)': Channel4}

    df = pd.DataFrame(dict)
    dataBaseName = DirectoryName+"/"+"c_" + Name + ".csv"
    df.to_csv(dataBaseName)

    return [Channel1,Channel2, Channel3,Channel4, Time]
```

```
#######################Individual Collection Method#############################
def collectDataIndividual(COMPort, Period, Name, DirectoryName):
    arduino = serial.Serial(COMPort , 2000000, timeout=1)
    Channel1 = []
    Channel2 = []
    Channel3 = []
    Channel4 = []
    Time = []
    period = int(Period)
    sR = int(4000/3.5)
    period = period*sR
    for i in range(period):
        line = arduino.readline()

        if line != (''):
            print(line)
            try:
                string = line.decode()
            except:
                print("ignored")
            else:
                numS = string.replace("\r\n", '')
                vals = numS.split(" ")
                if len(vals)>3:
                    if vals[0].isdigit():
                        if vals[1].isdigit():
                            if vals[2].isdigit():
                                if vals[3].isdigit():
                                    Channel1.append(int(vals[0]))
                                    Channel2.append(int(vals[1]))
                                    Channel3.append(int(vals[2]))
                                    Channel4.append(int(vals[3]))
                                    Time.append(i/sR)

    arduino.close()
    import pandas as pd
    dict = {'Time (s)' : Time, 'Channel 1 (V)': Channel1,'Channel 2 (V)':
Channel2, 'Channel 3 (V)': Channel3, 'Channel4 (V)': Channel4}

    df = pd.DataFrame(dict)
    dataBaseName = DirectoryName + "/"+"i_" + Name + ".csv"
    df.to_csv(dataBaseName)

    return [Channel1,Channel2, Channel3,Channel4, Time]
```

```
####################### Testing Collection ############################

def collectDataTest(COMPort, Period): #, AudioFileName):
    arduino = serial.Serial(COMPort, 2000000, timeout=1)
    Channel1 = []
    Channel2 = []
    Channel3 = []
    Channel4 = []
    Time = []
    period = int(Period)
    sR = int(4000 / 3.5)
    period = period * sR
    for i in range(period):
        line = arduino.readline()

        if line != (''):
            print(line)
            try:
                string = line.decode()
            except:
                print("ignored")
            else:
                numS = string.replace("\r\n", '')
                vals = numS.split(" ")
                if len(vals) > 3:
                    if vals[0].isdigit():
                        if vals[1].isdigit():
                            if vals[2].isdigit():
                                if vals[3].isdigit():
                                    Channel1.append(int(vals[0]))
                                    Channel2.append(int(vals[1]))
                                    Channel3.append(int(vals[2]))
                                    Channel4.append(int(vals[3]))
                                    Time.append(i / sR)

    arduino.close()



    arduino.close()
    import pandas as pd
    dict = {'Time (s)': Time, 'Channel 1 (V)': Channel1,'Channel 2 (V)':
Channel2, 'Channel 3 (V)': Channel3, 'Channel4 (V)': Channel4}

    df = pd.DataFrame(dict)
    dataBaseName = "Temp/Test.csv"
    df.to_csv(dataBaseName)
    print(dataBaseName)
    return [Channel1,Channel2, Channel3,Channel4, Time]
```

F.3. Preprocessing.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as sig
from SectionCutting import *
from os import listdir
from PIL import Image
import os
from os.path import isdir
from scipy.fft import fft, fftfreq
import random

def ProcessIm(Im1, Im2, fileName):
    image1 = Image.open(Im1)
    image1 = image1.rotate(90)

    image2 = Image.open(Im2)
    image1_size = image1.size
    new_image = Image.new('RGB', (2 * image1_size[0], image1_size[1]), (250,
250, 250))
    new_image.paste(image1, (0, 0))
    new_image.paste(image2, (image1_size[0], 0))
    new_image.save(fileName)

def ProcessConcat(Im1, Im2, Im3, Im4, fileName):
    image1 = Image.open(Im1)
    image2 = Image.open(Im2)
    image3 = Image.open(Im3)
    image4 = Image.open(Im4)

    image1_size = image1.size

    new_image = Image.new('RGB', (2 * image1_size[0], 2*image1_size[1]), (250,
250, 250))
    new_image.paste(image1, (0, 0))
    new_image.paste(image2, (image1_size[0], 0))
    new_image.paste(image3, (0, image1_size[1]))
    new_image.paste(image4, (image1_size[0], image1_size[1]))
    new_image.save(fileName)

def ProcessConcatAuto(Im, fileName):
    image1 = Image.open("Temp/"+Im[0])
    ChannelCount = 1
    if len(Im) > 1:
        image2 = Image.open("Temp/"+Im[1])
        ChannelCount = 2
        if len(Im) > 2:
            image3 = Image.open("Temp/"+Im[2])
```

```
            ChannelCount = 3
            if len(Im) > 3:
                image4 = Image.open("Temp/"+Im[3])
                ChannelCount = 4
                if len(Im) > 4:
                    image5 = Image.open("Temp/"+Im[4])
                    ChannelCount = 5
                    if len(Im) > 5:
                        image6 = Image.open("Temp/"+Im[5])
                        ChannelCount = 6
                        if len(Im) > 6:
                            image7 = Image.open("Temp/"+Im[6])
                            ChannelCount = 7
                            if len(Im) > 7:
                                image8 = Image.open("Temp/"+Im[7])
                                ChannelCount = 8
                                if len(Im) > 8:
                                    image9 = Image.open("Temp/" + Im[8])
                                    ChannelCount = 9

    image1_size = image1.size
    BlackImage = np.zeros([image1_size[1], image1_size[0], 3] ,dtype=np.uint8)
    BlackImage = Image.fromarray(BlackImage)

    if ChannelCount == 1:
        new_image = Image.new('RGB', (image1_size[0], image1_size[1]), (250,
250, 250))
        new_image.paste(image1, (0, 0))
        new_image.save(fileName)

    if ChannelCount == 2:
        new_image = Image.new('RGB', (2 * image1_size[0], 2*image1_size[1]),
(250, 250, 250))
        new_image.paste(image1, (0, 0))
        new_image.paste(BlackImage, (image1_size[0], 0))
        new_image.paste(image2, (image1_size[0], image1_size[1]))
        new_image.paste(BlackImage, (0, image1_size[1]))
        new_image.save(fileName)

    if ChannelCount == 3:
        new_image = Image.new('RGB', (2 * image1_size[0], 2*image1_size[1]),
(250, 250, 250))
        new_image.paste(image1, (0, 0))
        new_image.paste(image2, (image1_size[0], 0))
        new_image.paste(image3, (0, image1_size[1]))
        new_image.save(fileName)

    if ChannelCount == 4:
```

```
        new_image = Image.new('RGB', (2 * image1_size[0], 2*image1_size[1]),
(250, 250, 250))
        new_image.paste(image1, (0, 0))
        new_image.paste(image2, (image1_size[0], 0))
        new_image.paste(image3, (0, image1_size[1]))
        new_image.paste(image4, (image1_size[0], image1_size[1]))
        new_image.save(fileName)

    if ChannelCount == 5:
        new_image = Image.new('RGB', (3 * image1_size[0], 2*image1_size[1]),
(250, 250, 250))
        new_image.paste(image1, (0, 0))
        new_image.paste(image2, (image1_size[0], 0))
        new_image.paste(image3, (0, image1_size[1]))
        new_image.paste(image4, (image1_size[0], image1_size[1]))
        new_image.paste(image5, (2*image1_size[0], 0))
        new_image.save(fileName)

    if ChannelCount == 6:
        new_image = Image.new('RGB', (3 * image1_size[0], 2*image1_size[1]),
(250, 250, 250))
        new_image.paste(image1, (0, 0))
        new_image.paste(image2, (image1_size[0], 0))
        new_image.paste(image3, (0, image1_size[1]))
        new_image.paste(image4, (image1_size[0], image1_size[1]))
        new_image.paste(image5, (2*image1_size[0], 0))
        new_image.paste(image6, (2*image1_size[0], image1_size[1]))
        new_image.save(fileName)

    if ChannelCount == 7:
        new_image = Image.new('RGB', (4 * image1_size[0], 3*image1_size[1]),
(250, 250, 250))
        new_image.paste(image1, (0, 0))
        new_image.paste(image2, (image1_size[0], 0))
        new_image.paste(image3, (0, image1_size[1]))
        new_image.paste(image4, (image1_size[0], image1_size[1]))
        new_image.paste(image5, (2*image1_size[0], 0))
        new_image.paste(image6, (2*image1_size[0], image1_size[1]))
        new_image.paste(image7, (0, 2*image1_size[0]))
        new_image.save(fileName)

    if ChannelCount == 9:
        new_image = Image.new('RGB', (3 * image1_size[0], 3 * image1_size[1]),
(250, 250, 250))
        new_image.paste(image1, (0, 0))
        new_image.paste(image2, (image1_size[0], 0))
        new_image.paste(image3, (0, image1_size[1]))
        new_image.paste(image4, (image1_size[0], image1_size[1]))
```

```
        new_image.paste(image5, (2*image1_size[0], 0))
        new_image.paste(image6, (2*image1_size[0], image1_size[1]))
        new_image.paste(image7, (0, 2*image1_size[0]))
        new_image.paste(image8, (image1_size[0], 2*image1_size[1]))
        new_image.paste(image9, (2 * image1_size[0], 2 * image1_size[1]))
        new_image.save(fileName)

    if ChannelCount == 9:
        new_image = Image.new('RGB', (3 * image1_size[0], 3*image1_size[1]),
(250, 250, 250))
        new_image.paste(image1, (0, 0))
        new_image.paste(image2, (image1_size[0], 0))
        new_image.paste(image3, (0, image1_size[1]))
        new_image.paste(image4, (image1_size[0], image1_size[1]))
        new_image.paste(image5, (2*image1_size[0], 0))
        new_image.paste(image6, (2*image1_size[0], image1_size[1]))
        new_image.paste(image7, (0, 2*image1_size[1]))
        new_image.paste(image8, (image1_size[0], 2*image1_size[1]))
        new_image.paste(image9, (2 * image1_size[0], 2 * image1_size[1]))
        new_image.save(fileName)

def figure_maker(data, FolderSTFTTraining, FolderSTFTTesting,
FolderValTesting, FileName):
    time = data[150:len(data)-1, 1]
    Ch1 = data[150:len(data)-1, 2]
    Ch2 = data[150:len(data) - 1, 3]
    Ch3 = data[150:len(data) - 1, 4]
    Ch4 = data[150:len(data) - 1, 5]
    Fs = 4000/3.5

    detenCh1 = sig.detrend(Ch1)
    detenCh2 = sig.detrend(Ch2)
    detenCh3 = sig.detrend(Ch3)
    detenCh4 = sig.detrend(Ch4)


    filter = sig.butter(2, [70,550], 'bandpass', fs=Fs, output='sos')
    corrCh1 = sig.sosfilt(filter, detenCh1)
    corrCh1 = (corrCh1*5)/1023
    corrCh2 = sig.sosfilt(filter, detenCh2)
    corrCh2 = (corrCh2*5)/1023
    corrCh3 = sig.sosfilt(filter, detenCh3)
    corrCh3 = (corrCh3*5)/1023
    corrCh4 = sig.sosfilt(filter, detenCh4)
    corrCh4 = (corrCh4*5)/1023

    TrainingDirectory = [f for f in listdir(FolderSTFTTraining)]
    TestDirectory = [f for f in listdir(FolderSTFTTesting)]
```

```
    ValidationDirectory = [f for f in listdir(FolderValTesting)]
    countTrain = int(len(TrainingDirectory))
    countTest = int(len(TestDirectory))
    countVal = int(len(ValidationDirectory))

    maxV1 = max(corrCh1)
    normV1 = corrCh1 / maxV1
    maxV2 = max(corrCh2)
    normV2 = corrCh2 / maxV2
    maxV3 = max(corrCh3)
    normV3 = corrCh3 / maxV3
    maxV4 = max(corrCh4)
    normV4 = corrCh4 / maxV4
    seperationPoints = SepWithSTFT(normV1, normV2, normV3, normV4, time, Fs)
    #seperationPoints = [0, len(normV1)-1]

    N = len(seperationPoints)
    Arr = np.arange(N)
    np.random.shuffle(Arr)
    Training = Arr[:round(N*0.8)]
    Validation = Arr[round(N*0.8):round(N*0.9)]
    Testing = Arr[round(N*0.9):round(N*1)]

    for i in range(0,N-1):
        Ch1pp = normV1[int(seperationPoints[i]):int(seperationPoints[i+1])]
        Ch2pp = normV2[int(seperationPoints[i]):int(seperationPoints[i+1])]
        Ch3pp = normV3[int(seperationPoints[i]):int(seperationPoints[i+1])]
        Ch4pp = normV4[int(seperationPoints[i]):int(seperationPoints[i+1])]

        ## Channel 1
        f1, t1, Zxx1 = sig.stft(Ch1pp, Fs, nperseg=100)
        fig1 = plt.figure(frameon=False)
        ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
        plt.pcolormesh(t1, f1, np.abs(Zxx1), shading='gouraud', cmap='gray')
        plt.axis('off')
        plt.ylim([90, 500])
        ax1.set_axis_off()
        plt.tight_layout()
        plt.show()
        fig1.savefig('Temp/STFTTestingFigureCh1.png', bbox_inches='tight',
pad_inches=0)

        ## Channel 2
        f2, t2, Zxx2 = sig.stft(Ch2pp, Fs, nperseg=100)
        fig1 = plt.figure(frameon=False)
        ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
        plt.pcolormesh(t2, f2, np.abs(Zxx2), shading='gouraud', cmap='gray')
        plt.axis('off')
```

```
        plt.ylim([90, 500])
        ax1.set_axis_off()
        plt.tight_layout()
        plt.show()
        fig1.savefig('Temp/STFTTestingFigureCh2.png', bbox_inches='tight',
pad_inches=0)

        ##Channel 3
        f3, t3, Zxx3 = sig.stft(Ch3pp, Fs, nperseg=100)
        fig1 = plt.figure(frameon=False)
        ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
        plt.pcolormesh(t3, f3, np.abs(Zxx3), shading='gouraud', cmap='gray')
        plt.axis('off')
        plt.ylim([90, 500])
        ax1.set_axis_off()
        plt.tight_layout()
        plt.show()
        fig1.savefig('Temp/STFTTestingFigureCh3.png', bbox_inches='tight',
pad_inches=0)

        ## Channel 4
        f4, t4, Zxx4 = sig.stft(Ch4pp, Fs, nperseg=100)
        fig1 = plt.figure(frameon=False)
        ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
        plt.pcolormesh(t4, f4, np.abs(Zxx4), shading='gouraud', cmap='gray')
        plt.axis('off')
        plt.xlim([90, 500])
        ax1.set_axis_off()
        plt.tight_layout()
        plt.show()
        fig1.savefig('Temp/STFTTestingFigureCh4.png', bbox_inches='tight',
pad_inches=0)

        y1 = fft(Ch1pp)
        y2 = fft(Ch2pp)
        y3 = fft(Ch3pp)
        y4 = fft(Ch4pp)


        y1a = y1[80: 550]
        max_value1 = max(abs(y1a))
        Norm_y1 = abs(y1a) / 15 # max_value1
        Norm_y1a = abs(y1a)

        y2a = y2[80: 550]
        max_value2 = max(abs(y2a))
        Norm_y2 = abs(y2a) / 15 # max_value2
        Norm_y2a = abs(y2a)
```

```
y3a = y3[80: 550]
max_value3 = max(abs(y3a))
Norm_y3 = abs(y3a) / 15 # max_value3
Norm_y3a = abs(y3a)

y4a = y4[80: 550]
max_value4 = max(abs(y4a))
Norm_y4 = abs(y4a) / 15 # max_value4
Norm_y4a = abs(y4a)

N = len(y4a)
xf = fftfreq(N, 1/Fs)

fig1 = plt.figure()
ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
plt.scatter(xf, Norm_y1a, c=4**abs(Norm_y1), s=abs(Norm_y1**4)*500)
plt.ylim([0, 16])
plt.gray()
plt.axis('off')
ax1.set_axis_off()
plt.tight_layout()
fig1.savefig('Temp/Ch1fft.png', bbox_inches='tight', pad_inches=0)

fig2 = plt.figure()
ax2 = plt.Axes(fig2, [0., 0., 1., 1.])
plt.scatter(xf, Norm_y2a, c=4**abs(Norm_y2), s=abs(Norm_y2**4)*500)
plt.ylim([0, 16])
plt.gray()
plt.axis('off')
ax2.set_axis_off()
plt.tight_layout()
fig2.savefig('Temp/Ch2fft.png', bbox_inches='tight', pad_inches=0)

fig3 = plt.figure()
ax3 = plt.Axes(fig3, [0., 0., 1., 1.])
plt.scatter(xf, Norm_y3a, c=4**abs(Norm_y3), s=abs(Norm_y3**4)*500)
plt.ylim([0, 16])
plt.gray()
plt.axis('off')
ax3.set_axis_off()
plt.tight_layout()
fig3.savefig('Temp/Ch3fft.png', bbox_inches='tight', pad_inches=0)

fig4 = plt.figure()
ax4 = plt.Axes(fig4, [0., 0., 1., 1.])
plt.scatter(xf, Norm_y4a, c=4**abs(Norm_y4), s=abs(Norm_y4**4)*500)
plt.ylim([0, 16])
plt.gray()
```

```
        plt.axis('off')
        ax4.set_axis_off()
        plt.tight_layout()
        fig4.savefig('Temp/Ch4fft.png', bbox_inches='tight', pad_inches=0)

        if i in Training:
            nameTrain = str(countTrain)
            countTrain += 1
            FileName = FolderSTFTTraining + "/" + nameTrain + '.png'
            print(FileName)
            ProcessConcat('Temp/Ch1fft.png',
'Temp/Ch2fft.png','Temp/Ch3fft.png', 'Temp/Ch4fft.png', FileName)

        if i in Validation:
            nameVal = str(countVal)
            countVal += 1
            FileName = FolderValTesting + "/" + nameVal + '.png'
            print(FileName)
            ProcessConcat('Temp/Ch1fft.png', 'Temp/Ch2fft.png',
'Temp/Ch3fft.png', 'Temp/Ch4fft.png', FileName)

        if i in Testing:
            nameTest = str(countTest)
            countTest += 1
            FileName = FolderSTFTTesting + "/" + nameTest + '.png'
            print(FileName)
            ProcessConcat('Temp/Ch1fft.png', 'Temp/Ch2fft.png',
'Temp/Ch3fft.png', 'Temp/Ch4fft.png', FileName)

        os.remove("Temp/STFTTestingFigureCh1.png")
        os.remove("Temp/STFTTestingFigureCh2.png")
        os.remove("Temp/STFTTestingFigureCh3.png")
        os.remove("Temp/STFTTestingFigureCh4.png")
        os.remove('Temp/Ch1fft.png')
        os.remove('Temp/Ch2fft.png')
        os.remove('Temp/Ch3fft.png')
        os.remove('Temp/Ch4fft.png')


def Classifier(x):
    classification = x

    FolderSTFTTraining = "Figure/Training/"+ classification
    FolderSTFTTesting = "Figure/Testing/"+ classification
    FolderValidation = "Figure/Validation/" + classification

    if isdir(FolderSTFTTraining):
        print("Folders Already Excits!")
    else:
```

```
        os.mkdir(FolderSTFTTraining)
        os.mkdir(FolderSTFTTesting)
        os.mkdir(FolderValidation)
    print("Folder for", classification, "made!")
    return [FolderSTFTTraining, FolderSTFTTesting, FolderValidation]


def TestingPreprocessing(data):
    time = data[75:len(data)-1, 1]
    Ch1 = data[75:len(data)-1, 2]
    Ch2 = data[75:len(data) - 1, 3]
    Ch3 = data[75:len(data) - 1, 4]
    Ch4 = data[75:len(data) - 1, 5]
    Fs = 1000

    detenCh1 = sig.detrend(Ch1)
    detenCh2 = sig.detrend(Ch2)
    detenCh3 = sig.detrend(Ch3)
    detenCh4 = sig.detrend(Ch4)

    maxV1 = max(detenCh1)
    normV1 = detenCh1 / maxV1
    maxV2 = max(detenCh2)
    normV2 = detenCh2 / maxV2
    maxV3 = max(detenCh3)
    normV3 = detenCh3 / maxV3
    maxV4 = max(detenCh4)
    normV4 = detenCh4 / maxV4

    Ch1pp = normV1
    Ch2pp = normV2
    Ch3pp = normV3
    Ch4pp = normV4

    ## Channel 1
    fig0 = plt.figure()
    ax0 = plt.Axes(fig0, [0., 0., 1., 1.])
    plt.style.use('dark_background')
    plt.scatter(time, Ch1pp, c=abs(Ch1pp * Ch1pp), s=abs(Ch1pp))
    plt.gray()
    plt.axis('off')
    ax0.set_axis_off()
    plt.tight_layout()
    plt.show()
    fig0.savefig('ScattTestingFigureCh1.png', bbox_inches='tight',
pad_inches=0)

    f1, t1, Zxx1 = sig.stft(Ch1pp, Fs, nperseg=1000)
    fig1 = plt.figure(frameon=False)
```

```
    ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
    plt.pcolormesh(t1, f1, np.abs(Zxx1), shading='gouraud', cmap='gray')
    plt.axis('off')
    plt.ylim([90, 500])
    ax1.set_axis_off()
    plt.tight_layout()
    plt.show()
    fig1.savefig('STFTTestingFigureCh1.png', bbox_inches='tight',
pad_inches=0)

    ## Channel 2
    fig0 = plt.figure()
    ax0 = plt.Axes(fig0, [0., 0., 1., 1.])
    plt.style.use('dark_background')
    plt.scatter(time, Ch2pp, c=abs(Ch2pp * Ch2pp), s=abs(Ch2pp))
    plt.gray()
    plt.axis('off')
    ax0.set_axis_off()
    plt.tight_layout()
    plt.show()
    fig0.savefig('ScattTestingFigureCh2.png', bbox_inches='tight',
pad_inches=0)

    f2, t2, Zxx2 = sig.stft(Ch2pp, Fs, nperseg=1000)
    fig1 = plt.figure(frameon=False)
    ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
    plt.pcolormesh(t2, f2, np.abs(Zxx2), shading='gouraud', cmap='gray')
    plt.axis('off')
    plt.ylim([90, 500])
    ax1.set_axis_off()
    plt.tight_layout()
    plt.show()
    fig1.savefig('STFTTestingFigureCh2.png', bbox_inches='tight',
pad_inches=0)

    ##Channel 3
    fig0 = plt.figure()
    ax0 = plt.Axes(fig0, [0., 0., 1., 1.])
    plt.style.use('dark_background')
    plt.scatter(time, Ch3pp, c=abs(Ch3pp * Ch3pp), s=abs(Ch3pp))
    plt.gray()
    plt.axis('off')
    ax0.set_axis_off()
    plt.tight_layout()
    plt.show()
    fig0.savefig('ScattTestingFigureCh3.png', bbox_inches='tight',
pad_inches=0)
```

```
    f3, t3, Zxx3 = sig.stft(Ch3pp, Fs, nperseg=1000)
    fig1 = plt.figure(frameon=False)
    ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
    plt.pcolormesh(t3, f3, np.abs(Zxx3), shading='gouraud', cmap='gray')
    plt.axis('off')
    plt.ylim([90, 500])
    ax1.set_axis_off()
    plt.tight_layout()
    plt.show()
    fig1.savefig('STFTTestingFigureCh3.png', bbox_inches='tight',
pad_inches=0)

    ## Channel 4
    fig0 = plt.figure()
    ax0 = plt.Axes(fig0, [0., 0., 1., 1.])
    plt.style.use('dark_background')
    plt.scatter(time, Ch4pp, c=abs(Ch4pp * Ch4pp), s=abs(Ch4pp))
    plt.gray()
    plt.axis('off')
    ax0.set_axis_off()
    plt.tight_layout()
    plt.show()
    fig0.savefig('ScattTestingFigureCh4.png', bbox_inches='tight',
pad_inches=0)

    f4, t4, Zxx4 = sig.stft(Ch4pp, Fs, nperseg=1000)
    fig1 = plt.figure(frameon=False)
    ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
    plt.pcolormesh(t4, f4, np.abs(Zxx4), shading='gouraud', cmap='gray')
    plt.axis('off')
    plt.ylim([90, 500])
    ax1.set_axis_off()
    plt.tight_layout()
    plt.show()
    fig1.savefig('STFTTestingFigureCh4.png', bbox_inches='tight',
pad_inches=0)

def figure_makerMeth2(data, FolderSTFTTraining, FolderSTFTTesting,
FolderValTesting, FileName):
    ChannelNum = len(data[0]) - 1
    time = data[150:len(data)-1, 1]
    Ch1 = data[150:len(data)-1, 2]
    Ch2 = data[150:len(data) - 1, 3]
    Ch3 = data[150:len(data) - 1, 4]
    Ch4 = data[150:len(data) - 1, 5]
    Fs = 4000/3.5

    detenCh1 = sig.detrend(Ch1)
```

```
    detenCh2 = sig.detrend(Ch2)
    detenCh3 = sig.detrend(Ch3)
    detenCh4 = sig.detrend(Ch4)

    filter = sig.butter(2, [70,550], 'bandpass', fs=Fs, output='sos')
    corrCh1 = sig.sosfilt(filter, detenCh1)
    corrCh1 = (corrCh1*5)/1023
    corrCh2 = sig.sosfilt(filter, detenCh2)
    corrCh2 = (corrCh2*5)/1023
    corrCh3 = sig.sosfilt(filter, detenCh3)
    corrCh3 = (corrCh3*5)/1023
    corrCh4 = sig.sosfilt(filter, detenCh4)
    corrCh4 = (corrCh4*5)/1023

    TrainingDirectory = [f for f in listdir(FolderSTFTTraining)]
    TestDirectory = [f for f in listdir(FolderSTFTTesting)]
    ValidationDirectory = [f for f in listdir(FolderValTesting)]


    countTrain = int(len(TrainingDirectory))
    countTest = int(len(TestDirectory))
    countVal = int(len(ValidationDirectory))

    maxV1 = max(corrCh1)
    normV1 = corrCh1 / maxV1
    maxV2 = max(corrCh2)
    normV2 = corrCh2 / maxV2
    maxV3 = max(corrCh3)
    normV3 = corrCh3 / maxV3
    maxV4 = max(corrCh4)
    normV4 = corrCh4 / maxV4


    Num = random.randint(0, 10)
    seperationPoints = SepWithSTFT(normV1, normV2, normV3, normV4, time, Fs)
    if len(seperationPoints)>0:
        if len(seperationPoints) <= 2:
            Ch1pp =
normV1[int(seperationPoints[0]):int(seperationPoints[0]+Fs+10)]
            Ch2pp =
normV2[int(seperationPoints[0]):int(seperationPoints[0]+Fs+10)]
            Ch3pp =
normV3[int(seperationPoints[0]):int(seperationPoints[0]+Fs+10)]
            Ch4pp =
normV4[int(seperationPoints[0]):int(seperationPoints[0]+Fs+10)]
        if len(seperationPoints) > 2:
            Ch1pp =
normV1[int(seperationPoints[1]):int(seperationPoints[1]+Fs+10)]
```

```
            Ch2pp =
normV2[int(seperationPoints[1]):int(seperationPoints[1]+Fs+10)]
            Ch3pp =
normV3[int(seperationPoints[1]):int(seperationPoints[1]+Fs+10)]
            Ch4pp =
normV4[int(seperationPoints[1]):int(seperationPoints[1]+Fs+10)]
        ## Channel 1
        f1, t1, Zxx1 = sig.stft(Ch1pp, Fs, nperseg=75)
        fig1 = plt.figure(frameon=False)
        ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
        plt.pcolormesh(t1, f1, np.abs(Zxx1), shading='gouraud', cmap='gray')
        plt.axis('off')
        plt.ylim([70, 500])
        ax1.set_axis_off()
        plt.tight_layout()
        plt.show()
        fig1.savefig('Temp/STFTTestingFigureCh1.png', bbox_inches='tight',
pad_inches=0)

        ## Channel 2
        f2, t2, Zxx2 = sig.stft(Ch2pp, Fs, nperseg=75)
        fig1 = plt.figure(frameon=False)
        ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
        plt.pcolormesh(t2, f2, np.abs(Zxx2), shading='gouraud', cmap='gray')
        plt.axis('off')
        plt.ylim([70, 500])
        ax1.set_axis_off()
        plt.tight_layout()
        plt.show()
        fig1.savefig('Temp/STFTTestingFigureCh2.png', bbox_inches='tight',
pad_inches=0)

        ##Channel 3
        f3, t3, Zxx3 = sig.stft(Ch3pp, Fs, nperseg=75)
        fig1 = plt.figure(frameon=False)
        ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
        plt.pcolormesh(t3, f3, np.abs(Zxx3), shading='gouraud', cmap='gray')
        plt.axis('off')
        plt.ylim([70, 500])
        ax1.set_axis_off()
        plt.tight_layout()
        plt.show()
        fig1.savefig('Temp/STFTTestingFigureCh3.png', bbox_inches='tight',
pad_inches=0)

        ## Channel 4
        f4, t4, Zxx4 = sig.stft(Ch4pp, Fs, nperseg=75)
        fig1 = plt.figure(frameon=False)
```

```
        ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
        plt.pcolormesh(t4, f4, np.abs(Zxx4), shading='gouraud', cmap='gray')
        plt.axis('off')
        plt.ylim([90, 500])
        ax1.set_axis_off()
        plt.tight_layout()
        plt.show()
        fig1.savefig('Temp/STFTTestingFigureCh4.png', bbox_inches='tight',
pad_inches=0)

        if Num in range(0, 7):
            nameTrain = str(countTrain)
            countTrain += 1
            FileName = FolderSTFTTraining + "/" + nameTrain + '.png'
            print(FileName)
            ProcessConcat('Temp/STFTTestingFigureCh1.png',
'Temp/STFTTestingFigureCh2.png', 'Temp/STFTTestingFigureCh3.png',
'Temp/STFTTestingFigureCh4.png', FileName)

        if Num in range(7, 9):
            nameVal = str(countVal)
            countVal += 1
            FileName = FolderValTesting + "/" + nameVal + '.png'
            print(FileName)
            ProcessConcat('Temp/STFTTestingFigureCh1.png',
'Temp/STFTTestingFigureCh2.png', 'Temp/STFTTestingFigureCh3.png',
'Temp/STFTTestingFigureCh4.png', FileName)

        if Num in range(9, 11 ):
            nameTest = str(countTest)
            countTest += 1
            FileName = FolderSTFTTesting + "/" + nameTest + '.png'
            print(FileName)
            ProcessConcat('Temp/STFTTestingFigureCh1.png',
'Temp/STFTTestingFigureCh2.png', 'Temp/STFTTestingFigureCh3.png',
'Temp/STFTTestingFigureCh4.png', FileName)

        os.remove("Temp/STFTTestingFigureCh1.png")
        os.remove("Temp/STFTTestingFigureCh2.png")
        os.remove("Temp/STFTTestingFigureCh3.png")
        os.remove("Temp/STFTTestingFigureCh4.png")


def figure_makerTesting(data):
    time = data[150:len(data)-1, 1]
    Ch1 = data[150:len(data)-1, 2]
    Ch2 = data[150:len(data) - 1, 3]
    Ch3 = data[150:len(data) - 1, 4]
    Ch4 = data[150:len(data) - 1, 5]
```

```
    Fs = 4000/3.5

    detenCh1 = sig.detrend(Ch1)
    detenCh2 = sig.detrend(Ch2)
    detenCh3 = sig.detrend(Ch3)
    detenCh4 = sig.detrend(Ch4)

    filter = sig.butter(2, [70,550], 'bandpass', fs=Fs, output='sos')
    corrCh1 = sig.sosfilt(filter, detenCh1)
    corrCh1 = (corrCh1*5)/1023
    corrCh2 = sig.sosfilt(filter, detenCh2)
    corrCh2 = (corrCh2*5)/1023
    corrCh3 = sig.sosfilt(filter, detenCh3)
    corrCh3 = (corrCh3*5)/1023
    corrCh4 = sig.sosfilt(filter, detenCh4)
    corrCh4 = (corrCh4*5)/1023

    maxV1 = max(corrCh1)
    normV1 = corrCh1 / maxV1
    maxV2 = max(corrCh2)
    normV2 = corrCh2 / maxV2
    maxV3 = max(corrCh3)
    normV3 = corrCh3 / maxV3
    maxV4 = max(corrCh4)
    normV4 = corrCh4 / maxV4

    seperationPoints = SepWithSTFT(normV1, normV2, normV3, normV4, time, Fs)
    if len(seperationPoints) > 1:
        if len(seperationPoints) <= 2:
            Ch1pp =
normV1[int(seperationPoints[0]):int(seperationPoints[0]+Fs+10)]
            Ch2pp =
normV2[int(seperationPoints[0]):int(seperationPoints[0]+Fs+10)]
            Ch3pp =
normV3[int(seperationPoints[0]):int(seperationPoints[0]+Fs+10)]
            Ch4pp =
normV4[int(seperationPoints[0]):int(seperationPoints[0]+Fs+10)]
        if len(seperationPoints) > 2:
            Ch1pp =
normV1[int(seperationPoints[1]):int(seperationPoints[1]+Fs+10)]
            Ch2pp =
normV2[int(seperationPoints[1]):int(seperationPoints[1]+Fs+10)]
            Ch3pp =
normV3[int(seperationPoints[1]):int(seperationPoints[1]+Fs+10)]
            Ch4pp =
normV4[int(seperationPoints[1]):int(seperationPoints[1]+Fs+10)]
```

```
        ## Channel 1
        f1, t1, Zxx1 = sig.stft(Ch1pp, Fs, nperseg=150)
        fig1 = plt.figure(frameon=False)
        ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
        plt.pcolormesh(t1, f1, np.abs(Zxx1), shading='gouraud', cmap='gray')
        plt.axis('off')
        plt.ylim([70, 500])
        ax1.set_axis_off()
        plt.tight_layout()
        # plt.show()
        fig1.savefig('Temp/STFTTestingFigureCh1.png', bbox_inches='tight',
pad_inches=0)

        ## Channel 2
        f2, t2, Zxx2 = sig.stft(Ch2pp, Fs, nperseg=150)
        fig1 = plt.figure(frameon=False)
        ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
        plt.pcolormesh(t2, f2, np.abs(Zxx2), shading='gouraud', cmap='gray')
        plt.axis('off')
        plt.ylim([70, 500])
        ax1.set_axis_off()
        plt.tight_layout()
        # plt.show()
        fig1.savefig('Temp/STFTTestingFigureCh2.png', bbox_inches='tight',
pad_inches=0)

        ##Channel 3
        f3, t3, Zxx3 = sig.stft(Ch3pp, Fs, nperseg=150)
        fig1 = plt.figure(frameon=False)
        ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
        plt.pcolormesh(t3, f3, np.abs(Zxx3), shading='gouraud', cmap='gray')
        plt.axis('off')
        plt.ylim([70, 500])
        ax1.set_axis_off()
        plt.tight_layout()
        # plt.show()
        fig1.savefig('Temp/STFTTestingFigureCh3.png', bbox_inches='tight',
pad_inches=0)

        ## Channel 4
        f4, t4, Zxx4 = sig.stft(Ch4pp, Fs, nperseg=150)
        fig1 = plt.figure(frameon=False)
        ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
        plt.pcolormesh(t4, f4, np.abs(Zxx4), shading='gouraud', cmap='gray')
        plt.axis('off')
        plt.ylim([90, 500])
        ax1.set_axis_off()
        plt.tight_layout()
```

```
        fig1.savefig('Temp/STFTTestingFigureCh4.png', bbox_inches='tight',
pad_inches=0)

        FileName = "Temp/Test.png"
        print(FileName)
        ProcessConcat('Temp/STFTTestingFigureCh1.png',
'Temp/STFTTestingFigureCh2.png', 'Temp/STFTTestingFigureCh3.png',
'Temp/STFTTestingFigureCh4.png', FileName)
        os.remove("Temp/STFTTestingFigureCh1.png")
        os.remove("Temp/STFTTestingFigureCh2.png")
        os.remove("Temp/STFTTestingFigureCh3.png")
        os.remove("Temp/STFTTestingFigureCh4.png")

def figure_makerMeth2AutoSize(data, FolderSTFTTraining, FolderSTFTTesting,
FolderValTesting, FileName):

    TrainingDirectory = [f for f in listdir(FolderSTFTTraining)]
    TestDirectory = [f for f in listdir(FolderSTFTTesting)]
    ValidationDirectory = [f for f in listdir(FolderValTesting)]

    countTrain = int(len(TrainingDirectory))
    countTest = int(len(TestDirectory))
    countVal = int(len(ValidationDirectory))
    data = np.delete(data, 0, 1)
    ChannelNum = len(data[0]) - 1
    print(ChannelNum)
    Fs = 4000 / 3.5
    time = data[200:len(data) - 1, 0]
    row = len(time)
    print(row)
    col = len(data[0]) - 1
    print(col)
    normData = np.empty(shape=(row, col))
    for ch in range(ChannelNum):
        Detrend = sig.detrend(data[200:len(data) - 1, ch + 1])
        filter = sig.butter(2, [70, 550], 'bandpass', fs=Fs, output='sos')
        corrdet = sig.sosfilt(filter, Detrend)
        corrdet = (corrdet * 5) / 1023

        maxVal = max(corrdet)
        normData[:, ch] = corrdet #/ maxVal

    Num = random.randint(0, 10)
    seperationPoints = SepWithSTFTAuto(normData, time, Fs)

    if len(seperationPoints)>1 and len(seperationPoints)<=5:
        for ch in range(ChannelNum):
```

```
            if len(seperationPoints) <= 2:
                Ch1pp =
normData[int(seperationPoints[0]):int(seperationPoints[0]+Fs+10), ch]

            if len(seperationPoints) > 2:
                Ch1pp =
normData[int(seperationPoints[1]):int(seperationPoints[1]+Fs+10), ch]
            f1, t1, Zxx1 = sig.stft(Ch1pp, Fs, nperseg=75)
            fig1 = plt.figure(frameon=False)
            ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
            plt.pcolormesh(t1, f1, np.abs(Zxx1), shading='gouraud',
cmap='gray')
            plt.axis('off')
            plt.ylim([70, 500])
            ax1.set_axis_off()
            plt.tight_layout()
            plt.show()
            Figname = 'Temp/STFTTestingFigure'+ str(ch) + '.png'
            fig1.savefig(Figname, bbox_inches='tight', pad_inches=0)

        if Num in range(0, 7):
            nameTrain = str(countTrain)
            countTrain += 1
            FileName = FolderSTFTTraining + "/" + nameTrain + '.png'
            Figures = os.listdir("Temp/")
            print(FileName)
            print(Figures)
            ProcessConcatAuto(Figures, FileName)
        if Num in range(7, 9):
            nameVal = str(countVal)
            countVal += 1
            FileName = FolderValTesting + "/" + nameVal + '.png'
            Figures = os.listdir("Temp/")
            print(FileName)
            print(Figures)
            ProcessConcatAuto(Figures, FileName)
        if Num in range(9, 11):
            nameTest = str(countTest)
            countTest += 1
            FileName = FolderSTFTTesting + "/" + nameTest + '.png'
            Figures = os.listdir("Temp/")
            print(FileName)
            print(Figures)
            ProcessConcatAuto(Figures, FileName)
        for i in Figures:
            FigureName = "Temp/"+i
            os.remove(FigureName)
```

F.4. SectionCutting.py

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import scipy.signal as sig
from random import randint

def SectionCutting(V, t, FileName):
    seperation = []
    seperation.append(0)
    time = t
    voltage = V

    uplowBoundNull = 0.1
    perofMaxVal = 0.7
    binSize = 100
    perNullDen = 0.6
    perMaxDen = 0.7

    t = time[0: 70000]
    fderenV = V[0: 70000]

    maxVal = max(fderenV)

    Nullpeaks, _ = sig.find_peaks(fderenV, height=(-uplowBoundNull,
uplowBoundNull))
    peaks, _ = sig.find_peaks(fderenV, height=perofMaxVal*maxVal)
    plt.plot(t, fderenV, linewidth=0.05)
    plt.scatter(t[peaks], fderenV[peaks], color='red')
    plt.show()

    f, (a0, a1, a2) = plt.subplots(3, 1, gridspec_kw={'height_ratios':
[10,3,3]})

    a0.plot(t, fderenV, linewidth=0.1, label="Signal")
    a0.scatter(t[Nullpeaks], fderenV[Nullpeaks], color='red', label="Low
amplitude peaks")
    a0.scatter(t[peaks], fderenV[peaks], color='green', label="Large amplitude
peak")
    NullpeakDensity, NullpeakRange, _ = a1.hist(Nullpeaks, bins=binSize,
facecolor='r', alpha=1, edgecolor='k', linewidth=1)
    peakDensity, peakRange, _ = a2.hist(peaks, bins=binSize, facecolor='g',
alpha=1, edgecolor='k', linewidth=1)
    plt.show()

    NullmaxDensity  = max(NullpeakDensity)
    maxDensity  = max(peakDensity)
    interestingNullPeakDis = []
```

```
    interestingPeakDis = []
    for i in range(len(peakDensity)):
        if NullpeakDensity[i]>= perNullDen* NullmaxDensity:
            interestingNullPeakDis.append(i)

    for i in range(len(peakDensity)):
        if peakDensity[i]>= perMaxDen*maxDensity:
            interestingPeakDis.append(i)

    for i in range(len(interestingNullPeakDis)):
        if interestingNullPeakDis[i] in interestingPeakDis:
            interestingNullPeakDis[i] = ''
    chekcer = True
    while chekcer:
        if '' in interestingNullPeakDis:
            interestingNullPeakDis.remove('')
        else:
            chekcer = False

    NullpeakRange = np.insert(NullpeakRange, 0, 0)


    for i in range(len(interestingNullPeakDis)):
        place = interestingNullPeakDis[i]+2
        seperation.append(int((NullpeakRange[place])))

    for i in range(5):
        for i in range(0,len(seperation)-1):
            if seperation[i]+3000 >= seperation[i+1]:
                NewSep = (seperation[i] + seperation[i+1])/2 - randint(50,
500)
                seperation[i] = ''
                seperation[i+1] = int(NewSep)

        chekcer = True
        while chekcer:
            if '' in seperation:
                seperation.remove('')
            else:
                chekcer = False

    from mutagen.wave import WAVE
    fileName = FileName.replace("TrainingData\\", '')
    AudioFileName = fileName.split('-')
    AudioFileName = "AudioOriginal/"+AudioFileName[0]+'.wav'

    audio = WAVE(AudioFileName)
    audio_info = audio.info
    length = int(audio_info.length)
```

```
    for i in range(0,len(seperation)-1):
        if seperation[i] + length*4000*0.7 >= seperation[i + 1]:
            seperation[i+1] = 0

    chekcer = True
    while chekcer:
        if 0 in seperation:
            seperation.remove(0)
        else:
            chekcer = False
    fig1 = plt.figure()
    plt.plot(t, fderenV, linewidth=0.1, label="Signal")
    plt.scatter(t[seperation], fderenV[seperation], color='red', label =
"Seperation Points")
    plt.show()

    displacemetData = 750
    whileChecker = True
    while whileChecker:
        if seperation[len(seperation)-1]+70000<=len(voltage):
            t = time[seperation[len(seperation)-1]-displacemetData:
seperation[len(seperation)-1]+80000]
            fderenV = voltage[seperation[len(seperation)-1]-displacemetData:
seperation[len(seperation)-1]+80000]

            maxVal = max(fderenV)

            NewSection = []

            Nullpeaks, _ = sig.find_peaks(fderenV, height=(-uplowBoundNull,
uplowBoundNull))
            peaks, _ = sig.find_peaks(fderenV, height=perofMaxVal * maxVal)


            plt.plot(t, fderenV, linewidth=0.05)
            plt.scatter(t[Nullpeaks], fderenV[Nullpeaks], color='red')
            plt.scatter(t[peaks], fderenV[peaks], color='green')

            NullpeakDensity, NullpeakRange, _ = plt.hist(Nullpeaks,
bins=binSize, facecolor='r', alpha=1, edgecolor='k', linewidth=1)

            peakDensity, peakRange, _ = plt.hist(peaks, bins=binSize,
facecolor='g', alpha=1, edgecolor='k', linewidth=1)
            plt.close()

            NullmaxDensity = max(NullpeakDensity)
            maxDensity = max(peakDensity)

            interestingNullPeakDis = []
```

```
            interestingPeakDis = []
            for i in range(len(peakDensity)):
                if NullpeakDensity[i] >= perNullDen * NullmaxDensity:
                    interestingNullPeakDis.append(i)

            for i in range(len(peakDensity)):
                if peakDensity[i] >= perMaxDen * maxDensity:
                    interestingPeakDis.append(i)

            for i in range(len(interestingNullPeakDis)):
                if interestingNullPeakDis[i] in interestingPeakDis:
                    interestingNullPeakDis[i] = ''
            chekcer = True
            while chekcer:
                if '' in interestingNullPeakDis:
                    interestingNullPeakDis.remove('')
                else:
                    chekcer = False

            NullpeakRange = np.insert(NullpeakRange, 0, 0)

            for i in range(len(interestingNullPeakDis)):
                place = interestingNullPeakDis[i] + 2
                NewSection.append(int((NullpeakRange[place])))

            for j in range(10):
                for i in range(len(NewSection) - 1):
                    if NewSection[i] + length * 4000*0.7 >= NewSection[i + 1]
and NewSection[i] + length * 4000*0.7 <= NewSection[i + 1]:
                        NewSection[i + 1] = 0

                chekcer = True
                while chekcer:
                    if 0 in NewSection:
                        NewSection.remove(0)
                    else:
                        chekcer = False

            for j in range(10):
                for i in range(0, len(NewSection) - 1):
                    if NewSection[i] + 2000 >= NewSection[i + 1]:
                        NewSep = (NewSection[i] + NewSection[i + 1]) / 2 -
randint(50, 100)
                        NewSection[i] = ''
                        NewSection[i + 1] = int(NewSep)

                chekcer = True
                while chekcer:
```

```
                    if '' in NewSection:
                        NewSection.remove('')
                    else:
                        chekcer = False

            for i in range(2):
                for i in range(len(NewSection) - 1):
                    if NewSection[i] + length * 4000*0.7 >= NewSection[i + 1]:
                        NewSection[i + 1] = 0

                chekcer = True
                while chekcer:
                    if 0 in NewSection:
                        NewSection.remove(0)
                    else:
                        chekcer = False

            plt.plot(t, fderenV, linewidth=0.05)
            plt.scatter(t[NewSection], fderenV[NewSection], color='red')
            plt.show()
            previousSep = seperation[len(seperation) - 1]
            for i in range(1,len(NewSection)):
                seperation.append(NewSection[i]+previousSep-displacemetData)
        else:
            whileChecker = False
    return(seperation)


def SectionCuttingTesting(V, t):
    seperation = []

    t = t
    fderenV = V

    maxVal = max(fderenV)

    Nullpeaks, _ = sig.find_peaks(fderenV, height=(-0.07, 0.07))
    Nullpeaks = np.transpose(Nullpeaks)
    peaks, _ = sig.find_peaks(fderenV, height=0.5*maxVal)


    f, (a0, a1, a2) = plt.subplots(3, 1, gridspec_kw={'height_ratios': [7, 2,
2]})
    a0.plot(t, fderenV, linewidth=0.05)

    NullpeakDensity, NullpeakRange, _ = a1.hist(Nullpeaks, bins=60,
facecolor='r', alpha=1, edgecolor='k', linewidth=1)
```

```
    peakDensity, peakRange, _ = a2.hist(peaks, bins=60, facecolor='g',
alpha=1, edgecolor='k', linewidth=1)
    f.tight_layout()

    NullmaxDensity  = max(NullpeakDensity)
    maxDensity  = max(peakDensity)
    interestingNullPeakDis = []
    interestingPeakDis = []
    for i in range(len(peakDensity)):
        if NullpeakDensity[i]>= 0.8* NullmaxDensity:
            interestingNullPeakDis.append(i)

    for i in range(len(peakDensity)):
        if peakDensity[i]>= 0.5* maxDensity:
            interestingPeakDis.append(i)


    for i in range(len(interestingNullPeakDis)):
        if interestingNullPeakDis[i] in interestingPeakDis:
            interestingNullPeakDis[i] = ''
    chekcer = True
    while chekcer:
        if '' in interestingNullPeakDis:
            interestingNullPeakDis.remove('')
        else:
            chekcer = False

    NullpeakRange = np.insert(NullpeakRange, 0, 0)

    for i in range(len(interestingNullPeakDis)):
        place = interestingNullPeakDis[i]+2
        seperation.append(int((NullpeakRange[place])))

    for i in range(5):
        for i in range(0,len(seperation)-1):
            if seperation[i]+3000 >= seperation[i+1]:
                NewSep = (seperation[i] + seperation[i+1])/2 - randint(50,
500)
                seperation[i] = ''
                seperation[i+1] = int(NewSep)

        chekcer = True
        while chekcer:
            if '' in seperation:
                seperation.remove('')
            else:
                chekcer = False
    diffLow = []
    return(seperation)
```

```
def SepWithSTFT(normV1, normV2, normV3, normV4, t, Fs):
    inst = 100
    f1, t1, Zxx1 = sig.stft(normV1, Fs, nperseg=inst)
    fig1 = plt.figure(frameon=False)
    ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
    plt.pcolormesh(t1, f1, np.abs(Zxx1), shading='gouraud')
    ax1.set_axis_off()
    plt.tight_layout()


    f2, t2, Zxx2 = sig.stft(normV2, Fs, nperseg=inst)
    fig1 = plt.figure(frameon=False)
    ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
    plt.pcolormesh(t2, f2, np.abs(Zxx2), shading='gouraud')
    ax1.set_axis_off()
    plt.tight_layout()
    plt.show()

    f3, t3, Zxx3 = sig.stft(normV3, Fs, nperseg=inst)
    fig1 = plt.figure(frameon=False)
    ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
    plt.pcolormesh(t3, f3, np.abs(Zxx3), shading='gouraud')
    ax1.set_axis_off()
    plt.tight_layout()
    plt.show()

    f4, t4, Zxx4 = sig.stft(normV4, Fs, nperseg=inst)
    fig1 = plt.figure(frameon=False)
    ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
    plt.pcolormesh(t4, f4, np.abs(Zxx4), shading='gouraud')
    ax1.set_axis_off()
    plt.tight_layout()
    plt.show()

    z1 = abs(Zxx1)
    z2 = abs(Zxx2)
    z3 = abs(Zxx3)
    z4 = abs(Zxx4)
    count = 0

    for i in f1:
        if round(i) == 80:
            #print(count)
            minFreq = count
            continue
        count = count + 1

    maxz1 = max(z1[minFreq])
    maxz2 = max(z2[minFreq])
```

```
    maxz3 = max(z3[minFreq])
    maxz4 = max(z4[minFreq])
    z = [z1[minFreq], z2[minFreq], z3[minFreq], z4[minFreq]]
    Zs = [maxz1, maxz2, maxz3, maxz4]
    maxZ = max(Zs)
    indexZ = Zs.index(maxZ)
    z = z[indexZ]
    upperLim = max(z) * 0.55
    Nullpeaks, _ = sig.find_peaks(z, height=upperLim)
    plt.plot(t1, z)
    plt.scatter(t1[Nullpeaks], z[Nullpeaks])

    def truncate(n, decimals=0):
        multiplier = 10 ** decimals
        return int(n * multiplier) / multiplier

    plt.plot(t, normV1, label="Channel 1", linewidth=0.5, alpha=0.3)
    plt.scatter(t1[Nullpeaks], z[Nullpeaks], color='k')
    # plt.show()

    adjPe = inst / 2
    Sectioning = Nullpeaks*adjPe
    plt.close()
    return Sectioning

def SepWithSTFTAuto(data, time, Fs):
    inst = 100
    normV1 = data[0:len(time), 0]
    f, t, Zxx = sig.stft(normV1, Fs, nperseg=inst)
    fig = plt.figure(frameon=False)
    ax1 = plt.Axes(fig, [0., 0., 1., 1.])
    plt.pcolormesh(t, f, np.abs(Zxx), shading='gouraud')
    ax1.set_axis_off()
    plt.tight_layout()
    plt.show()

    z1 = abs(Zxx)

    count = 0
    for i in f:
        if round(i) == 80:
            minFreq = count
            continue
        count = count + 1

    z = z1[minFreq]

    upperLim = max(z) * 0.50
```

```
    Nullpeaks, _ = sig.find_peaks(z, height=upperLim)
    plt.plot(t, z)
    plt.scatter(t[Nullpeaks], z[Nullpeaks])
    plt.show()

    def truncate(n, decimals=0):
        multiplier = 10 ** decimals
        return int(n * multiplier) / multiplier

    plt.plot(time, normV1, label="Channel 1", linewidth=0.5, alpha=0.3)
    plt.scatter(t[Nullpeaks], z[Nullpeaks], color='k')
    plt.show()
    print(Nullpeaks)

    if len(Nullpeaks) >1:
        adjPe = int(inst / 2)

        Sectioning = Nullpeaks*adjPe
        f, t, Zxx = sig.stft(normV1[Nullpeaks[0] * adjPe: Nullpeaks[1] *
adjPe], Fs, nperseg=inst)
        fig1 = plt.figure(frameon=False)
        ax1 = plt.Axes(fig1, [0., 0., 1., 1.])
        plt.pcolormesh(t, f, np.abs(Zxx), shading='gouraud')
        ax1.set_axis_off()
        plt.tight_layout()
        plt.show()
    else:
        Sectioning = [0]
    return Sectioning
```

F.5. MachineLearning.py

```
import matplotlib.pyplot as plt
import seaborn as sns
import os
os.environ["CUDA_VISIBLE_DEVICES"]="-1"
import tensorflow.keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten,
Dropout, Softmax
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from Labeler_Data import get_data
from sklearn.metrics import classification_report,confusion_matrix
from keras.utils.vis_utils import plot_model
import pydot
from datetime import datetime
from packaging import version
from tensorflow.keras import regularizers

import tensorflow as tf

import numpy as np

def CNN_Training(folderTraining, folderTesting, ep, LearningRate, dataType,
img_size, label):
    #img_size = 32
    CAT = label
    trainData = get_data(folderTraining, img_size, labels=CAT)

    x_train = []
    y_train = []



    for feature, label in trainData:
      x_train.append(feature)
      y_train.append(label)

    # Normalize the data
    x_train = np.array(x_train) / 255

    x_train.reshape(-1, img_size, img_size, 1)
    y_train = np.array(y_train)

    testData = get_data(folderTesting, img_size, labels= CAT)
    x_val = []
    y_val = []
```

```
    for feature, label in testData:
        x_val.append(feature)
        y_val.append(label)

    x_val = np.array(x_val) / 255

    x_val.reshape(-1, img_size, img_size, 1)
    y_val = np.array(y_val)

    num_labels = len(CAT)
    datagen = ImageDataGenerator(
            featurewise_center=False,  # set input mean to 0 over the dataset
            samplewise_center=False,  # set each sample mean to 0
            featurewise_std_normalization=False,  # divide inputs by std of
the dataset
            samplewise_std_normalization=False,  # divide each input by its
std
            zca_whitening=False,  # apply ZCA whitening
            rotation_range = False,  # randomly rotate images in the range
(degrees, 0 to 180)
            zoom_range = False,#0.3, # Randomly zoom image
            width_shift_range=False,#0.2,  # randomly shift images
horizontally (fraction of total width)
            height_shift_range=False,  # randomly shift images vertically
(fraction of total height)
            horizontal_flip = False,  # randomly flip images
            vertical_flip=False)  # randomly flip images


    datagen.fit(x_train)
    logdir = "logs/fit/" + datetime.now().strftime("%Y%m%d-%H%M%S")
    tensorboard_callback =
tensorflow.keras.callbacks.TensorBoard(log_dir=logdir)

    model = Sequential()
    model.add(Conv2D(35, 5, padding="same", activation="relu",
input_shape=x_train.shape[1:]))
    model.add(MaxPool2D(pool_size=(2, 2), strides=3))
    model.add(Dropout(0.25))

    model.add(Conv2D(65, 5, padding="same", activation="relu"))
    model.add(MaxPool2D())
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(128, activation="sigmoid",
                    kernel_regularizer= regularizers.L1L2(l1=1e-5, l2=1e-4),
                    bias_regularizer= regularizers.L2(1e-4),
                    activity_regularizer= regularizers.L2(1e-5)))
```

```
    model.add(Dense(128, activation="relu",
                    kernel_regularizer= regularizers.L1L2(l1=1e-5, l2=1e-4),
                    bias_regularizer= regularizers.L2(1e-4),
                    activity_regularizer= regularizers.L2(1e-5)))

    model.add(Dropout(0.25))
    model.add(Dense(num_labels))

    model.summary()

    opt = Adam(lr=LearningRate)
    model.compile(optimizer = opt , loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True) , metrics =
['accuracy'])

    history = model.fit(x_train,y_train,epochs = ep , validation_data =
(x_val, y_val), callbacks=[tensorboard_callback])

    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs_range = range(ep)

    fig = plt.figure(figsize=(15, 15))
    plt.subplot(2, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    plt.subplot(2, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')
    plt.show()
    Name = "Figures/" + dataType+"AccurracyandErrorPlot.png"
    fig.savefig(Name, transparent=True, bbox_inches='tight')

    model.summary()
    return model, acc, val_acc, loss, val_loss

def TestingNetwrok(model, folderTesting, img_size, labels):
    testData = get_data(folderTesting, img_size, labels)
    x_val = []
    y_val = []
```

```
    for feature, label in testData:
      x_val.append(feature)
      y_val.append(label)

    x_val = np.array(x_val) / 255

    x_val.reshape(-1, img_size, img_size, 1)
    y_val = np.array(y_val)

    #CAT = ['A', 'B', 'C', 'D', 'E', 'F']
    CAT = labels
    DegNum = int(labels[1])
    DegNum = str(DegNum)
    predictions = model.predict(x_val)
    predictions = np.argmax(predictions,axis=1)
    print(classification_report(y_val, predictions, target_names = CAT))
    fig = plt.figure()
    confusion_mtx = tf.math.confusion_matrix(y_val, predictions)
    sns.heatmap(confusion_mtx, xticklabels=CAT, yticklabels=CAT,
                annot=True, fmt='g')
    plt.rc('font', family='Helvetica')
    plt.xlabel('Prediction',fontsize=20)
    plt.xticks(rotation=90)
    plt.ylabel('Label',fontsize=20)
    plt.yticks(rotation=90)
    plt.show()
    Name = "Figures/" + DegNum + " Confusion Matrix.png"
    fig.savefig(Name, transparent=True, bbox_inches='tight')
    return y_val, predictions

def Save_CNN(model, Name):
    json_model = model.to_json()

    with open(Name, 'w') as json_file:
        json_file.write(json_model)

def CNN_TrainingImprove(folderTraining, folderTesting, ep, LearningRate,
dataType, img_size, label):
    #img_size = 32
    CAT = label
    trainData = get_data(folderTraining, img_size, labels=CAT)

    x_train = []
    y_train = []


    for feature, label in trainData:
      x_train.append(feature)
      y_train.append(label)
```

```
    # Normalize the data
    x_train = np.array(x_train) / 255

    x_train.reshape(-1, img_size, img_size, 1)
    y_train = np.array(y_train)

    testData = get_data(folderTesting, img_size, labels= CAT)
    x_val = []
    y_val = []

    for feature, label in testData:
        x_val.append(feature)
        y_val.append(label)

    x_val = np.array(x_val) / 255

    x_val.reshape(-1, img_size, img_size, 1)
    y_val = np.array(y_val)

    num_labels = len(CAT)
    datagen = ImageDataGenerator(
            featurewise_center=False,  # set input mean to 0 over the dataset
            samplewise_center=False,  # set each sample mean to 0
            featurewise_std_normalization=False,  # divide inputs by std of
the dataset
            samplewise_std_normalization=False,  # divide each input by its
std
            zca_whitening=False,  # apply ZCA whitening
            rotation_range = False,  # randomly rotate images in the range
(degrees, 0 to 180)
            zoom_range = False,#0.3, # Randomly zoom image
            width_shift_range=False,#0.2,  # randomly shift images
horizontally (fraction of total width)
            height_shift_range=False,  # randomly shift images vertically
(fraction of total height)
            horizontal_flip = False,  # randomly flip images
            vertical_flip=False)  # randomly flip images


    datagen.fit(x_train)

    logdir = "logs/fit/" + datetime.now().strftime("%Y%m%d-%H%M%S")
    tensorboard_callback =
tensorflow.keras.callbacks.TensorBoard(log_dir=logdir)

    model = Sequential()
    model.add(Conv2D(8, 20,padding="same", activation="relu",
input_shape=x_train.shape[1:]))
    model.add(MaxPool2D())
```

```
    model.add(Flatten())
    model.add(Dense(128,activation="relu"))
    model.add(Dropout(0.25))
    model.add(Dense(num_labels))
    model = tf.keras.models.load_model("DirectionRecCNN")
    model.summary()

    opt = Adam(lr=LearningRate, beta1=0.9, beta2=0.999, epsilon=1e-08,)
    model.compile(optimizer = opt , loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True) , metrics =
['accuracy'])

    history = model.fit(x_train, y_train, epochs = ep , validation_data =
(x_val, y_val), callbacks=[tensorboard_callback])

    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs_range = range(ep)

    fig = plt.figure(figsize=(15, 15))
    plt.subplot(2, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    plt.subplot(2, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')
    plt.show()
    Name = dataType+"AccurracyandErrorPlot.png"
    fig.savefig(Name, transparent=True, bbox_inches='tight')

    model.summary()
    return model, acc, val_acc, loss, val_loss
```

F.6. Labeler_Data.py

```
import cv2
import os
import numpy as np

def get_data(data_dir, img_size, labels):
    data = []
    for label in labels:
        path = os.path.join(data_dir, label)
        class_num = labels.index(label)
        for img in os.listdir(path):
            try:
                img_arr = cv2.imread(os.path.join(path, img))[...,::-1]
                resized_arr = cv2.resize(img_arr, (img_size, img_size)) #
                data.append([resized_arr, class_num])
            except Exception as e:
                print(e)
    return np.array(data)
```

Appendix G

Python code for real-time testing of the asymmetrical multichannel sensor with the GitHub link: https://github.com/abmoineddini/MPhil_sound_localisation

Spatial Recognition.

```
import serial
import numpy as np
import time
from csv import writer
from Preprocessor import *
import cv2
import tensorflow as tf
from DataCollector import *
from tkinter import *
from DataSender import *
import random as rnd
import winsound

ws = Tk()
ws.title('Directional Recognition')
ws.geometry('1000x0750')
ws.config(bg='#000000')

mylabel = Label(ws,
                text="...",
                bg='#000000',
                fg='#ffffff',
                font='Times 100',
                width=50,
                height=10)

mylabel.pack()
ws.update()

countDown = 100
test = True

model = tf.keras.models.load_model("DirectionRecCNN")

FileAdd = []
COMPortMotor = "COM5"
currAng, MotorController = Inititialise(COMPortMotor)
time.sleep(1)
ws.update()
labels = os.listdir("Figure/Training")
labelsInt = []
for i in labels:
    labelsInt.append(int(i))
labelsInt.sort()
labelsOrd = []
```

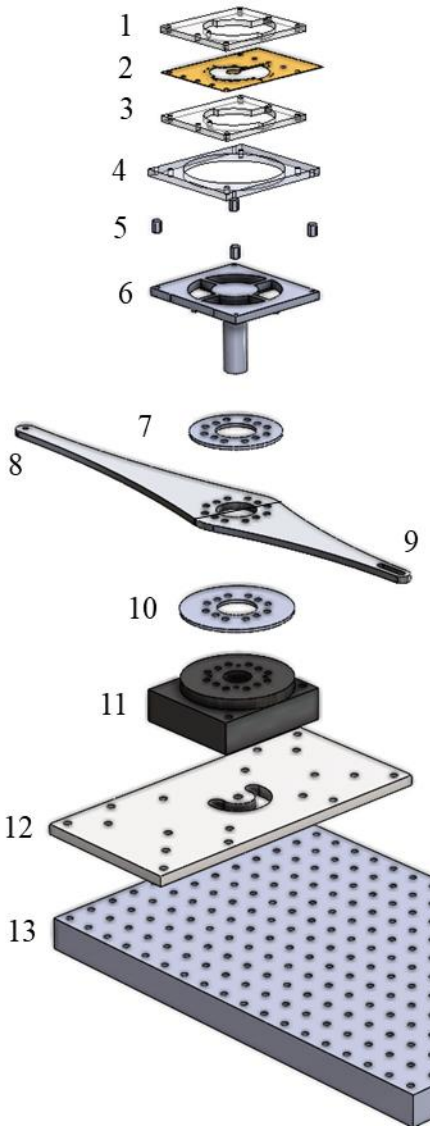
```
for i in labelsInt:
    labelsOrd.append(str(i))
Increment = labelsInt[1]-labelsInt[0]
COMPort = 'COM6'
while test:
    div = rnd.randint(0, 12)
    angle = div * Increment

    currAng = AngleSet(angle, MotorController, currAng)
    Name = str(currAng)
    time.sleep(3)
    Period = 3
    [Channel1, Channel2, Channel3, Channel4, Time] = collectDataTest(COMPort,
Period)
    plt.show()
    plt.plot(Time, Channel1)
    plt.plot(Time, Channel2)
    plt.plot(Time, Channel3)
    plt.plot(Time, Channel4)
    plt.show()
    df = pd.read_csv("Temp/Test.csv")
    data = df.to_numpy()
    figure_makerTesting(data)
    PnGFile = "Temp/Test.png"
    while not os.path.isfile(PnGFile):
        [Channel1, Channel2, Channel3, Channel4, Time] =
collectDataTest(COMPort, Period)
        plt.plot(Time, Channel1)
        plt.plot(Time, Channel2)
        plt.plot(Time, Channel3)
        plt.plot(Time, Channel4)
        plt.show()
        df = pd.read_csv("Temp/Test.csv")
        data = df.to_numpy()
        figure_makerTesting(data)
    img_size = 64

    def Preprocess(path):
        img_arr = cv2.imread(path)[..., ::-1]
        resized_arr = cv2.resize(img_arr, (img_size, img_size))
        norm_arr = np.array(resized_arr) / 255
        return norm_arr.reshape(-1, img_size, img_size, 3)


    predict = model.predict([Preprocess("Temp/Test.png")])
    print(predict)
    predictVal = np.argmax(predict)
    os.remove("Temp/Test.png")
    os.remove("Temp/Test.csv")
```

```
    text = labelsOrd[predictVal]
    preVal = labelsOrd[predictVal]
    Soundplay = True

    if countDown == 0:
        Test = False

    mylabel.config(text=text)

    FileAdd = [currAng, preVal]
    with open("Tracking/BlindTest.csv", 'a+', newline='') as f_object:
        writer_object = writer(f_object)
        writer_object.writerow(FileAdd)
        f_object.close()
        FileAdd =[]
    mylabel.pack()
    ws.update()
    if currAng <180:
        angle = 360
    elif currAng ==180:
        angle = 170
        currAng = AngleSet(angle, MotorController, currAng)
        angle = 360
    else:
        angle = -360
    currAng = AngleSet(angle, MotorController, currAng)
    time.sleep(5)
    mylabel.config(text='...')
    mylabel.pack()
    ws.update()
```

* The Preprocessing Speech recognition is the same as the one in in Appendix C Section F.3.

Appendix H

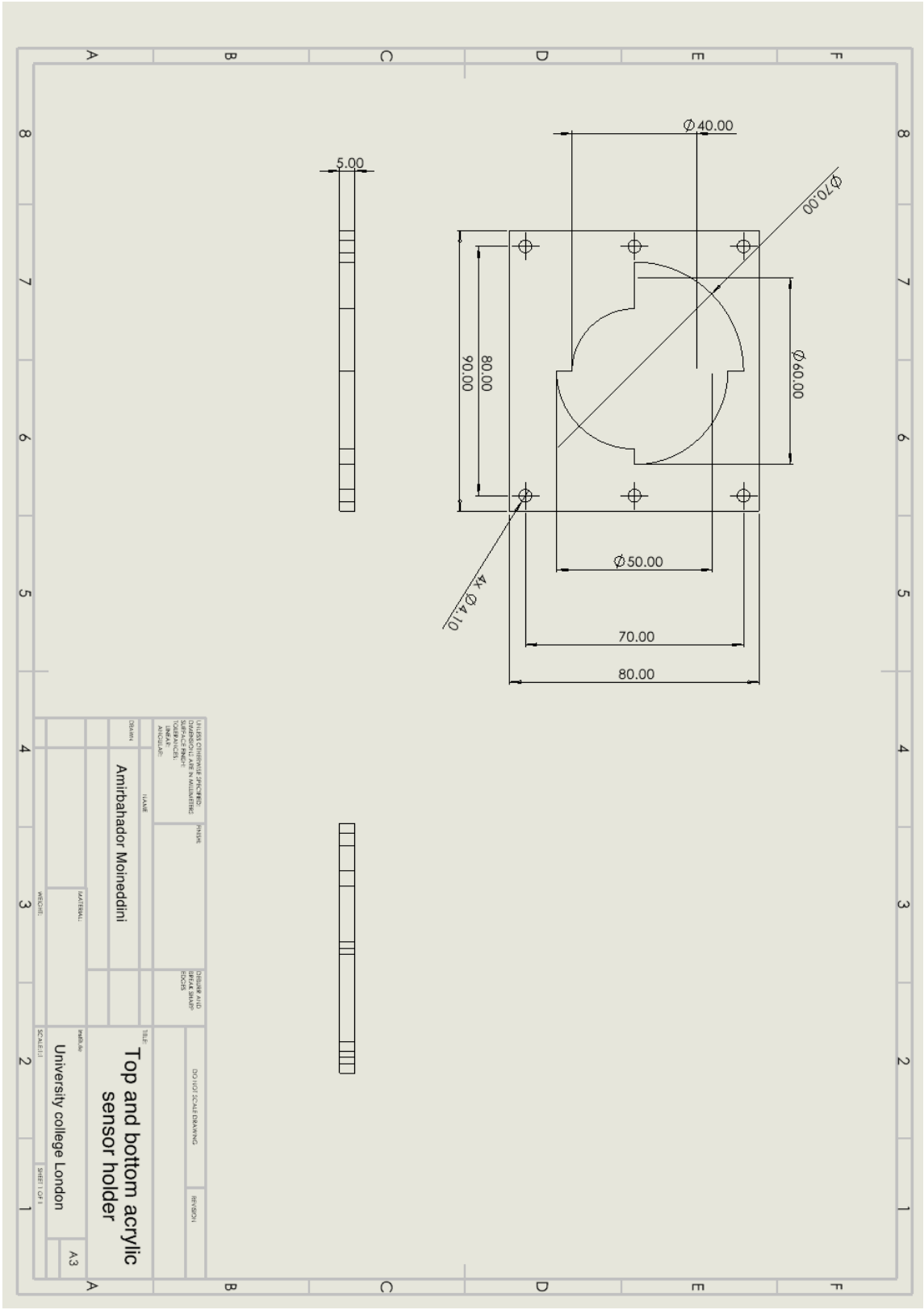
The device spatial testing rig design.



1) Top acrylic sensor holder
2) The sensor
3) Bottom acrylic sensor holder
4) Universal connector
5) Height adjustment Spacers
6) PLA 3D printed sensor holder
7) Top aluminium stage mount
8) Counterbalance mass mount arm
9) Mouth simulator mount arm
10) Bottom aluminium stage mount
11) Rotary stage
12) Acrylic offset mount
13) Anti vibration platform mount

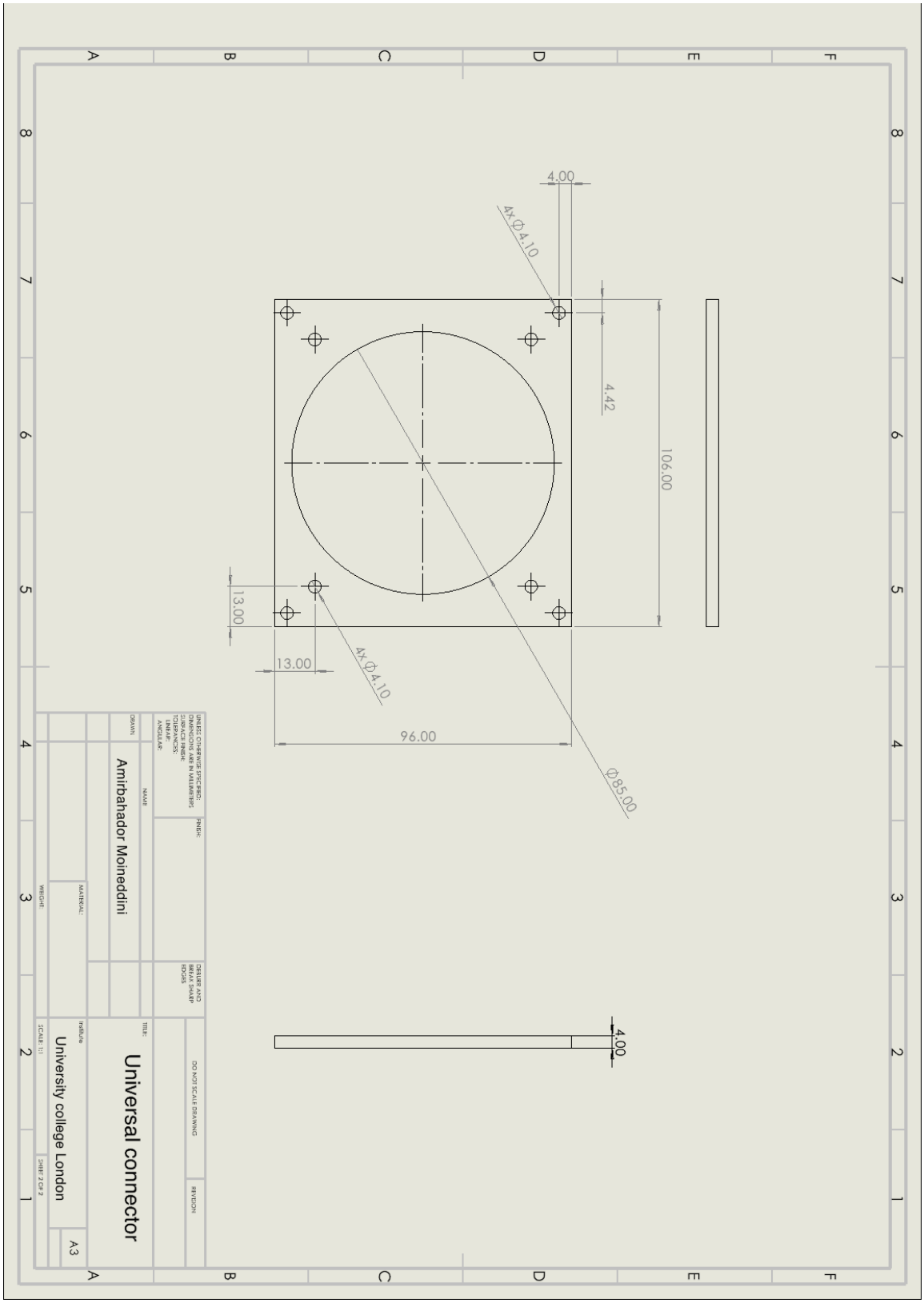
H.1. Top and bottom acrylic sensor holder:

Top and bottom acrylic sensor holder were used to sandwich the sensor and minimise the variation in movement of the sensor and clamping force between trials.



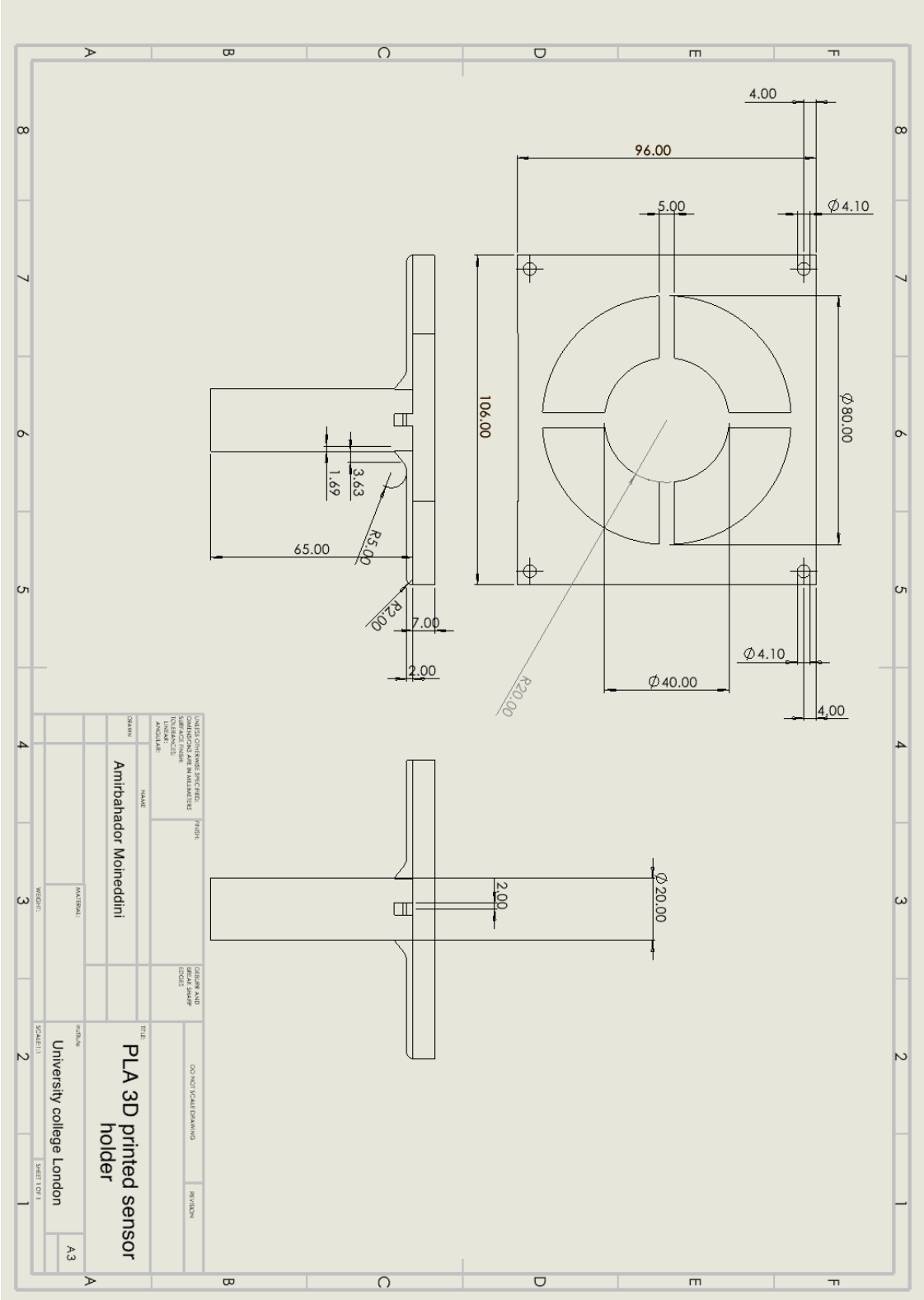
H.2. Universal connector:

To mount the sensor to the rest of the rig.



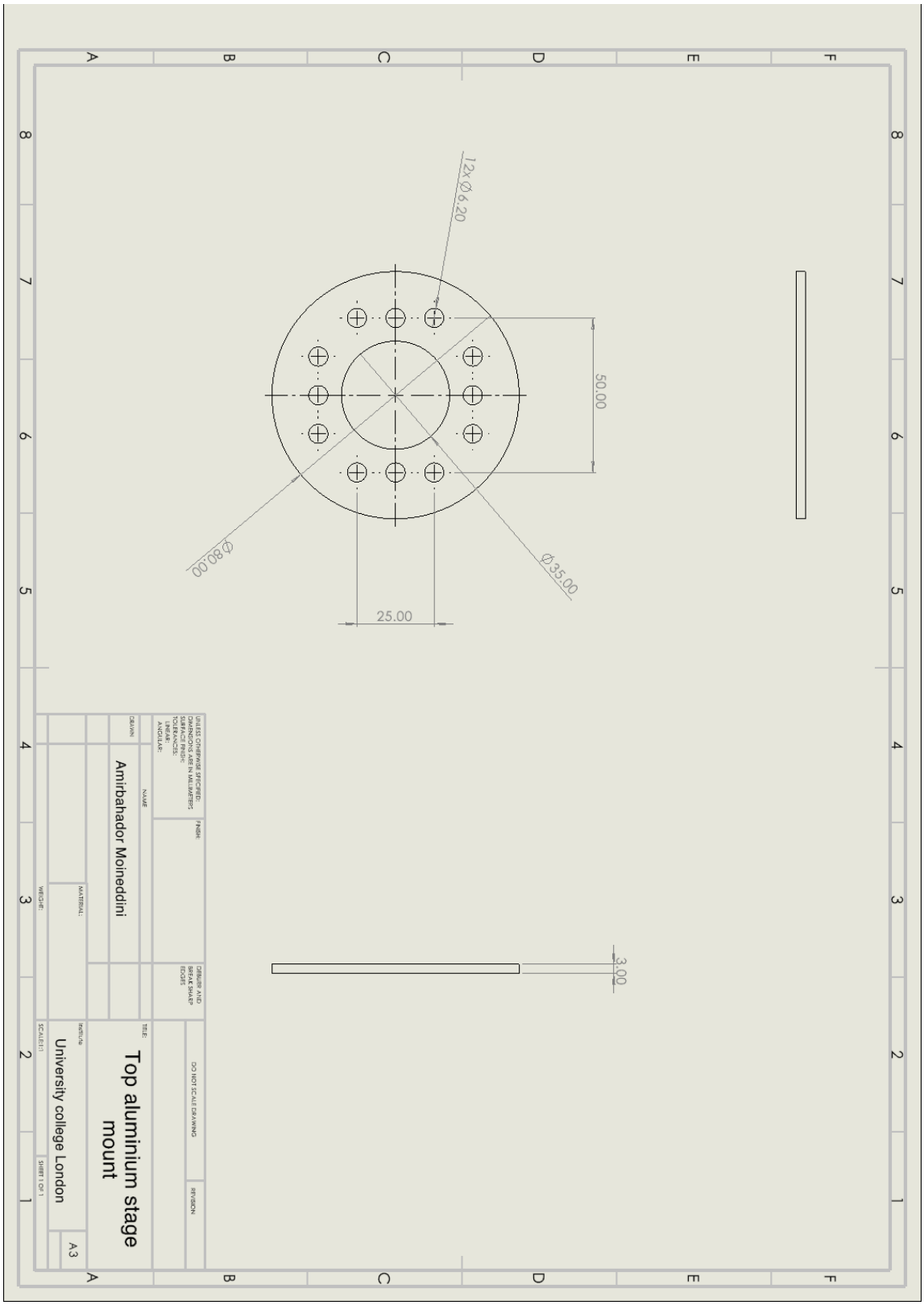
H.3. PLA 3D printed sensor holder:

To connect the sensor to the rest of the stage in the middle and allow for the speaker to move freely around the sensor.



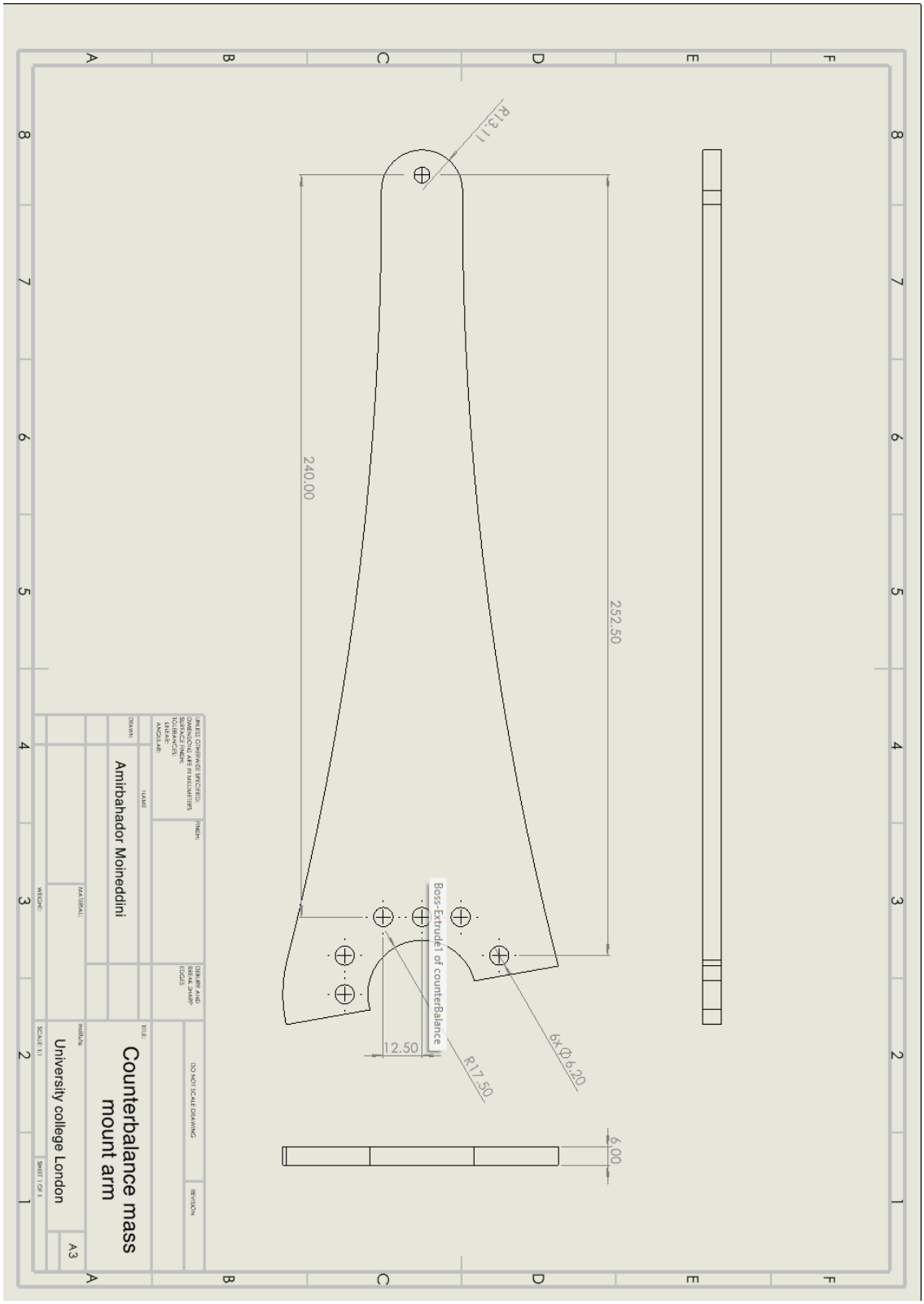
H.4. Top aluminium stage mount:

To spread the load of the weight of the mouth simulator and the counterbalance weight over the larger and reduce uneven loading on the stage.



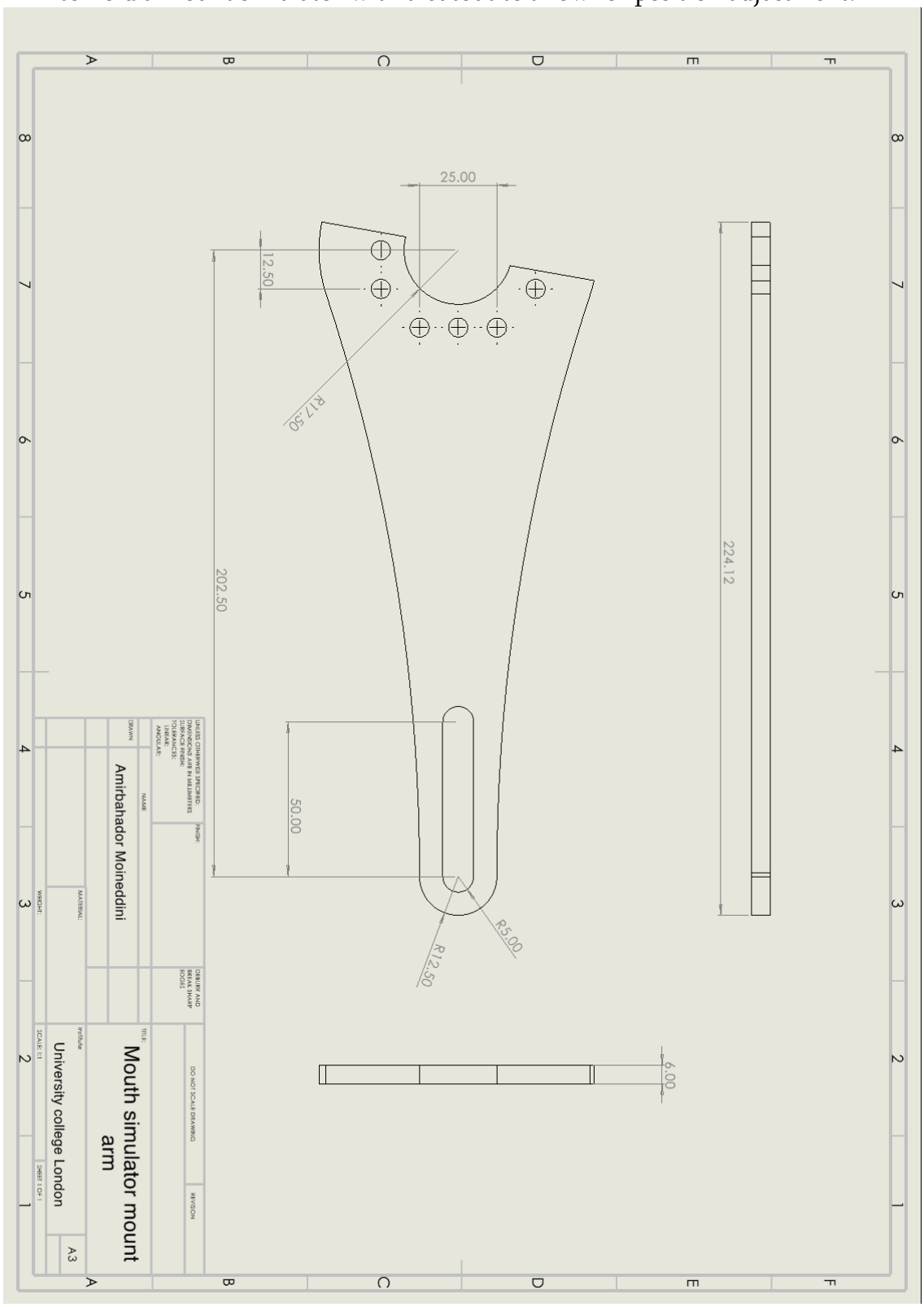
H.5. Counterbalance mass mount arm:

Aluminium arm to hold the counterbalance weight to the mount simulator.



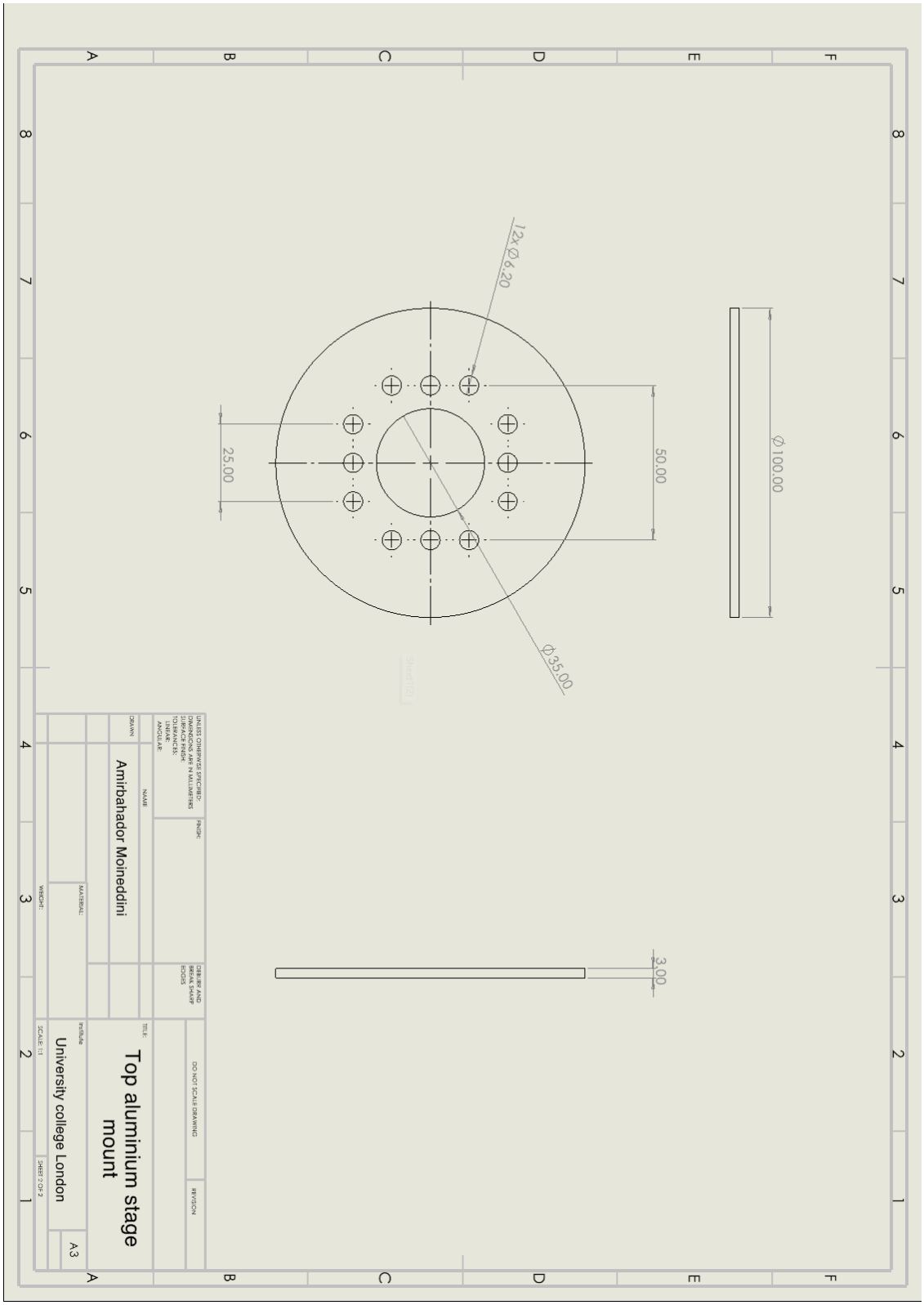
H.6. Mouth simulator mount arm:

Arm to hold a mount simulator with a cutout to allow for position adjustment.



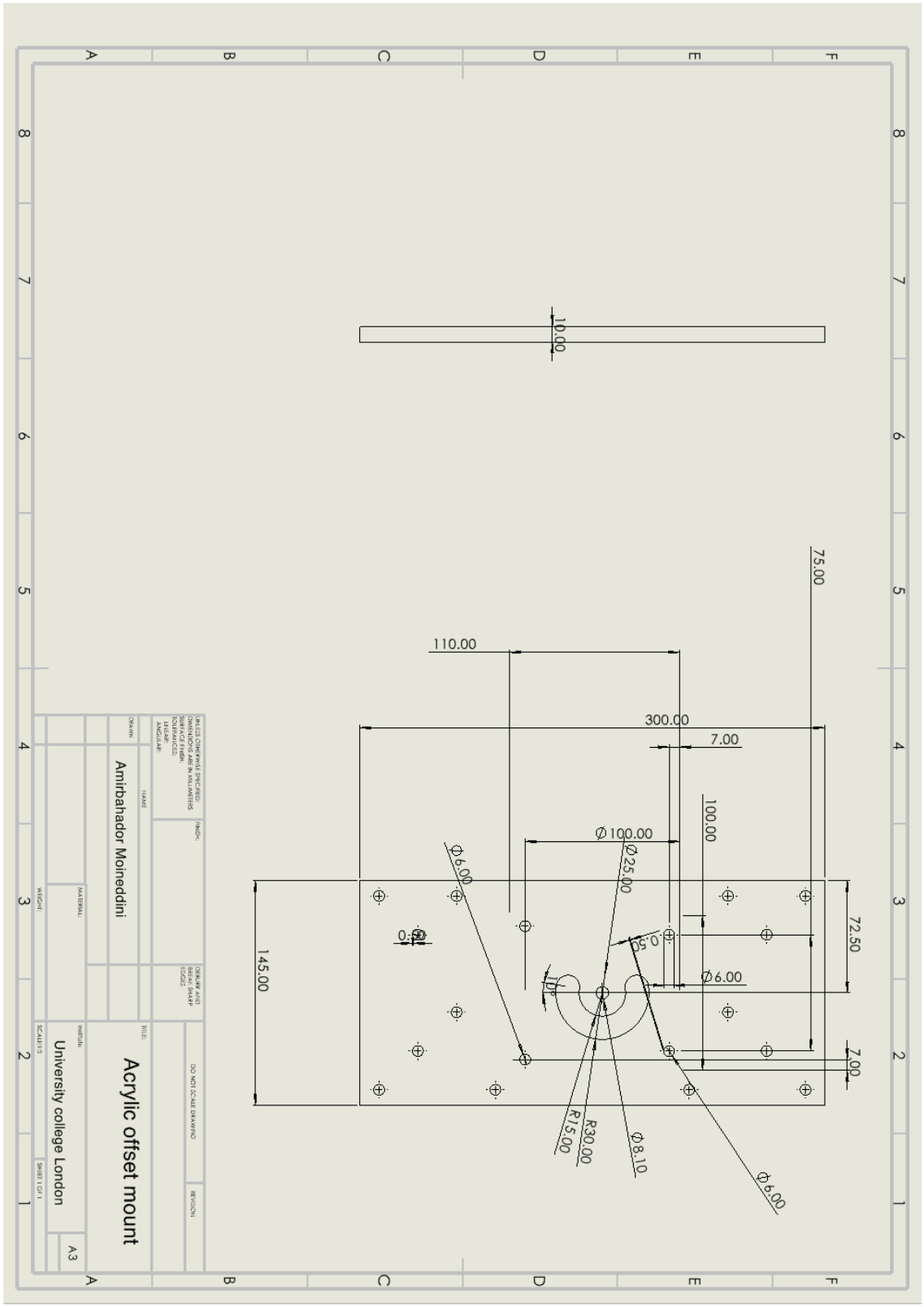
H.7. Bottom Aluminium stage mount:

To spread the load of the weight of the mouth simulator and the counterbalance weight over the larger and reduce uneven loading on the stage.



H.8.Acrylic offset mount:

This a intermediary offset mount that allow the rotary stage to connect to the antivibration platform and allow for the wiring to pass underneath the stage.



Appendix I

MATLAB code for Slicing STL file to get the waypoints.

Here is a link to the GitHub repository containing the MATLAB files and script along with a examples: abmoineddini/Electrohydrodynamics_slicer: Custom slicer for the Electrohydrodynamic printer (github.com)

This was a code is the modified version of the from a STL slicer by Sunil Bhandari. Here is the link to the original code: slice_stl_create_path(triangles,slice_height) - File Exchange - MATLAB Central (mathworks.com)

Appendix J

Arduino code for controlling the 3axis control EHDP printer.

GitHub repository: https://github.com/abmoineddini/EHD_printer/tree/main

```
/* Arduino Mega ElectroHydrodynamics Machine
 * by: Amirbahador Moineddini
 * date: August 08th, 2022
 * V2 version of the Arduino EHD Printer


/* Pin Setup
 * X-Axis stepper motor driver board:
 * - STEP  - pin 31
 * - DIR   - pin 33
 * - MS1   - pin 29
 * - MS2   - pin 27
 * - MS3   - pin 25
 * - En    - pin 23
 *
 * Y-Axis stepper motor driver board:
 * - STEP - pin 43
 * - DIR  - pin 45
 * - MS1  - pin 41
 * - MS2  - pin 39
 * - MS3  - pin 37
 * - EN   - pin 35


// MS1    MS2     MS3     Res
//  0      0       0       1
//  1      0       0      1/2
//  0      1       0      1/4
//  1      1       0      1/8
//  1      1       1      1/16
// Enable low => motors enabled
// xDir = HIGH => +X   xDir = LOW => -X
// yDir = HIGH => -Y   xDir = LOW => +Y
*/

// Declaring Controller Pins
const int xStep = 6;
const int xDir  = 7;
const int xMS3  = 3;
const int xMS2  = 4;
const int xMS1  = 5;

const int yStep = 11;
const int yDir  = 12;
const int yMS3  = 10;
const int yMS2  = 9;
const int yMS1  = 8;
```

```
const int enPin   = 2;

// Declaring microSwitches pins
const int xHome = 23;
const int yHome = 27;
int xVal;
int yVal;
// val = 1 => away from switch
// Val = 0 => Stage at home

// Counter and Homer
float xPos  = 0;
float yPos  = 0;
int Homer = 0;
int xStepSize = 0;
int yStepSize = 0;

int xStepsVal[]= {} // Array of x-direction waypoints obtained from the MATLAB
Code
int yStepsVal[]= {} // Array of y-direction waypoints obtained from the MATLAB
Code
int sz = 0;
int noXSteps = 0;
int noYSteps = 0;
int sumX = 0;
int sumY = 0;
int printStat = 0;

void hommingSequence(int xSpd, int ySpd);
void rotateMotors(int& noXSteps, int& noYSteps, int xSpd, int ySpd);
void StepSetx(int spd);
void StepSety(int spd);

void setup() {
  //Open serial communications
  Serial.begin(2000000);
  Serial.println("Starting Up....");

  //Define stepper pins as digital output pins
  pinMode(xStep,OUTPUT);
  pinMode(xDir,OUTPUT);
  pinMode(xMS1,OUTPUT);
  pinMode(xMS2,OUTPUT);
  pinMode(xMS3,OUTPUT);

  pinMode(yStep,OUTPUT);
  pinMode(yDir,OUTPUT);
  pinMode(yMS1,OUTPUT);
```

```
  pinMode(yMS2,OUTPUT);
  pinMode(yMS3,OUTPUT);

  pinMode(enPin,OUTPUT);

  // Setting input microswitches pins
  pinMode(xHome, INPUT);
  pinMode(yHome, INPUT);

  //Set microstepping mode for stepper driver boards.  Using 1.8 deg motor
angle (200 steps/rev) NEMA 17 motors (12V)

  //X-Axis motor: no micro stepping (MS1 Low, MS2 Low) = 1/16 deg/step (200
steps/rev)
  digitalWrite(xMS1,LOW);
  digitalWrite(xMS2,LOW);
  digitalWrite(xMS3,LOW);

  //Y-Axis motor: no micro stepping (MS1 Low, MS2 Low) = 1/16 deg/step (200
steps/rev)
  digitalWrite(yMS1,LOW);
  digitalWrite(yMS2,LOW);
  digitalWrite(yMS3,LOW);

  delay(5000);
}
void loop() {
  //Homing Sequence
  if (Homer == 0){
    StepSetx(1);
    StepSety(1);
    hommingSequence(700, 700);
    sz = sizeof(xStepsVal);
    Serial.print("Total number of steps = ");
    Serial.println(sz);
    for(int i =0; i<=sz/16; i++){
      sumX = sumX + xStepsVal[i];
      sumY = sumY + yStepsVal[i];
    }
    Serial.print("Total Travel = ");
    Serial.print(sumX);
    Serial.print(" y= ");
    Serial.println(sumY);
    if (sumX<0) //Set X-Axis rotation +X
    {
      digitalWrite(xDir,HIGH);
    }
    else{
```

```
    digitalWrite(xDir,LOW);
  }
  for (int i=0; i<=sumX; i++){
    digitalWrite(xStep,HIGH);
    delayMicroseconds(300);
    digitalWrite(xStep,LOW);
    delayMicroseconds(300);
  }
  if (sumY<0) //Set X-Axis rotation +X
  {
    digitalWrite(yDir,LOW);
  }
  else{
    digitalWrite(yDir,HIGH);
  }
  for (int i=0; i<=sumY; i++){
      digitalWrite(yStep,HIGH);
      delayMicroseconds(700);
      digitalWrite(yStep,LOW);
      delayMicroseconds(700);
  }
  Homer = 1;
  Serial.println("Calibration Done!");
}
else{
  if (printStat == 0){
    Serial.print(Homer);
    Serial.print("  ");
    Serial.print("x = ");
    Serial.print(xVal);
    Serial.print("  ");
    Serial.print("y = ");
    Serial.println(yVal);
    Serial.println("Platformed is homed");
    digitalWrite(enPin,LOW);

    for (int i=0; i<=sz/14; i++){
      StepSetx(1);
      StepSety(1);
      noXSteps = xStepsVal[i];
      noYSteps = yStepsVal[i];
      rotateMotors(noXSteps, noYSteps, 700, 700);
    }
    printStat = 1;
  }
  else {
    digitalWrite(enPin, HIGH);
    Serial.println("Print Done");
```

```
    }
  }
}
void hommingSequence(int xSpd, int ySpd){
  int xHomed = 0;
  int yHomed = 0;

  //Enable motor controllers
  digitalWrite(enPin, LOW);

  //Enable motor controllers
  digitalWrite(xDir,LOW);
  digitalWrite(yDir,LOW);
  while (xHomed == 0){
    xVal = digitalRead(xHome);

    if (xVal == 1){
      xHomed ++;
      digitalWrite(xDir,HIGH);
      for (int i=0; i<=5000; i++){
        digitalWrite(xStep,HIGH);
        delayMicroseconds(xSpd);
        digitalWrite(xStep,LOW);
        delayMicroseconds(xSpd);
      }
      xPos = 0;
      Serial.print("x-axis Homed");
    }
    else {
      digitalWrite(xStep,HIGH);
      delayMicroseconds(xSpd);
      digitalWrite(xStep,LOW);
      delayMicroseconds(xSpd);
    }
   }
   delay(1000);
  while (yHomed == 0){
    yVal = digitalRead(yHome);

    if (yVal == 1){
      yHomed ++;
      digitalWrite(yDir,HIGH);
      for (int i=0; i<=5000; i++){
        digitalWrite(yStep,HIGH);
        delayMicroseconds(ySpd);
        digitalWrite(yStep,LOW);
        delayMicroseconds(ySpd);
      }
```

```
      yPos = 0;
      Serial.print("y-axis Homed");
    }
    else {
      digitalWrite(yStep,HIGH);
      delayMicroseconds(ySpd);
      digitalWrite(yStep,LOW);
      delayMicroseconds(ySpd);
    }
   }
   Homer = 1;
   Serial.println("Platformed is Homed");
   delay(5000);
}

void StepSetx(int spd){
  switch (spd) {
  case 1:
    // Full step
    //X-Axis motor: no micro stepping (MS1 Low, MS2 Low) = 1/16 deg/step (200
steps/rev)
    digitalWrite(xMS1,LOW);
    digitalWrite(xMS2,LOW);
    digitalWrite(xMS3,LOW);
    break;
  case 2:
    // 1/2 step
    //X-Axis motor: no micro stepping (MS1 Low, MS2 Low) = 1/16 deg/step (200
steps/rev)
    digitalWrite(xMS1,HIGH);
    digitalWrite(xMS2,LOW);
    digitalWrite(xMS3,LOW);
    break;
  case 3:
    // 1/4 step
    //X-Axis motor: no micro stepping (MS1 Low, MS2 Low) = 1/16 deg/step (200
steps/rev)
    digitalWrite(xMS1,LOW);
    digitalWrite(xMS2,HIGH);
    digitalWrite(xMS3,LOW);
    break;
  case 4:
    // 1/8 step
    //X-Axis motor: no micro stepping (MS1 Low, MS2 Low) = 1/16 deg/step (200
steps/rev)
    digitalWrite(xMS1,HIGH);
    digitalWrite(xMS2,HIGH);
    digitalWrite(xMS3,LOW);
```

```
    break;
  case 5:
    // 1/2 step
    //X-Axis motor: no micro stepping (MS1 Low, MS2 Low) = 1/16 deg/step (200
steps/rev)
    digitalWrite(xMS1,HIGH);
    digitalWrite(xMS2,HIGH);
    digitalWrite(xMS3,HIGH);
    break;
    default:
    // 1/2 step
    //X-Axis motor: no micro stepping (MS1 Low, MS2 Low) = 1/16 deg/step (200
steps/rev)
    digitalWrite(xMS1,HIGH);
    digitalWrite(xMS2,HIGH);
    digitalWrite(xMS3,HIGH);
    break;
}
}

void StepSety(int spd){
  switch (spd) {
  case 1:
    // Full step
    //Y-Axis motor: no micro stepping (MS1 Low, MS2 Low) = 1/16 deg/step (200
steps/rev)
    digitalWrite(yMS1,LOW);
    digitalWrite(yMS2,LOW);
    digitalWrite(yMS3,LOW);
    break;
  case 2:
    // 1/2 step
    //Y-Axis motor: no micro stepping (MS1 Low, MS2 Low) = 1/16 deg/step (200
steps/rev)
    digitalWrite(yMS1,HIGH);
    digitalWrite(yMS2,LOW);
    digitalWrite(yMS3,LOW);
    break;
  case 3:
    // 1/4 step
    //Y-Axis motor: no micro stepping (MS1 Low, MS2 Low) = 1/16 deg/step (200
steps/rev)
    digitalWrite(yMS1,LOW);
    digitalWrite(yMS2,HIGH);
    digitalWrite(yMS3,LOW);
    break;
  case 4:
    // 1/8 step
```

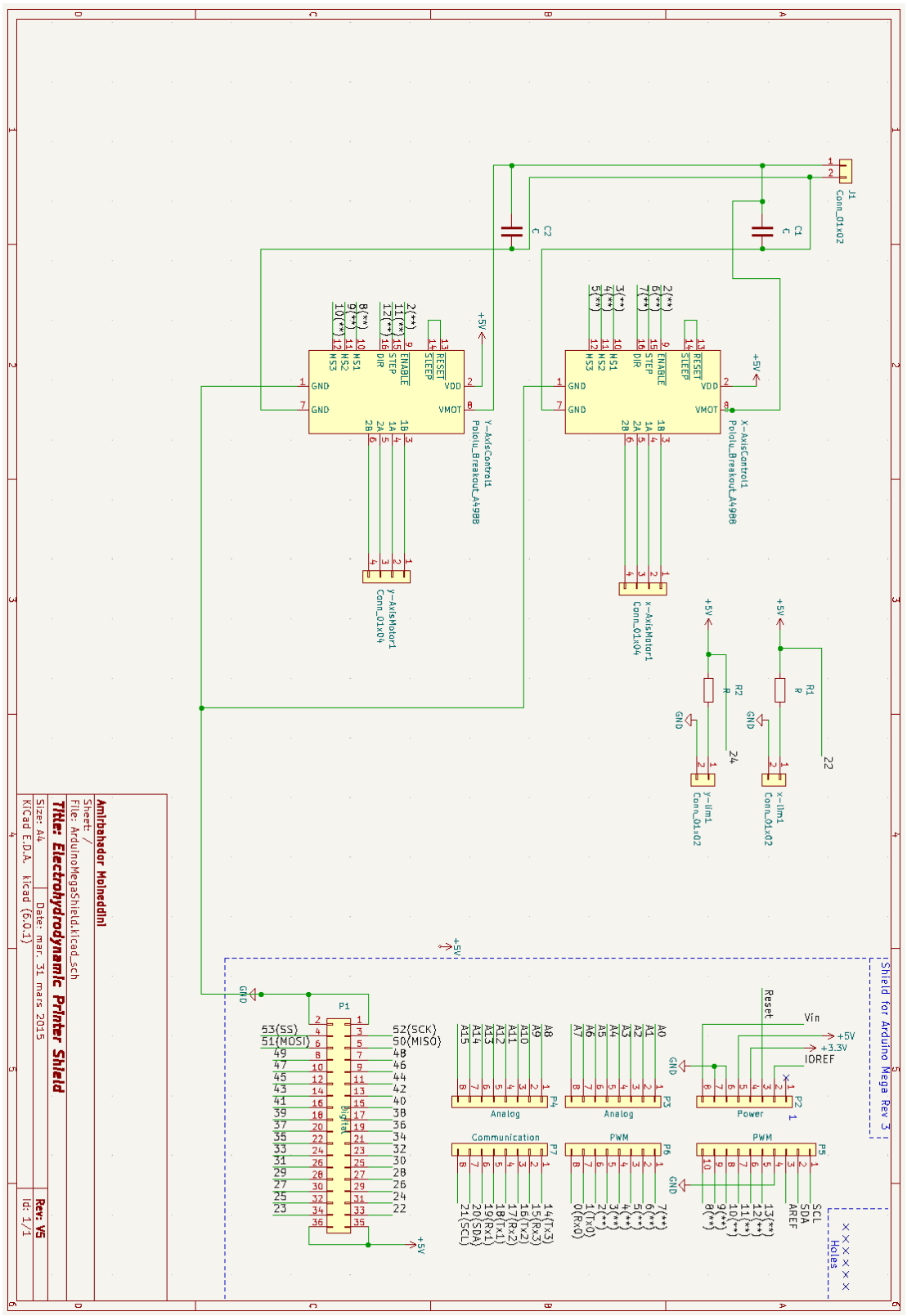
```
    //Y-Axis motor: no micro stepping (MS1 Low, MS2 Low) = 1/16 deg/step (200
steps/rev)
    digitalWrite(yMS1,HIGH);
    digitalWrite(yMS2,HIGH);
    digitalWrite(yMS3,LOW);
    break;
  case 5:
    // 1/2 step
    //Y-Axis motor: no micro stepping (MS1 Low, MS2 Low) = 1/16 deg/step (200
steps/rev)
    digitalWrite(yMS1,HIGH);
    digitalWrite(yMS2,HIGH);
    digitalWrite(yMS3,HIGH);
    break;
    default:
    // 1/2 step
    //Y-Axis motor: no micro stepping (MS1 Low, MS2 Low) = 1/16 deg/step (200
steps/rev)
    digitalWrite(yMS1,HIGH);
    digitalWrite(yMS2,HIGH);
    digitalWrite(yMS3,HIGH);
    break;
}
}
void rotateMotors(int& noXSteps, int& noYSteps, int xSpd, int ySpd){
  //Initialize while loop counter
  int totalSteps=0;
  int stepDelay=10;
  float xIncrement=0;
  float yIncrement=0;
  switch (xStepSize) {
  case 1:
    // Full Steps
    xIncrement = (0.5/200);
    break;
  case 2:
    // 1/2 Steps
    xIncrement = (0.5/200)/2;
    break;
  case 3:
    // 1/4 Steps
    xIncrement = (0.5/200)/4;
    break;
  case 4:
    // 1/8 Steps
    xIncrement = (0.5/200)/8;
    break;
  case 5:
```

```
    // 1/16 Steps
    xIncrement = (0.5/200)/16;
    break;
  default:
    // 1/16 Steps
    xIncrement = (0.5/200)/16;
    break;
  }
  switch (yStepSize) {
  case 1:
    // Full Steps
    yIncrement = (0.5/200);
    break;
  case 2:
    // 1/2 Steps
    yIncrement = (0.5/200)/2;
    break;
  case 3:
    // 1/4 Steps
    yIncrement = (0.5/200)/4;
    break;
  case 4:
    // 1/8 Steps
    yIncrement = (0.5/200)/8;
    break;
  case 5:
    // 1/16 Steps
    yIncrement = (0.5/200)/16;
    break;
  default:
    // 1/16 Steps
    yIncrement = (0.5/200)/16;
    break;
  }

  //Set X-Axis motor rotation direction based on read value
  if (noXSteps<0) //Set X-Axis rotation +X
  {
    digitalWrite(xDir,LOW);
  }
  else //Set X-Axis rotation -X
  {
    digitalWrite(xDir,HIGH);
    xIncrement = -xIncrement;
  }

  //Set Y-Axis motor rotation direction based on read value
  if (noYSteps<0) //Set Y-Axis rotation to CCW
```

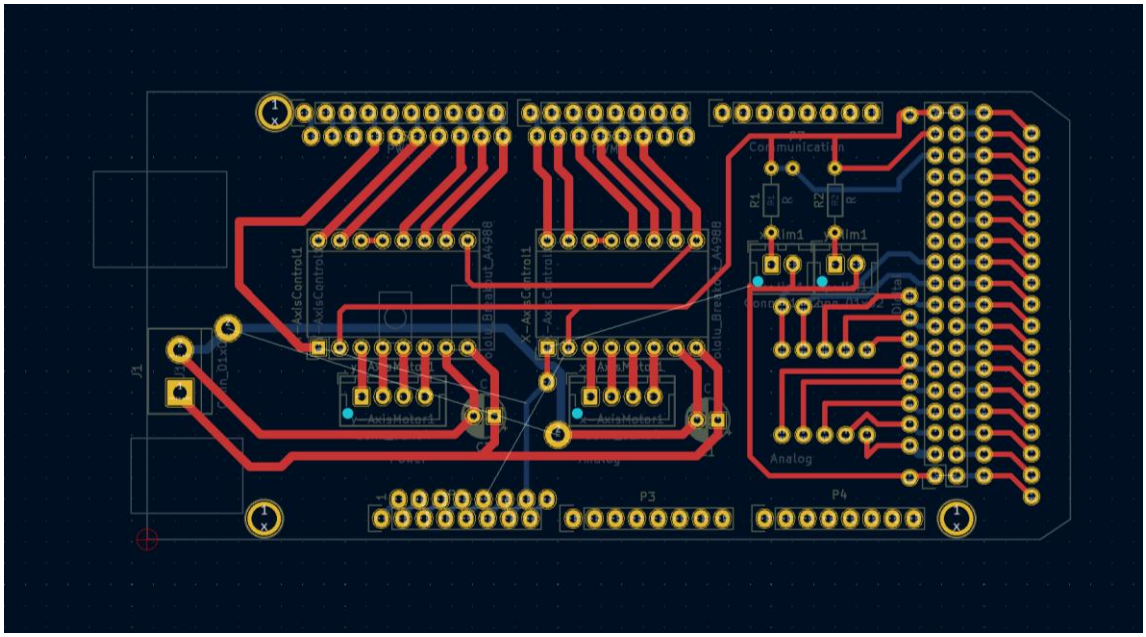
```
  {
    digitalWrite(yDir,HIGH);
    yIncrement = -yIncrement;
  }
  else
  {
    digitalWrite(yDir,LOW);
  }

  //Calculate total number of steps for while loop indexing
  totalSteps=(abs(noXSteps)+abs(noYSteps))/16;

  //Get absolute value of steps
  noXSteps=abs(noXSteps)/16;
  noYSteps=abs(noYSteps)/16;

  //Move motors appropriate number of steps
  while (totalSteps>0){
    if (noXSteps>0) //Move X-Axis
    {
      //Move X-Axis one step
      digitalWrite(xStep, LOW); //LOW to HIGH changes creates the "Rising
Edge" so that the EasyDriver knows when to step.
      delayMicroseconds(xSpd);
      digitalWrite(xStep, HIGH);
      delayMicroseconds(xSpd);
      noXSteps=noXSteps-1; //Decrement remaining number of X-Axis steps
      totalSteps=totalSteps-1; //Decrement remaining number of total steps
      xPos += xIncrement;
    }

    if (noYSteps>0) //Move Y-Axis
    {
      //Move Y-Axis one step
      digitalWrite(yStep, LOW); //LOW to HIGH changes creates the "Rising
Edge" so that the EasyDriver knows when to step.
      delayMicroseconds(ySpd);
      digitalWrite(yStep, HIGH);
      delayMicroseconds(ySpd);
      yPos += yIncrement;

      noYSteps=noYSteps-1; //Decrement remaining number of Y-Axis steps
      totalSteps=totalSteps-1; //Decrement remaining number of total steps
    }
  }
}
```

Appendix K

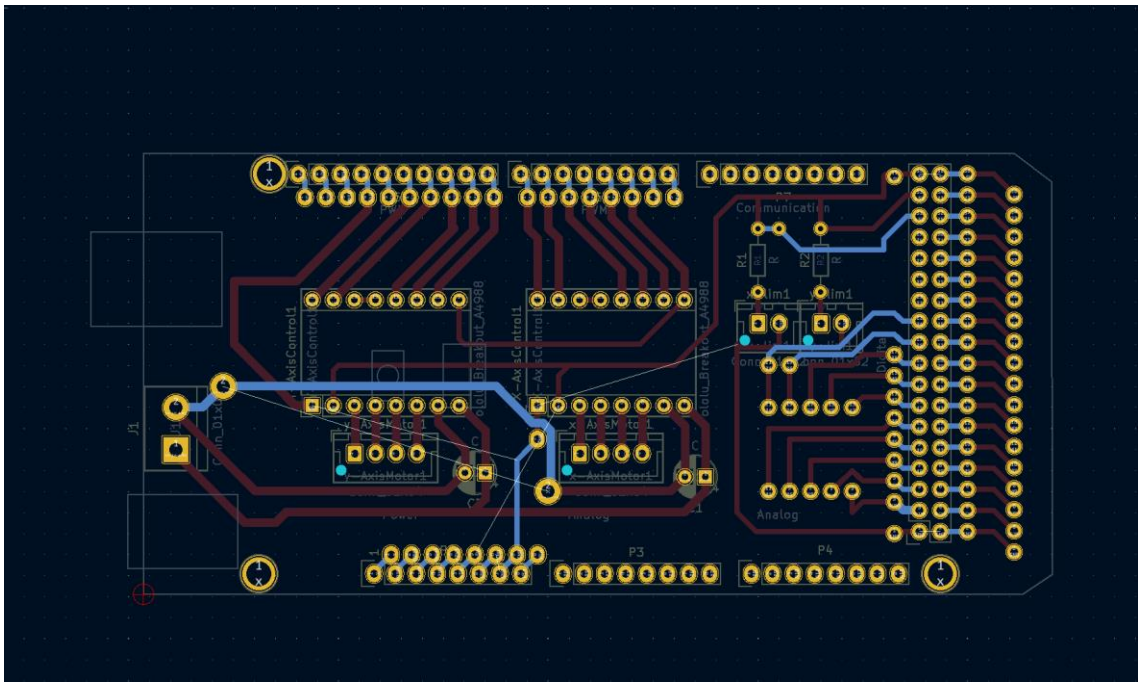
The circuit board diagram for the Arduino Mega Shield used to control the EHD printer. The Circuit was designed in KiCad.



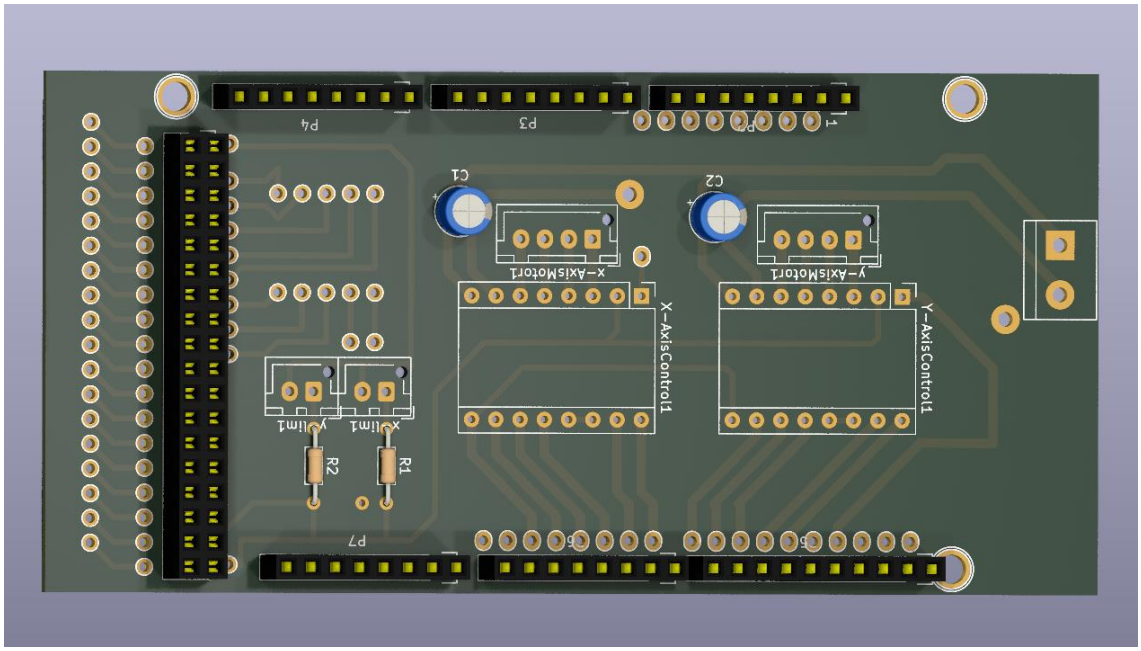
Circuit layout highlighting Top routs:



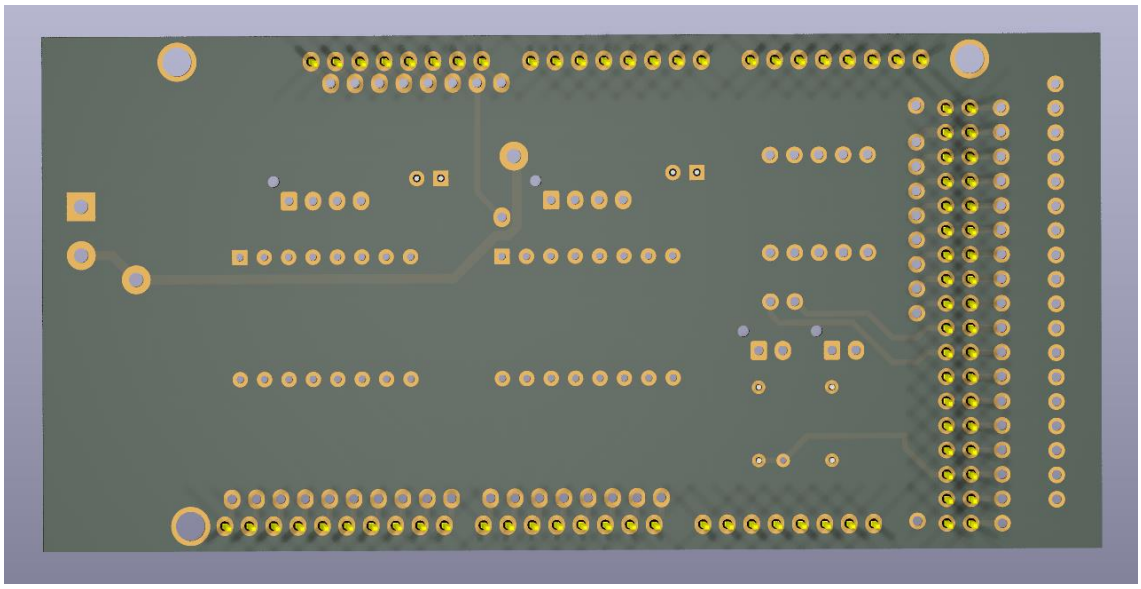
Circuit layout highlighting Bottom routs:



Top view:



Bottom View:



Isometric View of circuit board