\documentclass[a4paper]{article}

\usepackage{fourier}

\usepackage[margin=1in]{geometry}

\usepackage{float}

\usepackage{wrapfig}

\usepackage{subfig}

\usepackage{graphicx}

\usepackage{amssymb, amsmath}

\usepackage{lipsum}

\usepackage[]{mcode}

\usepackage[parfill]{parskip}

\usepackage{fancyhdr}

\usepackage[nottoc,numbib]{tocbibind}

\usepackage{tablefootnote}

\usepackage{etoolbox}% http://ctan.org/pkg/etoolbox

% HEAD & FOOT

\pagestyle{fancy}

\fancyhf{}

\rhead{Kartiksinh K. Gohil - CID: 00692607}

\lhead{Final Year Project - Final Report 2015}

\cfoot{\thepage}

\renewcommand

\lstlistingname{Code}

% REDUCE TITLE BLANK SPACE

\title{\vspace{-20pt}}

\author{\vspace{-20pt}}

\date{\vspace{-20pt}}

% START

\begin{document}

\maketitle

\input{frontpage.tex}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\tableofcontents

\newpage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\section{Introduction} \label{Introduction}

The aim of this project is to design a modern musical instrument that gives a user the freedom to use their environment as part of their sound production process. I will be designing and implementing an instrument that consists of a microphone on a wearable circuit board that, when tapped against any surface, will trigger a sample sound file selected by the user through a software interface. In other words, the user will potentially be able to play an entire drum kit simply by tapping a rhythm with their fingers. The instrument will be limited to this functionality for this project but has a large future scope, in that it could integrate motion gesture control to allow a user to, for example, generate synthesised sounds and modify the amplitudes and pitches in real-time using spatial positioning, or to generate the sounds of guitar chords when mimicking the action of playing one (the air guitar concept).

\begin{wrapfigure}{r}{0.35\textheight}

\centering

! MISSING IMAGE !

%\includegraphics[scale=0.2]{Images/paws\_final}

\caption{PAWS Prototype 0.2.04}

\label{fig:paws\_final}

\end{wrapfigure}

Section~\ref{Implementation} outlines the developments made during implementation. I started by using an Arduino Uno to interface my microphone with my computer's software application, which I initially began to code in Python. Through substantial testing and eventual failures, I switched to an audio library called JUCE in C++. I produced a user-friendly Graphical User Interface (GUI), with two functionalities: a 'Voice' function to feed the microphone signal through to the computer's audio device, and a 'Sample' function to trigger a user-selectable sample sound file upon tapping the microphone.

Throughout this project, I maintained the full scope of the product, detailed in Section~\ref{Product Concept} Product Concept, in mind. Despite focusing mostly on the 'Sample' function and ensuring that that was ready for a robust demonstration by the end of the project, consideration was always taken into the future development of the product, including the integration of accelerometers for motion control, despite not having had the chance to test them.

The most difficult part of this project was designing a fast and reliable method of data transmission, from sampling the microphone signal on the Arduino side, to transmitting it via USB to the computer, processing the signal accordingly, and throwing the relevant data to the computer's audio device. Dealing with what was effectively a multirate system, where the sample rates were variable at each stage of data capture or transmission, was challenging but developing a strong data structure for each of these components and optimising the speeds of data transmission and processing at each stage inevitably led to an overall successful system.

This report begins with some research into the current market for wearable technologies, especially in the field of music production. Section~\ref{Product Concept} Product Concept involves the design of an idea for a musical instrument to fit the original brief: \textit{'The development and evaluation of a novel and unusual musical instrument to be constructed using a 3D printer'}. Section~\ref{Requirements} analyses the requirements the designed product will have to meet, and the following sections detail the design of the product and initial testing of my sample triggering theories and key components such as an Arduino Uno microprocessor. Section~\ref{Implementation} Implementation provides a discussion on the development of my various prototypes from the design stage to a working demonstrable product. Subsequently, I evaluate the final prototype and the progress of my project and also detail further developments to be made if the project were to be continued. Section~\ref{Finances} Finances provides a summary of the costs incurred in developing each prototype and the projected costs of future designs, and Section~\ref{User Manual} provides a User Manual on using my final prototype. Appendices have also included at the end with supporting diagrams and schematics.

\newpage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\section{Market Research} \label{Background}

This final year project's original brief was \textit{'The development and evaluation of a novel and unusual musical instrument to be constructed using a 3D printer'}, which was fairly open-ended, relying more on a creative musically-oriented approach rather than the usual best-fit engineering solution.

The definition of a 'musical instrument' has evolved drastically over the years, ranging from traditional acoustic instruments (\textit{piano, violin}) to electrical (\textit{guitars, keyboards}), electronic (\textit{synthesisers, Theremin}), and even virtual instruments that exist purely as software models in computer-based audio production tools.

The current music market has been overtaken by new computer vision and wearable technologies, giving way to innovative products such as the \emph{Mi.Mu}\textsuperscript{\cite{mimu}} gloves and \emph{DrumPants}\textsuperscript{\cite{drumpants}}. The idea of capturing motion through worn sensors or distant cameras allows the user to integrate their bodies with electronic systems. Musicians can make sweeping gestures to produce and control sounds, much like was traditionally done with mechanical instruments. With the initial introduction of electronic instruments, musicians became confined to button-based keyboards and drum machines, and later to point-and-click software on computers.

\begin{wrapfigure}{r}{0.25\textwidth}

\centering

\includegraphics[scale=0.3]{Images/MiMuGloves}

\caption{Mi.Mu Gloves\textsuperscript{\cite{mimugloves}}}

\label{fig:mimugloves}

\end{wrapfigure}

New technologies have allowed the modern musician to let the motion of their body contribute to their overall sound but they still seem to be fairly restrictive. The Mi.Mu gloves, as shown in Figure~\ref{fig:mimugloves}, allow the user to generate music simply by moving their hands. They can emulate playing a drum kit with their hands and the gloves will send MIDI signals to a digital audio workstation (DAW), which will produce the correct sounds in response to the user's 'air-drumming'. The gloves can also be used to control functions on sound, such as altering amplitude, filtering and even adding effects like Reverb, all through a pre-determined hand motion. This technology, however, is incredibly expensive. Provisionally selling at nearly \pounds 5000\textsuperscript{\cite{mimugloves}}, these gloves are not accessible to the average person. It also requires other software tools for it to interact with, meaning that the user is in fact confined to a particular physical workspace, be it a studio or a live stage.

\begin{wrapfigure}{r}{0.35\textwidth}

\centering

\includegraphics[scale=0.2]{Images/drumpantsbasic}

\caption{DrumPants Basic Kit\textsuperscript{\cite{drumpantsprice}}}

\label{fig:drumpants}

\end{wrapfigure}

DrumPants is another product that utilises wearable sensors to control sounds. The instrument, shown in Figure~\ref{fig:drumpants}, uses sensor strips that attach to your clothes and connect wirelessly to a central controller. The strips contain pressure sensors that, when hit, send MIDI data to the controller, which can be set by the user to play any virtual sound in response, such as a drum kit or a piano. This controller can output audio directly to a speaker or can send the raw MIDI data to a compatible computer program. Currently on pre-order from \$129.99\textsuperscript{\cite{drumpantsprice}}, this instrument allows the user to program each sensor to play any pre-recorded sample sound, and is also being marketed to be able to remotely control video games. The functionality, however, is limited to contact via pressure, and slightly malforms the user's experience by again restricting them to a particular physical area at any given instance in time. Even though this physical workspace, or the area where the sensor is located, is movable between instances of use, it still limits the musician's ability to improvise and requires an inordinate amount of setup time before playing can commence.

\begin{wrapfigure}{r}{0.3\textwidth}

\centering

\includegraphics[scale=0.2]{Images/gesturering}

\caption{Ring by Logbar Inc. \textsuperscript{\cite{gesturering}}}

\label{fig:gesturering}

\end{wrapfigure}

A third product, not specifically designed for music but important nonetheless, is the \emph{Ring}\textsuperscript{\cite{gesturering}} by Logbar Inc. The Ring has the capability of controlling any web-linked interface with gestures through an Android or iOS App. It is designed for the user to assign gestures to specific features, from controlling music play-out on a smartphone to opening a set of shower curtains (provided they have internet connectivity). The idea behind the Ring is that it allows a user to directly interact with the world around them through a portable wearable sensor with apparently no physical limitations. The concept of this device, in its functionality and freedom of use, would greatly enhance a musician's creativity if applied to the world of sound.

Based on these new products, I aim to produce a musical instrument that heads in the direction of user-programmable electronics that is portable and not restricted to a physical workspace in the way that most other motion-based products (especially computer-vision related) are. The musical instrument should allow the user to define the manner in which they wish to produce sounds and should allow them to use the range of their entire bodies in the process without introducing any physical or mental limitations.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\section{Product Concept} \label{Product Concept}

\begin{wrapfigure}{r}{0.5\textwidth}

\centering

\includegraphics[scale=0.8]{Images/UserLevelDia}

\caption{User Level Design}

\label{fig:userlvl}

\end{wrapfigure}

The musical instrument I will be building will be based around the concept of programmable, wearable sensors. Firstly, the instrument shall use an array of sensors, such as audio (microphone) or motion (accelerometer/gyroscope), to capture and control sound. These sensors should be wearable by the user, most likely through bespoke 3D printed housings so as to meet that particular aspect of the project brief. The ability to wear these sensors in any location will increase the adaptability of their use, as well as removing the restrictions of a fixed physical environment. These sensors should also be programmable by the user in order to complete any given task. The user should be able to perform functions such as recording their voice, emulating a real live drum kit, mapping a tempo, and even harmonising their singing with virtual instruments, all by programming the sensors in various ways. Figure~\ref{fig:userlvl} shows a user-level diagram of a sensor array (PAWS Board) sending information wirelessly to an interface on either a computer or a smartphone and consequently producing audio.

\begin{wrapfigure}{r}{0.4\textwidth}

\centering

\includegraphics[scale=0.7]{Images/conceptphoneinterface3}

\caption{Concept Design for a smartphone-based Interface}

\label{fig:conceptphoneinterface}

\end{wrapfigure}

My instrument, entitled PAWS (Programmable And Wearable Sound), will in fact have both a hardware and software aspect to it. The hardware shall be a number of standardised sensor arrays that can be attached to any part of the body through custom 3D printed housings, which send their captured data to a software interface. This interface, be it on a laptop or on a smartphone, will allow the user to control the function of each PAWS Board and record the output audio to file, as shown in the concept design in Figure~\ref{fig:conceptphoneinterface}.

The focus of this product is not on building perfect sensor arrays with minimum latency, or on developing a new signal processing technology, but rather on conglomerating the various existing ideas on the market into a single instrument that can be used by musicians of all capabilities in any way they like to accomplish any given task. The key features of this instrument, therefore, should be flexibility and simplicity of use.

The current PAWS Board concept has three main functionalities. The first and simplest is to record vocals or any other sound that a PAWS Board may capture through its microphone. The interface should be able to obtain the input from the assigned Board and save it to a file or play it back for the musician to listen to through the interface's built-in audio output.

The second functionality the instrument should have is to be able to trigger sample sounds based on percussive motion. Figure~\ref{fig:conceptsketchright} shows a finger with a PAWS Board attached tapping a rhythm on a random surface. If the Board has been programmed to trigger a sample such as a drum sound (as demonstrated in Figure~\ref{fig:conceptphoneinterface}), the user should be able to play a virtual drum kit through any given surface. Figure~\ref{fig:conceptsketchleft} shows a hand with another PAWS Board controlling select parameters of the output audio through motion gestures. This would be achieved by programming a range of motion to adjust a specific parameter such as Amplitude or Pitch.

Further designs for the PAWS Board include allowing a musician to set a particular tempo by tapping their feet (with a PAWS Board attached), which could also aim to quantise any other sounds that they produce, thus improving the playback quality. The sample trigger function could also be used to let a musician harmonise with themselves. For example, if they were singing into one PAWS Board and tapped another onto a surface, as if playing a piano, the instrument would trigger a piano sample at the same pitch as their vocal melody. Multiple PAWS Boards for sample triggering could be used to allow a musician to play entire chords in harmony with their voice.

\begin{figure}[H]

\centering

\subfloat[PAWS Board: Motion Function]{

\includegraphics[scale=0.7]{Images/conceptsketchesleft}

\label{fig:conceptsketchleft}

}

\subfloat[PAWS Board: Sample Function]{

\includegraphics[scale=0.7]{Images/conceptsketchesright}

\label{fig:conceptsketchright}

}

\caption{Concept Sketch of Usage}

\label{fig:conceptsketch}

\end{figure}

\newpage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\section{Concept Requirements} \label{Requirements}

This section explores what will be required of the implemented product and details the necessary specifications to evaluate each prototype with. The plan to evaluate the success of the musical instrument, and therefore of the project, mainly revolves around its ability to produce sound. There are certain specifications highlighted in Table~\ref{tab:specifications} that must be met in order to enable my musical instrument to contend with the currently developing technologies.

\begin{table}[H]

\centering

\begin{tabular}{|l l}

Flexibility Of Use & What can the user do with the instrument?\\

Simplicity Of Use & How easily can the user program PAWS Boards? \\

Functionality & What can the instrument do?\\

Portability & How easy is the instrument to carry?\\

Setup Time & How long does it take to ready the instrument?\\

Performance & Does it do what it has been programmed to do?

\end{tabular}

\caption{Evaluation Criteria of Product}

\label{tab:criteria}

\end{table}

The Evaluation Criteria are open-ended questions that can be answered qualitatively to enable a discussion on the objectives a particular prototype has met and how it could be further improved upon. A more detailed set of questions is shown in Table~\ref{tab:criteria} to highlight whether specific functionalities of the concept design have been implemented or not. These questions help to keep in mind the future scope, and hence the overall direction, of the project such that the implementation section does not get stuck on any one aspect of the concept design.

\begin{table}[H]

\centering

\begin{tabular}{|l}

\textit{The PAWS Board}\\ \hline \hline

Can it capture audio? \\

Can it modify the amplitude of the audio signal?\\

Can it modify the frequency content (filter) the audio signal?\\

Can it capture motion? \\

Can it connect to the Interface? \\

Can it send captured data to the Interface via wired transmission?\\

Can it send captured data to the Interface via wireless transmission?\\

Is it wearable through 3D printed attachments? \\

Is it comfortable to wear?\\

Can it be switched on and off?\\

Can it remind the user of its functionality?\\

Can it be carried in a pocket or bag? \\

\hline \hline

\textit{The Interface} \\ \hline \hline

Is the Interface available on a computer? \\

Is the Interface available on a smartphone? \\

Can it connect to PAWS Boards? \\

Can it read data sent by the connected PAWS Boards?\\

Can the user control the function of every connected PAWS Board? \\

Can it modify the amplitude of the audio signal from each PAWS Board?\\

Can it modify the frequency content (filter) the audio signal from each PAWS Board?\\

Can it map motion gestures to control parameters?\\

Can it open and play sample sound files?\\

Can it trigger sample sound files from PAWS Board data?\\

Can it output the generated audio to the output audio device? \\

Can it save the output audio to file?\\

\end{tabular}

\caption{List of Specifications}

\label{tab:specifications}

\end{table}

\newpage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\section{Concept Design} \label{Concept Design}

This section outlines the design of the concept musical instrument. Section~\ref{System Design} tackles the design of the overall system including its various conceptualised functionalities, while Section~\ref{Implementation Design} talks about the methods in which the system may be implemented.

\subsection{System Design}\label{System Design}

Figure~\ref{fig:syslvlhi} shows the high-level system design of the instrument. The instrument consists of multiple hardware PAWS Boards, each with the ability to capture sound and motion data, connected to a central interface which processes the data and produces an audio output.

\begin{wrapfigure}{r}{0.2\textheight}

\centering

\includegraphics[scale=0.8]{Images/SystemLevelHi}

\caption{System Design - High Level}

\label{fig:syslvlhi}

\end{wrapfigure}

Figure~\ref{fig:syslvllo} gives a more detailed view of the processes involved inside the PAWS Boards and the Interface. The Interface has the ability to receive data wirelessly from any number of PAWS Boards and the function of each can be selected by the user.

The Voice function simply routes the captured audio to the output. The Sample function allows the user to select a saved sample file and then processes the input data in order to trigger this sample. The Motion function allows the user to calibrate a particular gesture (such as sweeping a PAWS Board horizontally through the air) and assign it to change a particular modification parameter. For example, the selected parameter could be the global amplitude of the output audio, and the motion gesture would provide a continuous scale to change the amplitude level.

In this initial system design, the Motion function serves to control particular parameters of the global audio output rather than a sound synthesis tool in itself. The Interface also allows the user to save the produced audio to a file.

%\newpage

\begin{figure}[H]

\centering

\includegraphics[scale=1.2]{Images/SystemLevelLo}

\caption{System Design - Low Level}

\label{fig:syslvllo}

\end{figure}

\subsection{Implementation Plan}\label{Implementation Design}

The PAWS Board hardware needs to be light, portable and easily attachable to 3D printed wearable housings. This concept requires the production of printed circuit boards (PCBs) with components small enough to allow each Board to be placed anywhere on the body. During the initial production stages, various components and circuits will be tested on breadboard for functionality before being printed on a PCB. Initial implemented prototypes will also feature much bigger components that are easier to test designs with, and therefore will not meet some of the required specifications of the overall product until much later in the production stage.

The Interface will initially be a computer application (specifically built for Mac OS X) as a starting point. It is required to provide a graphical user interface (GUI) to allow the user to control the signal processing chain. The easiest way to implement this program would be using the \emph{Python} programming language. Python is an incredibly high-level language that is simple to write and can be used with a variety of signal processing and user-interface libraries. Libraries such as \emph{Pydub}\textsuperscript{\cite{pydub}} and \emph{PyAudio}\textsuperscript{\cite{pyaudio}} were found to integrate the ability to manipulate audio with the Python programming language but did not specifically meet the requirements of the Interface software. The \emph{PyO}\textsuperscript{\cite{pyolibrary}} library is specifically designed for digital signal processing (DSP) in Python and will allow the Interface to be able to process audio signals in order to complete the user's tasks. The article \emph{Python For Audio Signal Processing}\textsuperscript{\cite{pyasparticle}}, Glover et al., demonstrates the use of other libraries for manipulating audio using Python, which may be useful when attempting to find the best solution to programming the Interface. Other languages such as \emph{Objective-C} may also be explored for designing the GUI, or \emph{C} for a much lower-level (and therefore more efficient) implementation of signal manipulation techniques.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\begin{wrapfigure}{r}{0.3\textwidth}

\centering

\subfloat[Arduino Uno Development Board\textsuperscript{\cite{arduinopic}}]{

\includegraphics[scale=0.06]{Images/ArduinoUno}

\label{fig:arduinouno}

}\\

\subfloat[HC-06 Bluetooth Module for Arduino\textsuperscript{\cite{hc06ebay}}]{

\includegraphics[scale=0.05]{Images/hc06pic}

\label{fig:hc06}

}

\caption{Development Kit used for Prototyping}

\label{fig:btarduino}

\end{wrapfigure}

\section{Initial Testing of Concept Design} \label{Proof of Concept}

Before starting on the implementation of the product, I tested several necessary elements such as using a microcontroller to read audio and output it to a connected terminal and I simulated the concept of sample triggering using \emph{Matlab}.

\subsection{Data Input and Output - Microcontroller}

I tested the streaming of data in real-time by connecting a HC-06 (JY-MCU) Bluetooth Module to an Arduino Uno, both of which are shown in Figure~\ref{fig:btarduino}. The Arduino Uno is a development board aimed at hobbyists built around an Atmel ATMEGA328P-PU microcontroller. The microcontroller can be programmed through Arduino's bespoke integrated development environment (IDE) that can be downloaded for free from Arduino's website\textsuperscript{\cite{arduinosite}} alongside documentation for the Uno board itself.

The HC-06 Bluetooth Module can easily be attached to the Serial outputs of the microcontroller through a simple circuit, shown in Appendix~\ref{arduinobluetooth}.

The ATMEGA microcontroller can be programmed through Arduino's IDE to read analogue signals with its built in analogue-to-digital-converter (ADC) and output the read data to the HC-06 module via the Serial bus. In this sense, the microcontroller acts as a central digital processor that connects the output of the microphone circuit to the bluetooth module to be wirelessly transmitted to the instrument Interface.

I used a smartphone-based bluetooth terminal app to connect with the HC-06 module and ensure that it transmitted all the data sent to it by the microcontroller correctly.

\subsection{Data Input and Output - Interface}

Certain elements of the Interface design were first tested using Matlab before implementing them in Python. I wrote Matlab scripts that would connect to the Arduino's Serial Bus (via USB) and read the incoming data from the Arduino Uno. I used timed interrupts (in the way that the final implemented Interface would) to output the read data at a fixed sample rate.

After simulating the input and output of data on the Interface side, I moved onto implementing the software application using Python through an integrated development environment (IDE) called \textit{XCode}.

\subsection{Sample Triggering}

Figure~\ref{fig:samptrigmatlab} shows the various processing stages of a simulation of triggering a pre-recorded drum sample from recorded audio.

The raw recording was of a rhythm being tapped into the microphone of my smartphone. The recording then underwent very basic filtering and transient detection processes to produce spikes that were used to finally trigger the chosen drum sample. As we can see, the \textit{microphone taps} were recognised correctly as triggers by the system and therefore produced a drum beat of the same rhythm, although slightly latent than the original recording. In this instance, the raw recording was very clean, which may not be the case with the PAWS Board, and was processed offline whereas the PAWS Board's audio stream will be processed in real-time and thus will be prone to more computational complications.

\begin{figure}[H]

\centering

\includegraphics[width = 0.6\textheight, height = 0.4\textheight]{Images/slaptodrum}

\caption{Simulation of Sample Triggering}

\label{fig:samptrigmatlab}

\end{figure}

\newpage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\section{Implementation} \label{Implementation}

This section entails the implementation of the PAWS Board and counterpart Interface as a series of prototypes, with each version building upon the previous in terms of system design or aesthetics.

\subsection{Prototype 0.1.01} \label{0101}

\begin{wrapfigure}{r}{0.35\textheight}

\centering

\includegraphics[scale = 0.045]{Images/prototype101board}

\caption{Photograph of Prototype 0.1.01 setup}

\label{fig:101board}

\end{wrapfigure}

Prototype 0.1.01 is the very first realisation of the PAWS musical instrument. Figure~\ref{fig:101board} shows the microphone circuit implemented on a breadboard and connected to the Arduino Uno development board. The breadboard also includes connections for a bluetooth module and a programmable gain amplifier (PGA) wired for testing but these have not been utilised for the current prototype. The circuit is comprised of a microphone connected to the Atmel ATMEGA328 microcontroller through a gain stage. The microcontroller is programmed to stream the audio data to the Serial bus, which is currently connected to the Interface on my laptop via a USB cable. The Interface was programmed in Python to read data from the Serial bus and output it to the laptop's core audio device. A library called \emph{PyO}\textsuperscript{\cite{pyolibrary}}, written for implementing digital signal processing (DSP) functions, was used to generate two python scripts: one to play whatever audio the microphone captured, and another to trigger drum samples in response to the microphone being tapped with a finger.

This prototype is able to play the audio captured by the microphone, although it is fairly distorted. The prototype is also able to trigger drum sounds when the microphone is slapped. However, the interface is not yet programmed to filter the raw input and detect the transients in a clean manner and therefore the drum sounds have a slight latency and sometimes misfire.

\begin{wrapfigure}{r}{0.25\textheight}

\centering

\includegraphics[scale=1]{Images/mic0101}

\caption{Prototype 0.1.01 - PAWS Board Circuit}

\label{fig:101circuit}

\end{wrapfigure}

\subsubsection{Microphone Circuit}

Figure~\ref{fig:101circuit} shows the microphone circuit used in Prototype 0.1.01. The first stage, connected as described in the datasheet\textsuperscript{\cite{kingstatedatasheet}}, is fed into a TL072 op-amp with a gain of ten thousand to boost the audio signal to within the dynamic range of the microcontroller's ADC. This prototype uses the 5V power rails provided by the Arduino and is therefore laced with digital noise. Thus, a 100{$\mu$}F electrolytic capacitor was placed across the rails in an attempt to reduce this noise.

\subsubsection{Microcontroller Program}

The microcontroller program uses a timed interrupt to read a value from its input analogue pin at a set frequency of 8kHz and this value is added to a global circular buffer. In the main program loop, the values in this circular buffer are printed to the Serial bus. The print to Serial cannot be handled inside the interrupt function as it takes much longer to process than the frequency of the function call. Therefore, the print to Serial process is handled in the main loop, however it can lead to inaccuracies due to the two functions writing to and reading from the circular buffer at different frequencies. A dynamically allocated queue would be preferable but the Arduino libraries \emph{QueueList}\textsuperscript{\cite{queuelist}} and \emph{QueueArray}\textsuperscript{\cite{queuearray}} seemed to be unable to cope with the speed at which the data was being read. Appendix~\ref{mcflow} shows a flowchart diagram of the

microcontroller program.

\subsubsection{Interface - Input and Output}

Programming the Interface to read from the Serial bus is well-documented in the \emph{pySerial}\textsuperscript{\cite{pyserial}} library and was therefore very straightforward to implement. Each read sample was appended to a Python list variable, used as a buffer queue, as a \emph{PyO} signal, ready for processing.

The first version of the Interface was developed to simply take the input signal buffer and output it to the core audio device, and thereafter clear the samples that had been played from the buffer. The \emph{PyO} library uses an audio server that receives data samples asynchronously and outputs them at the user-defined sampling rate. Therefore, it was simple enough to program an infinite loop to constantly read in samples from the Serial bus and write them to the output server. Appendix~\ref{interfaceflow} shows a flowchart describing the processes involved in this Interface program.

\subsubsection{Interface - Sample Triggering}

Another version of the Interface was created where data samples were read in chunks from the Serial bus, compared against an intuitively-set threshold value and the generated on/off pulses were used to send the waveform of a drum sound to the output audio server. This method was done in the most basic manner in order to quickly demonstrate that it could indeed be done. The prototype was able to play drum sounds when the microphone was struck but there seemed to be a small delay between the two events. There were also many misfires, or drum sounds being played when they were not supposed to, due to the quickly implemented spike detection method. Appendix~\ref{interfaceflow} shows how this particular prototype implements the sample triggering function.

\subsection{Prototype 0.1.02} \label{0102}

Prototype 0.1.02 involved the creation of a Graphical User Interface (GUI) to control the Direct Read and Sample Triggering functions previously coded. These were respectively labelled as 'Voice' and 'Sample' functions on the interface.

\begin{wrapfigure}{r}{0.30\textheight}

\centering

\includegraphics[scale=0.5]{Images/pyblue\_play}

\caption{Prototype 0.1.02 - wxPython GUI}

\label{fig:pyblue\_play}

\end{wrapfigure}

\subsubsection{Python Program}

The GUI was built using a library called \textit{wxPython}\textsuperscript{\cite{wxpython}}, and as seen in Figure~\ref{fig:pyblue\_play} features top level buttons to 'Add' and 'Remove' boards. Each addition generates a drop-down list of serial ports that can be connected to, and once connected to the Arduino, radio buttons for selecting the required function are shown.

The program to work with the GUI was trickier to implement. Figure~\ref{fig:pyblue\_flow} shows a flow chart to demonstrate the workings of the code. Upon startup, the audio server and GUI are set up. The 'Add PAWS Board' button creates a \textit{Bus} which is in control of storing information on the selected serial port and function flags to determine whether the program should be in 'Voice' mode or 'Sample' mode. When a serial port is chosen from the drop-down list on the GUI, a function is called to open the port and ready it for data transfer. The function-selection radio buttons call the respective 'Voice' or 'Sample' functions in a new processing thread to simultaneously process the incoming audio data while continuously listening for events on the GUI. The 'Voice' and 'Sample' functions run endlessly based on their respective flags. Only one flag can be active at any time and the dropping of a flag will cause the respective function to exit. Upon quitting the program, all Busses are disconnected and closed and all flags are dropped to allow all running threads to finish correctly.

\begin{figure}

\centering

! MISSING IMAGE !

%\includegraphics{}

\caption{Flow Diagram for Python Program, Prototype 0.1.02}

\label{fig:pyblue\_flow}

\end{figure}

The GUI itself uses Sizers, a \textit{wxPython} functionality, that allows for elements to be dynamically grouped together. This meant that I was able to simply display the 'Add PAWS Board' button upon startup, which when pressed would create a 'Select Serial Port' drop down list on the GUI and a corresponding Bus in the back end. A function to read all available serial ports was also called to populate the drop down list on the GUI. Adding more boards would result in the creation of more Busses and Serial Port lists for the user to interact with. If a valid Serial Port was selected, the radio buttons for selection the required function would then display. Also, pressing the 'Remove PAWS Board' button would delete the GUI elements related to the last open Bus and would also call a function to reset the data stored in that Bus.

\subsection{Prototype 0.1.03} \label{0103}

This prototype made significant changes to the hardware. A Programmable Gain Amplifier (PGA) was bought and tested against the previously implemented TL072 operational amplifier for the microphone circuit. The Arduino's functionality was also optimised to remove some of the previously discussed redundancies.

\subsubsection{Improving the Operational Amplifier Circuit}

\begin{wrapfigure}{r}{0.30\textheight}

\centering

\includegraphics[scale=0.4]{Images/TL072circuit\_2}

\caption{Prototype 0.1.03 - Improved Op Amp Circuit}

\label{fig:103opamp}

\end{wrapfigure}

The Operational Amplifier designed in Prototype 0.1.01 (Figure~\ref{fig:101circuit}) was improved for better performance. One of the key problems of the initial circuit was that a single amplifier was being used to amplify the microphone signal by 10,000 or 80dB. Since I was using a dual-TL072 chip, it made sense to use two-stages for amplification, with an overall gain of 96dB but with each stage only providing a fraction of that. This would release the strain on each amplifier and meant that my signal would no longer be inverted at the input to the Arduino's ADC.

In the first design, the 1$\mu$F decoupling capacitor and the gain resistor at the input of the op amp was generating a high pass pole at 1.5kHz, which was significantly damaging the audio signal but leaving much of the high frequency thermal noise alone. To fix this, the input resistor of each op amp was set to 1k$\Omega$ to create a second order high pass filter at 159Hz such that it did not impact as much upon the audio signal. A first order low pass filter was created using the feedback loop of the first amplifier stage to remove frequencies above 15.9kHz, which ultimately corresponded to thermal noise.

After filtering the signal, I found that the overall gain was quite low so I set the feedback resistor of the second amplifier stage to 1M$\Omega$ to produce an overall gain of 100,000 or 100dB.

The capacitor across the power rails was also reduced to 10$\mu$F as the previous 100$\mu$F one was somewhat unnecessary. Figure~\ref{fig:103opamp} shows the final op amp circuit.

\subsubsection{Implementing a Programmable Gain Amplifier}

The PGA that I decided to buy was the \textit{AD605} by Analog Devices because its gain could be programmed with a simple analogue voltage and it was available as a Dual-Inline Package, useful for testing on breadboard, as well as in surface mount for final implementation on a printed circuit board.

The AD605 is a two-stage amplifier with a settable amplification range using feedback resistors and whose gain value is dependent on a dedicated control signal. I was able to set the amplification range by modifying two feedback resistors. Setting both resistors to 0$\Omega$ (i.e. short circuit) would give me a range of -28dB to 68.8dB and setting them to infinite (i.e. open circuit) would correspond to a range of 0dB to 96dB. Of course I opted for the latter as that was more appropriate for my circuit as I did not need to attenuate the microphone signal. I used a DC level capacitor to offset the input signal by 2.5V to ensure that its zero-level fell exactly in the centre of the input range for the Arduino's ADC.

When designing the AD605 circuit, the input pins required decoupling capacitors which also set a high pass corner on the signal in conjunction with the pins' internal resistances. The internal resistance of each of the input pins was determined to be 175$\Omega$, and therefore decoupling capacitors no smaller than 9.09$\mu$F were required to ensure a high pass cutoff lower than 100Hz. I decided to set the values of these decoupling capacitors to 10$\mu$F to ensure that most of the audible range of frequencies (20Hz to 20kHz) was preserved. The output of the amplifier was also low pass filtered with a combination of a 82k$\Omega$ series resistor and a 100pF capacitor to ground connected to the Arduino's ADC input in order to remove all instances of thermal noise above a cutoff of 19.4kHz.

\begin{wrapfigure}{r}{0.3\textheight}

\centering

\subfloat[AD605 Circuit]{

\includegraphics[scale = 0.05]{Images/ad605\_circuit}

}\\

\subfloat[AD605 Circuit with Microphone on Veroboard]{

\includegraphics[scale = 0.05]{Images/ad605\_circuit2}

\label{fig:veromic}

}

\caption{Programmable Gain Amplifier Circuit implemented on Breadboard}

\label{fig:ad605}

\end{wrapfigure}

To set the gain of the AD605 once its circuit had been designed the Arduino was to be programmed to output an analogue voltage. However, the Arduino Uno that I was using could only output PWM (Pulse-Width-Modulated) signals, meaning that the amplifier was receiving pulses of 5V and 0V rather than a constant particular voltage. In order to fix this, the PWM signal from the Arduino was low pass filtered with a series resistance of 68k$\Omega$ and a 10$\mu$F capacitor to ground, which was required to obtain a DC voltage from the Arduino's 490Hz PWM signal. This voltage could then be set between 1.25 and 2.5V, which would correspond to gains of 0dB and 96dB respectively, by assigning a number between 63 and 127 to the relevant output pin on the Arduino. Setting the Gain Voltage Signal to zero would turn the amplifier off.

Appendix~\ref{fingerboardsch} shows the full circuit diagram for Prototype 0.2.03, including the pin connections for the AD605 chip to implement the discussed circuit with all decoupling capacitors, output lowpass filter and PWM-Analogue filter. The diagram also shows a potential divider connected to the Voltage Reference pin using two 10k$\Omega$ resistors to give the 20dB/V scaling that maps the Gain Voltage Signal to the generated gain of the amplifier. After implementing this circuit, I decided to use it over the two-stage TL072 amplifier since I had the ability to control the gain digitally through the Arduino.

\subsubsection{Arduino Code}

The Arduino's code for reading samples and writing them to the serial bus was also modified. Instead of using a circular buffer which introduced redundancies due to differences in its reading and writing speeds, an endless loop was written to read a single sample value and write it to the serial bus iteratively. With this method, the input sample rate was considerably hampered by the \textit{Serial.println()} function that wrote the sample to the serial bus since another sample could not be read until the previous had been written, but it removed the possibility of inaccuracies in the transmission of data such as repeated or lost sample values.

\subsubsection{Microphone Circuit on Veroboard}

I isolated the microphone circuit onto a small piece of Veroboard, as can be seen in Figure~\ref{fig:veromic}. This allowed me to quickly plug the microphone into whichever amplifier circuit I was testing and I would also be able to emulate the final usage of the microphone by gaining the ability to move it around with my fingers by removing it from the breadboard.

\newpage

\subsection{Prototype 0.2.01} \label{0201}

Prototype 0.2.01 involved a significant development in the Software Application as I shifted from coding in Python to C++. This was after finding that the signal from the microphone was glitched when attempting to play it out of the speakers and this was due to code redundancies in the \textit{PyO} library for audio playback. Essentially, I realised that as much as I developed the User Interface, the audio would not simply be as robust or clean as I would like it to be, and the best way forward would be to switch over to C++ as soon as possible as it provided a much more efficient way of programming despite the fact that I would have to find new libraries and methods for implementing what I already had in Python.

\subsubsection{New Program using JUCE in C++}

I found a library called JUCE\textsuperscript{\cite{juce}} which was designed to code audio-based applications in C++. Introjucer\textsuperscript{\cite{introjucer}}, a program for setting up a JUCE application, provided me with skeleton code and allowed me to create GUI elements as classes which I could instantiate from the main application. The skeleton code automatically generated two running threads: one to manage audio samples based on a hidden recurring timer, and one to listen for events in the GUI. The interfacing of the application with my computer's Core Audio Device was hidden and I simply had to write code to provide the program with audio samples upon request.

I used Introjucer to develop two GUI classes to run under the main application: an \textit{Overlay} class to show the 'Add Board' and 'Remove Board' buttons as before, and a \textit{Board} class to display the elements for selecting a serial port and the required function. The main application instantiates a single \textit{Overlay} object, which can then spawn as many \textit{Board} objects as are required. This in fact allows me to integrate the 'Bus' functionality from Python into the Board class, to perform functions and display the GUI elements all from the same class. Figure~\ref{fig:juceflow} shows a system diagram of the JUCE application. Like the Python program it still utilises threading (using the C++ library \textit{pthread}\textsuperscript{\cite{pthread}}) to call the 'Voice' and 'Sample' functions, only this time the produced audio samples are pulled into the main application's audio request thread and pushed into the audio device from there.

\begin{figure}[H]

\centering

! MISSING IMAGE !

%\includegraphics[scale=0.5]{}

\caption{System Diagram for JUCE Program, Prototype 0.2.01}

\label{fig:juceflow}

\end{figure}

\subsubsection{Designing the GUI}

\begin{figure}[H]

\centering

\includegraphics[scale=1]{Images/JuceConcept}

\caption{Initial Sketches of new Juce GUI}

\label{fig:juceguisketches}

\end{figure}

Figure~\ref{fig:juceguisketches} shows concept sketches for the JUCE GUI. Upon startup, the application will show only the 'Add Board' button, as was the case with the Python GUI. However, this GUI will be more eye-friendly than the Python one in that it will look much prettier. When a new board is added, the 'Add Board' button will shift to the top of the screen and the centre will be populated by a box with all the necessary elements, such as selecting the serial port and function, as defined by the \textit{Board} class. The 'Remove Board' button will then also appear below the \textit{Board} object. The addition of more boards will result in more of these objects being spawned and they will collectively be centred on the screen.

The main application continuously runs a 'paint' function to re-generate the background of the main frame as part of its GUI management thread. I exploited this function to display a waveform of the audio signal being processed behind the User Interface elements. This would provide visual feedback of the time-domain signal being sent to the audio device. Since the output signal is stereo or dual-channel (even though the microphone signal from my Boards are in mono or single-channel), I decided to overlay the Left and Right signals in red and blue respectively, which would show up as separate signals if the channels were different but would appear as a combined purple waveform if they were the same. This red and blue theme was used for the colour scheme of the entire application.

\begin{wrapfigure}{r}{0.4\textheight}

\centering

\includegraphics[scale = 0.2]{Images/paws\_3boards}

\caption{Multiple Boards Added in PAWS Application}

\label{fig:paws\_3boards}

\end{wrapfigure}

As an extra addition, volume controls were added to both the \textit{Board} class and \textit{Overlay} class, the former to set the gains of each individual board, and the latter as an overall amplitude control. These were also coloured in red and blue and were visualised as rotary faders to mimic what one would find on an audio mixing desk or professional studio equipment. Figure~\ref{fig:paws\_3boards} shows the produced GUI with controls for three boards and the waveform in the background shows the signal being received by one of the \textit{Board} objects connected to the Arduino with the 'Voice' function active.

\subsection{Prototype 0.2.02} \label{0202}

\subsubsection{Serial Communication and Bluetooth}

In order to connect to the Arduino using serial communication, I used a C++ library called \textit{termios}\textsuperscript{\cite{termios}}. This library allowed me to connect to a serial port and read data as was done in Python. I was not able to find a solution to detecting available ports to display in the GUI's 'Select Serial Port' drop down list, and therefore pre-specified the Arduino's port name for the time being.

I found that if my computer was paired with the HC-06 bluetooth module, it could be connected to as a serial device. After ensuring that I was able to read in values accurately through the usb cable, I attempted to test connecting to the bluetooth module. Connecting to the HC-06 was straightforward enough but I found that the module itself was affecting the microphone circuitry. The power rails were being distorted by a steady square wave, most likely due to the bluetooth module's change in power consumption as it pooled and transmitted data.

\begin{wrapfigure}{r}{0.4\textheight}

\centering

\includegraphics[scale=0.2]{Images/paws\_sample}

\caption{Triggering a Sample in PAWS Interface}

\label{fig:paws\_sample}

\end{wrapfigure}

Due to this, I stuck to using the usb cable for data transmission as I could also use it to power my arduino and microphone circuit. An independent power source for the microphone circuit would be ideal as the power drawn from the computer is corrupted with digital noise and the arduino only seems to add to that, but this solution seemed to work for the time being and was in fact more reliable than communication via bluetooth.

\subsubsection{Implementing Sample Function}

I was able to program a function to read a sample sound file, for example that of a drum hit, into a buffer but I had to rethink the way the function would trigger the sound because of the architecture of this program. The top-most class, called 'Main Component', was calling a function at a set frequency to request 512 sample values to be sent to the Audio Device. This function, \textit{getNextAudioBlock()}, was being used to read the sample values stored in the buffer of each 'Board' object, which was of course being populated by the Voice Function receiving data from the Arduino.

In order to implement the Sample Function, I used flags to determine whether a peak in the input audio signal had been detected, and the \textit{getNextAudioBlock()} would check these flags and pull the drum sound samples from its respective buffer. This meant a lot of processing was involved to ensure that only 512 samples of the drum sound were pulled at a time and that the position of the next samples to be played was recorded. I also enabled the function to reset the next sample to be played if another tap was detected during playback, so as to improve its responsiveness. This meant that the previous drum sound being played would be cut short rather than overlayed onto the new sound but it would react better to the user's performance.

All in all, it took some time to implement this function but I was able to play a sample drum sound from detecting peaks in the input signal from the Arduino. Figure~\ref{fig:paws\_sample} shows a screenshot of the GUI during a Sample being triggered by a tap on the microphone. However, since the peaks in the microphone signal did not necessarily represent 'taps' and could in fact be artifacts of speech or even noise, I found that the program misfired, or played drum sounds when it was not supposed to. The input signal would have to be filtered to ensure that only the sound of tapping a surface was being processed for peak detection.

\subsubsection{Interpolation for Voice Function}

As discussed previously, the Arduino reads a sample value and then writes it to the serial bus as quickly as it possibly can. The Arduino Uno's \textit{analogRead()} function is rated at being able to read 10,000 samples per second (i.e. a 10kHz sample rate) but since the function that writes values to the serial bus is so processor intensive, the effective sample rate is in fact much lower.

\begin{wrapfigure}{r}{0.4\textheight}

\centering

\includegraphics[scale=0.2]{Images/paws\_voice}

\caption{Capturing the Microphone Signal in PAWS GUI}

\label{fig:paws\_voice}

\end{wrapfigure}

Thus, when we read samples in to the Interface on the computer, and output them at the JUCE library's preset rate of 44.1kHz, we cannot hear the audio signal correctly. In order to remedy this change of sample rate, we need to upsample it to 44.1kHz. I used linear interpolation to perform the upsampling, whereby new samples are created in between two read samples that form a linear ramp from the first sample to the last. This increases the number of samples to represent a particular time period (the effective sample rate) while maintaining the waveform to some degree of accuracy.

When outputting the input samples without any upsampling, I found that the input sample rate was approximately 13 times smaller than 44.1kHz. This was determined by calculating that every time the \textit{getNextAudioBlock()} function requested 512 samples, the program had in fact only read in about 39 samples from the Arduino. Therefore the input signal was interpolated by a factor of 13, that is, twelve new sample values were created and stored in between any two input values in such a way as to form a linear change between them.

By doing this, the output signal was able to produce the correct frequencies and my voice could be heard at the correct pitch, if a bit unclear. The signal was found to be fairly distorted and fuzzy, which I determined to be digital noise from the Arduino.

\subsubsection{Coding Serial Transmission}

I attempted to code the serial transmission in order to improve the speed at which samples are written to the serial bus by the Arduino, and thereby reduce the time spent between reading samples.

I maximised the transmission baud rate to ensure that data was sent as quickly as possible between the Arduino and the Interface. I was using a 'print line' function to print the sample values to the serial bus followed by a 'new line' character to separate each value. However, I realised that this function was first converting the integer values into a string of characters, where each character was an ASCII representation of a digit, and then transmitting it. This meant that an integer of size two bytes was being transmitted as a maximum of five bytes: a maximum of four to represent each of the digits in the range 0 to 1024, and a final byte for the 'new line' character.

I removed this redundancy by simply writing the two bytes of the integer value directly to the serial bus followed by the 'new line' character, thereby reducing a variable transmission rate of three to five bytes of data to a constant three bytes per sample. I realised that the \textit{analogRead()} function only produced a 10 bit value, to give the range 0 to 1024, and therefore only occupied a fraction of the upper byte of an integer value. In order to reduce the transmission rate even further, I divided each sample value by 4, to remove the lower 2 bits, and simply transmitted the first byte. The transmission rate was then lowered to two bytes per sample.

The Interface was re-programmed to wait for a 'new line' character before reading in a byte of information, parsing it to an integer data type, and then multiplying it by 4 again before saving it to the buffer of samples, which could later be called for upsampling. Another method for reducing the size of a sample from 10 bits to 8 bits was by simply shifting all the bits along by two and then taking the first byte to ensure the most significant bit (MSB) was captured. The bit shifting worked on the same principle as a division by 4 but it was found that the audio quality was somewhat clearer using the division by 4, despite the two methods being absolutely identical in terms of function.

\subsection{Prototype 0.2.03} \label{0203}

\begin{wrapfigure}{r}{0.27\textheight}

\centering

\subfloat[Finger Board PCB]{

\includegraphics[scale=0.07]{Images/fingerboard\_pcb}

}\\

\subfloat[Arduino Shield PCB]{

\includegraphics[scale=0.05]{Images/ardshield\_pcb}

}

\caption{Printed Circuit Boards for Prototype 0.2.03}

\label{fig:finalpcbs}

\end{wrapfigure}

With this prototype, I designed printed circuit boards for the hardware I had implemented on breadboard and 3D printed housings in the shape of rings to allow one to wear the microphone circuit on one's finger.

\subsubsection{Hardware on Printed Circuit Boards}

I used a software tool called \textit{Eagle}\textsuperscript{\cite{eagle}} to design my circuits and generate layouts for printed circuit boards (PCBs). I designed two circuits: one for the microphone which could physically be worn at your fingertips, and another as a shield to connect with the Arduino Uno.

The Finger Board was designed to house the microphone circuit, along with the AD605 programmable gain amplifier. Since the circuitry would require a positive rail voltage from the Arduino of 5V, a ground voltage, and a signal to control the gain of the amplifier, a connector would also be required to receive these voltages via a cable from the Arduino. A four way connector was chosen to account for the microphone signal being outputted to the Arduino.

The Arduino Shield was designed to be of the same size as the Arduino Uno with connections such that it could simply slot in place. It was designed to connect via a cable to not one but three Finger Boards, for such a time when the software was upgraded to be able to control three microphone inputs all at once. The circuitry for the HC-06 bluetooth module was also included just in case I wished to insert it in the future.

Appendices~\ref{fingerboardsch} and \ref{ardshieldsch} show the drawn circuit schematics for Prototype 0.2.03.

After drawing the circuit schematics, I designed the PCBs themselves in terms of their size and the layout of all the components. I chose the components to be as small as possible while maintaining the ability to solder them onto the boards easily. I used the same Kingstate Electret Microphones as before but I used a surface mount package of the AD605 chip, which happened to be slightly smaller than its Dual-Inline Package counterpart that I was testing on breadboard. For the resistors and capacitors, I opted for the MELF0204 and C3216 packages respectively as they were also surface mountable but large enough to solder with my hands. I also chose to use a 4-way male-to-male 3.5mm jack audio cable to connect the Arduino Shield to the Finger Boards. This meant that the female connectors required by both boards could be small as they featured concentric connection rings for the 4 signal paths, and I was also able to find them in a surface mountable package.

For the Shield, I used Single-Inline male connectors to slot directly into the pin headers in the Arduino Uno. The connectors for Shield were of course exactly the same as the ones for the Finger Boards.

Appendices~\ref{fingerboardpcb} and \ref{ardshieldpcb} show the PCB layouts for each of these boards and Figure~\ref{fig:finalpcbs} shows images of the final PCBs after soldering all the components down.

The Arduino Shield was designed to connect to three Finger Boards and provide a separate gain voltage to each of their programmable amplifiers separately. This design was designed to work with the current Interface, which could connect to the Arduino and access a single microphone circuit, and with a future Interface, detailed in Subsection~\ref{0204}, to allow the user to control the functionality of three Finger Boards individually.

\subsubsection{3D Printed Housings}

To wear the Finger Boards on your fingers, I required plastic housings that could be 3D printed. The design itself would have to consist a ring to sit around the top knuckle and a support to hold the PCB in place. The measurements I took to design the ring were of the index finger of my right hand.

I used a free software called \textit{SketchUp}\textsuperscript{\cite{sketchup}} to design a model of a Ring to be printed later. The tool itself was easy to learn how to use and the first version of the Ring, Figure~\ref{fig:fngrbrd\_1}, was designed as a line sketch which was then extruded by 10mm to form a 3D model.

\begin{wrapfigure}{r}{0.29\textheight}

\centering

\subfloat[Top View]{

\includegraphics[scale=0.2]{Images/fngrbrdmod1\_top}

}

\subfloat[Side View]{

\includegraphics[scale=0.195]{Images/fngrbrdmod1\_side}

}

\caption{Fingerboard Ring 3D Model - Version 1}

\label{fig:fngrbrd\_1}

\end{wrapfigure}

Once the Ring had been designed, I used \textit{Cura}\textsuperscript{\cite{cura}} to detail how it would be printed using an \textit{Ultimaker 2}\textsuperscript{\cite{ultimaker}}. The tool allowed me to set the fill of the piece (i.e. hollow to solid) and the depth of each printed layer, which would determine the precision of the print. After setting all of the parameters correctly, I was able to print the Ring.

With the first version, I found that I had overestimated the dimensions, The ring was too large and slid all the way down the bottom of my index finger, which is not where it should have been, and the support at the top did not hold the PCB in place at all.

So, I redetermined the measurements and designed a second version, shown in Figure~\ref{fig:fngrbrd\_2}. This version was designed to have less of a curvature around the bottom to prevent it from being as bulky as the first version. I also added a 'lip' like feature to the bottom of the ring, which would sit under the tip of the index finger and produce a loud tone when tapped on a surface. This design would in future help my Interface to better determine whether a surface was being tapped and therefore make my Sample Function a lot more robust.

Appendix~\ref{fingerring\_dim} shows dimensioned sketched of Version 2 of the Ring.

\begin{figure}[H]

\centering

\subfloat[Top View]{

\includegraphics[scale=0.2]{Images/fngrbrdmod2\_top}

}

\subfloat[Side View]{

\includegraphics[scale=0.195]{Images/fngrbrdmod2\_side}

}

\caption{Fingerboard Ring 3D Model - Version 2}

\label{fig:fngrbrd\_2}

\end{figure}

\newpage

\subsection{Prototype 0.2.04} \label{0204}

The previous prototypes have all been designed with the idea of connecting to PAWS Boards one by one, with each Board housing a single microphone. This prototype deals with building an Interface for connecting to a single Arduino Uno that can control three different Finger Boards. The hardware was previously designed to allow for this design and the Interface had to be modified slightly to provide individual control of each Finger Board.

\subsubsection{Connecting Multiple Boards}

To allow the Interface to control each of the Finger Boards attached to the Arduino Uno separately, I had to make several changes to the functions of the two main classes that controlled the GUI. The \textit{Overlay} class was previously in charge of adding and removing \textit{Board} objects. For this prototype, I required the \textit{Overlay} class to perform the connection to the Arduino Uno immediately; the user would not be able to select the serial port at which the Arduino was located. This was simply to make the entire experience easier, and once the Arduino was connected to, three \textit{Board} objects would be immediately spawned. The \textit{queueInput( )} function that was used to receive samples from the Arduino was moved to the \textit{Overlay} class and was set running as soon as the Arduino was connected to. This function was also modified to place the samples it received into the buffers of the respective \textit{Board} objects. Because these functionalities were moved away from the \textit{Board} class, the dropdown list to select the serial port was entirely removed from this class.

The Arduino code was then modified to be able to read the sample values of the three Finger Boards and transmit them serially to the Interface. As before, I used a new line character to notify the Interface of the start of a 'packet' of data, and then proceeded to sample each of the three microphone signals in turn and transmit them. This packetisation technique helped to make the transmission robust as the \textit{queueInput( )} function on the Interface side could calculate exactly which received sample corresponded to which Finger Board.

\begin{figure}[H]

\centering

! MISSING !

%\includegraphics[scale = 0.2]{Images/paws\_3boards\_final}

\caption{Prototype 0.2.04 - Interface}

\label{fig:paws\_3boards\_final}

\end{figure}

\subsubsection{Enhancing User Control}

To enhance the user's control over the sample sounds triggered with each Finger Board, I implemented a drop down list in the \textit{Board} Interface to display available sound files and allow the user to select one on the fly. This element was programmed to trigger a function that would load the correct file into the respective \textit{Board} object's sample buffer, which the Interface would read from during playback.

I also found that each \textit{Board}'s amplitude control was not applied correctly to the 'Sample' function, and that it only affected the volume of the 'Voice' function. Therefore, I made a few minor adjustments to ensure that the user could set the volume of each selected sample sound relative to the others for better control over their musical production.

I also renamed the text at the top of each \textit{Board} object to display "One", "Two" and "Three" as opposed to just "Board", to further improve the user's interactive experience.

\subsubsection{Filtering the Audio Signal}

Filtered input signal using matlab analyses. Can stick graphs and plots in this section if manage to do this.

filtering sample function to only hear lip smack from finger ring.

filtering out the pwm noise - not working in software so have designed new shield

Figure~\ref{ardshieldpcb2} shows the PCB Layout for version 2 of the Arduino Shield.

\newpage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\section{Evaluation} \label{Evaluation}

\subsection{Prototype 0.2.04 Evaluation} \label{Prototype Evaluation}

\subsubsection{Hardware Analysis}

\begin{table}[H]

\begin{center}

\begin{tabular}{|l l l}

Circuit & Rail Voltage (V) & Noise Ripple (V$\_{p-p}$)\\ \hline \hline

\end{tabular}

\end{center}

\caption{Noise Levels In Circuitry}

\label{tab:noise}

\end{table}

1) rail noise

\\new fingerboard without extra electro cap

\\- rail noise on board

\\- rail noise on shield

\\old board with extra electro cap

\\- rail noise on board

\\- rail noise on shield

\\stick electro cap on shield

\\- rail noise on board

\\- rail noise on shield

2) freq response

\\fft of signal from mic

\\- on board

\\- on shield

\\- screenshots of scope response

\\fft of signal received by app

\\- chuck values to text file

\\fft of upsampled signal in app

\\- chuck values to text file

3) filter

\\design LPF in matlab for upsampled signal

\\- graphs of signal before and after

\\save coeffs to text file and implement in app

4) profiling

\\- Calculate time taken to execute getnextaudioblock vs. output rate (44.1k)

\subsubsection{Software Analysis}

The Arduino Uno was responsible for sampling the microphone signals of all connected Finger Boards and transmitting them to the Interface. According to Arduino's documentation, the \textit{analogRead( )} function responsible for sampling input voltages is limited to generating 10,000 samples per second\textsuperscript{\cite{analogread}}. However, since the code samples each connected Finger Board and transmits them in turn, along with a single new line character used as a 'start of packet' indicator at the very beginning, the effective sample rate is considerably hampered. This change in sample rate affects both the frequency content of the signal able to be captured and the upsampling ratio required to produce a signal at a sample rate of 44.1kHz to be outputted by the Interface to the audio device.

I changed the packet size (i.e. the number of Finger Boards I sampled and transmitted in each loop) and then calculated the time taken to process the entire packet, including sampling and transmission. This processing time can be argued to be the sample period of a signal, since this is the time taken before another sample can be taken of a particular signal. Therefore, the sample rate of each signal $f\_s = \frac{1}{time/packet}$. From this I was able to calculate the effective sample rate, or the overall number of samples read per second by the Arduino, with the equation $f'\_s = \frac{samples/packet}{time/packet}$. This effective sample rate gives us an idea of the fraction of time we spend per packet on sampling data as opposed to transmitting data, and can be compared against the Arduino's rated frequency of 10kHz. From the signal sample rate $f\_s$ we can also determine the upsampling ratio required by the Interface to transform the signal to its required rate of 44.1kHz. The upsampling ratio is calculated as $ratio = \frac{44.1}{f\_s}$ and these values are shown in Table~\ref{tab:packeteval}.

\begin{table}[H]

\begin{center}

\begin{tabular}{r| l l l l}

Samples & Processing Time & Signal Sample & Effective Sample & Upsampling \\

per Packet & per Packet (ms) & Rate (ksamp/sec) & Rate (ksamp/sec) & Ratio \\ \hline

1 & 0.17 & 5.882 & 5.882 & 7.50 \\

2 & 0.281 & 3.559 & 7.117 & 12.39 \\

3 & 0.42 & 2.381 & 7.143 & 18.52 \\

\end{tabular}

\end{center}

\caption{Table showing numerical statistics for different 'packet' sizes}

\label{tab:packeteval}

\end{table}

We can see that even though it takes longer to process larger packets, the effective sample rate seems to increase. This is due to the fact that with a packet size of one sample, half of the transmitted packet is the starting new line character, and the second half is an actual sample value, whereas with a packet size of 3, the new line character only takes up a quarter of the packet and the packet contains 3 sample values. This means that because less processing time is being spent towards transmitting the new line character relative to sampling, we are able to produce more samples per second, which directly corresponds to an increase in sample rate.

We can also see that the signal sample rates are very low, even worse than a telephone line, which is sampled at 8kHz or 16kHz. This means that the intelligibility of any voice information captured will almost non-present, however, the signal will still be able to pick up 'tapping' sounds, which usually feature high energies at 200 or 500Hz. Also, the calculated upsampling ratios directly correspond to the values I used in Prototype 0.2.02, detailed in Section~\ref{0202}, which I settled upon simply through trial and error and matching the frequency of my voice with what was being played back through the Interface from the microphone.

\subsubsection{Requirements Analysis}

///////need to analyse the prototype against the initial requirements.

\newpage

\subsection{Progress Evaluation} \label{Progress Evaluation}

Table~\ref{tab:listofprototypes} shows a list I developed for the Interim of my project detailing a plan of the successive development of prototypes. Table~\ref{tab:listofprogress} shows the actual progress I made over the duration of the project as a list of prototypes and their key features.

I initially planned for Prototype 0.1.xx to be a fully functioning program with digital gain control, wireless transmission and a separate power source for the PAWS Board. The following prototypes were then going to be focused on the implementation of specific features such as a Quantisation function to quantise any incoming 'taps' to a discrete time grid, or a Harmonisation function to trigger background harmonies depending on the primary audio generated by the user.

\begin{table}[H]

\begin{center}

\begin{tabular}{|l l}

Prototype & Features \\ \hline \hline

0.1.01 & Basic output of microphone and simple Sample Triggering function \\

0.1.02 & Cleaner microphone signal, single GUI\tablefootnote{GUI: Graphical User Interface} for Interface with both Voice and Sample functions \\

0.1.03 & PGA\tablefootnote{PGA: Programmable Gain Amplifier} in circuit for digital gain control \\

0.1.04 & Bluetooth transmission \\

0.1.05 & A fully featured GUI for user control \\ \hline

0.2.xx & All hardware on PCBs\tablefootnote{PCB: Printed Circuit Board} and 3D printed wearable housings \\

0.3.xx & Integration of SP\tablefootnote{SP: Signal Processing} functions such as Gain, EQ\tablefootnote{EQ: Equalisation}, Pitch Detection \& Correction \\

0.4.xx & Quantisation Function\tablefootnote{Quantisation Function: User can tap a tempo to quantise all produced sample sounds} \\

0.5.xx & Harmonisation Function\tablefootnote{Harmonisation Function: User can trigger virtual instrument samples to harmonise with their singing} \\

0.6.xx & Integrate Accelerometer onto PAWS Board and develop Motion feature on Interface \\

\end{tabular}

\end{center}

\caption{Interim Implementation Plan as a List of Prototypes and their Features}

\label{tab:listofprototypes}

\end{table}

The actual progress varied quite significantly, mainly due to the initial Python program not being able to handle audio processing code quickly enough. As discussed in Section~\ref{0201}, Prototype 0.2.xx involved a changeover to the JUCE library in C++, for which I had to completely restructure my program and re-implement everything I had done in Python. I also spent more time on making the GUI more engaging with an end-user oriented design and modern visuals. I was able to implement the Programmable Gain Amplifier, as planned, but it was mostly used for its cleaner signal over the Operational Amplifier circuit rather than its gain control capabilities. Printed Circuit Boards and 3D Printed wearable Rings were also produced later on, as wished for in the plan.

\begin{table}[H]

\begin{center}

\begin{tabular}{|l l}

Prototype & Features \\ \hline \hline

0.1.01 & Basic microphone circuit and test 'Voice' and 'Sample' functions in Python\\

0.1.02 & Full Python Program with GUI to perform 'Voice' and 'Sample' functions \\

0.1.03 & PGA in circuit for digital gain control \\ \hline

0.2.01 & New Program Structure and GUI designed using JUCE for C++ \\

0.2.02 & Implementation and Optimisation of 'Voice' and 'Sample' functions and data transmission \\

0.2.03 & All hardware on PCBs and 3D printed wearable housings \\

0.2.04 & Modification of Program to connect to multiple Boards simultaneously \\

\end{tabular}

\end{center}

\caption{Actual Implementation Progress as a List of Prototypes and their Features}

\label{tab:listofprogress}

\end{table}

Much of my project was dedicated to optimising audio processing code, in the sampling stage at the Arduino, the transmission between it and the computer, and all of the processing and data transfer within the Interface itself. The algorithms to upsample the input signals and to trigger sample sound files took the most time to code and improve, in order to ensure robustness and accuracy. The User Interface was focused upon with great detail too to ensure easy usability and an appealing look.

\newpage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\section{Improvements \& Enhancements} \label{Further Developments}

\subsection{Future Design (Prototype 0.3.xx)}

\subsubsection{Technical Specifications}

Prototype 0.2.04 was fairly bulky in design relative to the concept design and any further developments would need to remedy this. I realised that the only need for the Arduino Uno was to sample the connected microphone signals and transmit the captured data to the Interface where all of the processing was done. It was of course also used to set the gain of the microphone circuit's amplifier but this function was not used in the initially intended way and was in fact set to a value which remained constant during run-time.

Therefore, the Arduino Uno could be deemed to be a redundant component for what it was used to do, and could simply be replaced by an Analogue-to-Digital Converter (ADC) and a serial interface. This would mean that the entire circuitry could be contained on a single Finger Board, which would connect directly to a computer. A good enough ADC would also be able to detect the microphone signal's dynamic range and map the incoming voltages to the entire digital range of values, thereby removing the need for a bespoke amplifier as well. Some surrounding digital circuitry may also be required such as a clock, and Figure~\ref{fig:0304\_circuit} shows a mock-up system design for the Finger Board.

\begin{figure}[H]

\centering

\includegraphics[scale=2]{Images/FutureDesign}

\caption{Prototype 0.3.xx - System Design}

\label{fig:0304\_circuit}

\end{figure}

The inclusion of a battery would clean up the rail noise significantly compared to the power being drawn from the computer and suffering additive noise from the Arduino. The serial interface could be a simple microusb connector or even a bluetooth transmitter.

\subsubsection{Creative Functionalities}

accelerometer embedding

ideas such as air guitar, playing scales/notes

could even use accels for sample triggering function

\subsection{Other Improvements}

oversampling to compensate for transmission errors? sigma-delta methods [cite]\\

thermal isolation of microphone\\

coding of data. can most likely increase transmission rate with entropy and codeword analysis \\

trade-off between oversampling (more data/second) and transmission speed (less data/second) - look at current speed of reading data (queueInput) and compare with maximum transmission speed capable. (not all that reliable as most time dedicated to waiting for input data)

\subsection{Marketability}

needs significant increase in functionality before marketable. talk about user experience and freedom of musical expression needing to be met before product becomes marketable. currently still at stage where focus is purely technical (although some thought to user experience has been given by way of GUI design).

\newpage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\section{Finances} \label{Finances}

how much spent already. how expensive the final hardware would be (expected) depending on the types of components it requires.

breakdown in terms of prototype 0.2.04 PCB components.

could include cost of new design for prototype 0.3.xx.

\newpage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\section{User Manual For Prototype 0.2.04} \label{User Manual}

\begin{figure}[H]

\centering

\includegraphics[scale = 0.2]{Images/paws\_open}

\caption{Opening Screen of PAWS Application}

\label{fig:paws\_open}

\end{figure}

hardware:

how to attach fingerboard to ring

how to attach fingerboard to arduino shield

how to attach shield to arduino uno

how to attach uno to laptop

software:

on opening of program

how to add boards

how to remove boards

how to connect to board

how to select function

volume controls

red line (L) vs blue (R) vs purple (MONO)

\newpage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\newpage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\input{bib.tex}

\newpage

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\input{appendix.tex}

\end{document}