\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{Market Research} \label{Background}

This final year project has a short brief (shown in italics in the Project Specification section) and is 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}{0.4\textwidth}

\centering

\includegraphics[scale=0.6]{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}{0.3\textwidth}

\centering

\includegraphics[scale=0.5]{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=0.5]{Images/conceptsketchesleft}

\label{fig:conceptsketchleft}

}

\subfloat[PAWS Board: Sample Function]{

\includegraphics[scale=0.5]{Images/conceptsketchesright}

\label{fig:conceptsketchright}

}

\caption{Concept Sketch of Usage}

\label{fig:conceptsketch}

\end{figure}

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

\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 Design}\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.

\newpage

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

\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.

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

\centering

\includegraphics[width = 0.30\textheight, height = 0.20\textheight]{Images/slaptodrum}

\caption{Simulation of Sample Triggering}

\label{fig:samptrigmatlab}

\end{wrapfigure}

\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 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.

\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}

\begin{wrapfigure}{r}{0.4\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}{0.35\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}

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 GUI}

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.

\subsubsection{Python Program}

Figure~\ref{fig:pyblue\_class} shows a class diagram of the entire program and the functions that each element on the GUI is related to. 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}

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.

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.

Figure~\ref{fig:ad605} shows 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{Mic Circuit on Veroboard}

I isolated the microphone circuit onto a small piece of Veroboard, as shown 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.

\subsection{Prototype 0.2.01}

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:juceclass} 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.

\subsubsection{Designing the GUI}

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.

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.

\begin{figure}[H]

\centering

\includegraphics[scale = 0.5]{Images/paws\_3boards}

\caption{Multiple Boards Added in PAWS Application}

\label{fig:paws\_3boards}

\end{figure}

\subsection{Prototype 0.2.02}

\subsubsection{Testing Bluetooth Module}

attempted bluetooth addition

noticed power discontinuities in the form of square waves. have not analysed quantitatively but have decided to stick to usb transmission for now. would also solve the need for external power despite the rails being noisier due to digital noise from both computer and arduino

\subsubsection{Implementing Sample Function}

programmed sampling function.

\subsubsection{Interpolation for Voice Function}

problems with voice output 39/512 samples shit

software interpolation for upsampling but still distortion and fuzz.

\subsubsection{Coding Serial Transmission}

reduction of transfer size (to one byte) from arduino and increase of baud rate. went from characters representing data values to actual integer values. tried division and bit shifting. could look at further coding.

\subsection{Prototype 0.2.03}

\subsubsection{Hardware on Printed Circuit Boards}

building pcb. eagle designs and circuit schematics. sizes for through hole pads and why MELF smds and soics etc. decision to use 4-way 3.5mm jack for connection.

\subsubsection{3D Printed Housings}

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

\centering

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

\caption{Fingerboard Ring 3D Model - Top View}

\label{fig:fngrbrd\_top}

\end{wrapfigure}

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

\centering

\includegraphics[scale=0.2]{Images/fngrbrdmod\_side}

\caption{Fingerboard Ring 3D Model - Side View}

\label{fig:fngrbrd\_side}

\end{wrapfigure}

3d printing finger rings for finger boards. infill and other parameters used to print the model created in sketchup

first model overestimated sizes therefore was too big on finger and could not hold pcb correctly. second model resized and printed.

\subsection{Prototype 0.2.04}

\subsubsection{Filtering the Audio Signal}

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

\subsubsection{Enhancing User Control}

////allowing user to select sample

\subsubsection{Connecting Multiple Boards}

////multiple board connection

////playing three different sounds at once

\newpage

\subsection{Further Prototypes - SCRAP THIS} \label{Further Prototypes}

Table~\ref{tab:listofprototypes} shows a list of the implementable prototypes and their improved features.

Prototype 0.1.02 will include significant improvements over 0.1.01. On the hardware side, it will include an independent power supply (battery) for the microphone circuit in order to ensure it is not corrupted by the microcontroller's digital noise. The Atmel ATMEGA328 microcontroller will also feature a more robust method of reading and writing the microphone signal such that we avoid the inaccuracies of the currently implemented circular buffer.

The Interface will integrate the 'Voice' and 'Sample' functions into a single program and will perhaps even feature a basic GUI to allow the user to control the function of the PAWS Board.

Once all of the initially specified system components have been implemented in some way, Prototypes 0.2.xx onwards will involve expanding the functionality of the instrument to give the user more creative control over the sound they wish to produce. Further prototypes will also explore marketable designs of the instrument, including more professional microcontrollers for smaller PAWS Board package designs, and Interface functions coded in lower-level programming languages such as \emph{C} to optimise the signal processing chain.

\begin{table}[H]

\begin{center}

\begin{tabular}{|l l}

Prototype & Features \\ \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 PCB\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/Gyroscope onto PAWS Board and develop Motion feature on Interface \\

\end{tabular}

\end{center}

\caption{List of Prototypes and their Features}

\label{tab:listofprototypes}

\end{table}

\newpage

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

\section{User Manual For Current Prototype}

\begin{figure}[H]

\centering

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

\caption{Opening Screen of PAWS Application}

\label{fig:paws\_open}

\end{figure}

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

\centering

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

\caption{Voice Function in PAWS GUI}

\label{fig:paws\_voice}

\end{wrapfigure}

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

\centering

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

\caption{Sample Function in PAWS GUI}

\label{fig:paws\_sample}

\end{wrapfigure}

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

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

\section{Project Plan - SCRAP THIS} \label{Project Plan}

This section briefly describes the progress plan of the Project including information on the iterative implementation of prototype designs and the evaluation of the overall success.

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

Table~\ref{tab:implementationplan} shows the list of Prototypes from Table~\ref{tab:listofprototypes}, Section~\ref{Further Prototypes}, and their estimated completion dates, alongside the submission dates for all reports and the final presentation.

\begin{table}[H]

\begin{center}

\begin{tabular}{|l l}

Prototype & Completion Date (2015)\\ \hline

0.1.01 & 20th January \\ \hline

\hline \emph{Interim Report} & 2nd February \\ \hline \hline

0.1.02 & 11th February\\

0.1.03 & 17th February\\

0.1.04 & 20th February\\

0.1.05 & 28th February\\ \hline

0.2.xx & 18th March\\

0.3.xx & 1st April\\

0.4.xx & 17th April\\

0.5.xx & 1st May\\

0.6.xx & 15th May\\ \hline

\hline \emph{Abstract} & 8th June \\ \hline

\hline \emph{Final Report} & 17th June \\ \hline

\hline \emph{Presentation} & TBC \\ \hline

\end{tabular}

\end{center}

\caption{Implementation Timetable of Deliverables}

\label{tab:implementationplan}

\end{table}

Prototype 0.1.05 should be functioning by the end of February 2015. This prototype should meet all of the basic specifications of the musical instrument. Subsequent prototypes will improve upon either the functionality, what the user can do with the instrument, or usability, how the user can interact with the instrument, and I have given myself approximately half a month to ensure each prototype is fully functional.

Further detailed plans of implementation for each subsequent prototype will be drawn up as needed in order to ensure the successive completion of all the necessary design components. This production cycle where each iteration of prototypes builds upon the previous, ensures that, no matter the progress, there will always be a version of the musical instrument that is functional across all of its basic requirements.

\subsection{Evaluation Plan} \label{Evaluation Plan}

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.

Each fully built prototype will undergo a series of quantitative tests such as calculating audio latency and distortion levels, as well as more qualitative criteria described in Table~\ref{tab:criteria}. These will help to identify the achieved improvements of each prototype and would also possibly help to reconsider the direction in which subsequent prototypes would have to be taken if certain criteria are found to not have been met.

\begin{table}[H]

\centering

\begin{tabular}{|ll}

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}

\begin{table}[H]

\centering

\begin{tabular}{|l}

\emph{The PAWS Board}\\ \hline

Can it capture audio? \\

Can it capture motion? \\

Can it connect to the Interface? \\

Can it send captured data to the Interface?

Is it wearable through 3D printed attachments? \\

Can it be switched on and off?\\

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

\hline \hline

\emph{The Interface} \\ \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 process 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 core output device? \\

Can it save the output audio to file?\\

\end{tabular}

\caption{List of Specifications}

\label{tab:specifications}

\end{table}

Shortly before the presentation and final report, when my prototypes are at a stage where they can be demonstrated fairly easily, I will conduct several field tests as part of my evaluation of the instrument as a potential commercial product. The musical instrument is aimed at allowing anyone, musically experienced or not, to generate their desired sounds quickly and easily without having to go through extensive setup processes. Therefore, I will take my final prototypes to the general public to gauge their interest and to further evaluate how well the product meets its user-based specifications. I will test my product with students and young adults of ages 18-24 as they are able to adapt to new technologies fairly quickly and I will therefore be able to efficiently test my prototypes with as many people as possible.

\newpage

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

\section{Evaluation} \label{Evaluation}

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)

\newpage

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

\section{Further Developments} \label{Further Progress}

\subsection{Future Design}

prototype 0.3.xx

new adc directly to bluetooth circuit. analysis of technologies involved, such as SPI.

\subsection{Improvements \& Enhancements}

oversampling to compensate for transmission errors?

thermal isolation of microphone

coding of data. can most likely increase transmission rate.

trade-off between oversampling (more data/second) and transmission speed (less data/second)

\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 0.2.05 PCB components.

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

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

\newpage

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

\input{bib.tex}

\newpage

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

\input{appendix.tex}

\end{document}