**School of Electrical**

**and Electronic Engineering**



**Robot Orchestra**

**Final Report**

**Group 11**

|  |  |  |
| --- | --- | --- |
| Group Members: | Buruiana, Andrei | 9478411 |
|  | Chanda, Joyanto | 9015629 |
|  | Dimou, Theodoros | 9126435 |
|  | Fumagalli, Francesco | 9017237 |
|  | Petrovs, Antons | 9474345 |
|  | Simpson, Joshua | 8929879 |

Tutors: Prof. Danielle George and Dr. William McGenn

Date: 15 May 2018

Table of Contents

[1. Executive Summary 3](#_Toc512936913)

[2. Introduction 4](#_Toc512936914)

[2.1. Literature Review 5](#_Toc512936915)

[2.1.1. Conductor 5](#_Toc512936916)

[2.1.2. Keyboard 5](#_Toc512936917)

[2.1.3. Xylophone 5](#_Toc512936918)

[2.1.4. Stepper 5](#_Toc512936919)

[2.1.5. Tesla coil 5](#_Toc512936920)

[2.1.6. Panpipes 5](#_Toc512936921)

[2.1.7. Guitar 5](#_Toc512936922)

[2.2. Aims and Objectives 5](#_Toc512936923)

[2.3. Motivation 6](#_Toc512936924)

[2.4. Project Roles 6](#_Toc512936925)

[2.5. Required Skills 7](#_Toc512936926)

[2.6. Interdependencies 7](#_Toc512936927)

[2.6.1. Report structure 8](#_Toc512936928)

[3. Technical Chapters 8](#_Toc512936929)

[3.1. Conductor 8](#_Toc512936930)

[3.1.1. Conductor Overview 8](#_Toc512936931)

[3.1.2. SDManager 9](#_Toc512936932)

[3.1.3. Bithoven 10](#_Toc512936933)

[3.1.4. Wireless Network 10](#_Toc512936934)

[3.1.5. Conclusion 11](#_Toc512936935)

[3.2. Software Structure 11](#_Toc512936936)

[3.3. Xylophone 11](#_Toc512936937)

[3.3.1. Xylophone hardware 11](#_Toc512936938)

[3.3.2. Solenoid Software 11](#_Toc512936939)

[3.3.3. Piezo sensors 12](#_Toc512936940)

[3.3.4. Xylophone Testing 15](#_Toc512936941)

[3.3.5. Summary 15](#_Toc512936942)

[3.4. Keyboard 15](#_Toc512936943)

[3.4.1. Overall design description (reference literature review) 15](#_Toc512936944)

[3.4.2. Solenoid Selection 16](#_Toc512936945)

[3.4.3. Supporting the Solenoids 16](#_Toc512936946)

[3.4.4. Solenoid PCB design 17](#_Toc512936947)

[3.4.5. Supporting the Bosch Bar 17](#_Toc512936948)

[3.4.6. PCB Support Plate 18](#_Toc512936949)

[3.4.7. Keyboard software 19](#_Toc512936950)

[3.4.8. SN74HC595 Shift Registers 19](#_Toc512936951)

[3.4.9. Structure of the program 20](#_Toc512936952)

[3.4.10. Keyboard Testing 21](#_Toc512936953)

[3.4.11. Summary 21](#_Toc512936954)

[3.5. Stepper Motors 22](#_Toc512936955)

[3.5.1. Stepper Motor Hardware 22](#_Toc512936956)

[3.5.2. Acoustic box 22](#_Toc512936957)

[3.5.3. Digital Signal Synthesizer 23](#_Toc512936958)

[3.5.4. Motor Driver Board 24](#_Toc512936959)

[3.5.5. Stepper Motor Software 24](#_Toc512936960)

[3.5.6. Stepper motor Testing 25](#_Toc512936961)

[3.5.7. Summary 25](#_Toc512936962)

[3.6. Tesla coil 26](#_Toc512936963)

[3.6.1. Tesla coil testing 29](#_Toc512936964)

[3.6.2. Summary 29](#_Toc512936965)

[3.7. Panpipes 29](#_Toc512936966)

[3.7.1. Versions 29](#_Toc512936967)

[3.8. Outreach 30](#_Toc512936968)

[4. Summary 30](#_Toc512936969)

[4.1. Overall achievement 30](#_Toc512936970)

[4.2. Future work 30](#_Toc512936971)

[4.2.1. Conductor 30](#_Toc512936972)

[4.2.2. Keyboard 30](#_Toc512936973)

[4.2.3. Xylophone 30](#_Toc512936974)

[4.2.4. Stepper motors 30](#_Toc512936975)

[4.2.5. Tesla 30](#_Toc512936976)

[4.3. Progress against Gantt Chart 30](#_Toc512936977)

[4.4. Management 30](#_Toc512936978)

[5. Conclusion 30](#_Toc512936979)

[6. References 30](#_Toc512936980)

# Executive Summary

As part of the final year of The University of Manchester’s School of Electrical and Electronic Engineering Meng Programme, Team 11 has developed a new core infrastructure for the university’s Robot Orchestra project. The aim of the project has been to expand The University of Manchester’s existing Robot Orchestra by adding four new robot instruments as well as an electronic conductor to control the instruments. This would increase the orchestra’s capabilities and allow for a wider selection of musical styles to be played while also being able to be used as a standalone band. Additionally, the new instruments needed to be able to play at least two recognisable songs and allow for future additions of instruments as well as for easy transportation and set up.

The individual objectives of the project have been: choosing the song to be played, selecting four instruments, designing and manufacturing the instruments and finally, assembling them into a core orchestra that can play at least two songs together. The project was motivated by the rising interest in the existing Robot Orchestra which, if expanded, will be able to showcase the operation of multiple embedded systems to perform a shared task that combines engineering and art in a fun and exciting way. Additionally, this expansion can also lead to the Robot Orchestra appealing to a wider audience and advertising engineering to people who would not normally consider it.

The instruments that have been developed for this project are: a keyboard, a xylophone, Tesla coils and steppers motors. These are able to play *Eye of The Tiger* by *Survivor* and *Californication* by *Red Hot Chilli Peppers.* Additionally, the developed instruments have been individually tested with other songs as well and it has been determined that they can be easily integrated with the existing Robot Orchestra to play other songs together with another set of instruments. For controlling the designed orchestra, a conductor has been developed which consists of a Raspberry Pi platform that is capable of extract the key information from the industry-standard file format used for music encoding (MIDI), convert it to a format that can be used by the designed instruments and sent wirelessly to each instrument so that the user-level effort can be minimised.

The project has a dual commercial applicability. First, the designed orchestra can be used at trade fairs by multiple technology companies to showcase their products that are being used in this atypical application. By using this orchestra, the companies would have a higher chance of drawing attention and attracting new customers. Secondly, the project can also be used by universities to illustrate the potential projects that students could be involved in as part of their degree and thus determine more high school students to apply to their programmes. To date, the project has been used for one of the university’s open day events where it has been greatly appreciated by Year 8 students. Additionally, project has also been accepted to Manchester Museum of Science and Industry’s MakeFest event which is aimed at raising the interest of the general public in engineering and craft. This goes to demonstrate the high interest in this project and its substantial social-good and commercial potential.

The report concludes that the project has been completed successfully, having met all the requirements set in the beginning. The instruments can play two recognisable songs independently as well as in an orchestra, are easy to transport and set up and the designed conductor ensures that future additions of instruments can be made. In terms of future work, the designed orchestra could be enhanced with a higher number of instruments and adapting the orchestra to play other recognisable songs could prove beneficial as it would increase its’ visibility and potential to draw a crowd at industry-related events.

# Introduction

The University of Manchester Robot Orchestra was created in 2016 to celebrate Manchester becoming the European City of Science. Since it was created, the orchestra has gone on to feature in a popular BBC iPlayer documentary [1], played at the Museum of Science and Industry in Manchester and gained backing from Siemens, EPSRC, The Granada Foundation and National Instruments [2]. The objective behind the orchestra is to show that a range of technical engineering skills can be used to create something that is accessible to a mass audience, which hopefully inspires more young people to pursue a career in the technical sciences. So far, the orchestra has had a positive reception, so this project has been dedicated towards developing a new stand-alone core. Four new instruments will be constructed as well as a new conductor to improve the reliability and flexibility of the orchestra.

## Literature Review

### Conductor

### Keyboard

### Xylophone

### Stepper

### Tesla coil

### Panpipes

### Guitar

## Aims and Objectives

**2.2.1. Aims**

The aim of this fourth-year project is to create four or more new robot music instruments. These robots should follow a consistent aesthetic and theme that should represent electronic and mechanical engineering so it does not fall out of line with the existing robot orchestra. They must also be controlled by an electronic conductor which will control the robots playing the instruments.

These four robots need to be able to reproduce at least two different songs or themes, but should also be designed to be able to be programmed to play a variety of other musical pieces. A high level of autonomy is expected from the robots; ideally the only interaction should be the communication between the conductor and the robots.

Apart from the flexibility in the ability to play different styles of music, this new core of instruments as well as the conductor should be mindful of future additions and should be expected to accommodate for new instruments as well as the old ones. The robot instruments must also allow being disassembled, transported and reassembled quickly and easily so that time is saved when preparing for concerts and demonstrations.

**2.2.2. Objectives**

After the roles were assigned to individual members planning was done for the upcoming blocks, setting the following objectives:

1. Propose designs for the four new instruments:

* Select multiple songs and assign instruments that could perform its parts.
* Select two suitable songs that use four of the same instruments.
* Divide the team into four groups to work on designing each instrument.

1. Construct the four instruments:

* Design and manufacture prototypes for each instrument considering how they will interface with the conductor.
* Test the prototype instruments to play tunes or songs individually.
* Redesign / remanufacture as required based upon testing.

1. Design a conductor:

* Design the conductor which will communicate with the instruments.
* Assemble the conductor.

1. Assembling the new core orchestra:

* Test individual instruments with the conductor
* Interface the conductor with all the instruments.
* Synchronise the four instruments to produce coherent music.

## Motivation

The work undertaken in this project is motivated by the rising interest in the university’s existing Robot Orchestra with the orchestra having most recently been part of an engineering tour sponsored by the Royal Academy of Engineering as well as having been part of a BBC programme [1].

As the existing orchestra has a limited range of songs it can play, adding new instruments that can enhance its capabilities would allow for a wider selection of musical styles to be played and appeal to a broader audience. In addition to its outreach aspect, the project also stands to illustrate the operation of multiple embedded systems in performing a common task, making use of platforms such as Arduino and Raspberry Pi to control the performance of the orchestra.

Additionally, due to the combination of different development platforms that the orchestra uses, the team will also investigate its’ commercial exploitation by being leased to various businesses. As companies in the technology industry can be expected to be present at several trade fairs throughout the year, the Robot Orchestra offers a good opportunity for technology companies to showcase the creativity aspects of engineering as well as the applications of their products in an atypical project that has a high chance of drawing attention and attracting new customers.

Lastly, since it can be difficult to explain the benefits of pursuing one’s interest in science and engineering to an audience of different age categories and backgrounds, the combination of engineering and arts in the robot orchestra developed during this project can also serve as a good aid for the university to promote its courses as well as advertise engineering to people who normally would not consider it.

## Project Roles

In addition to the technical skills required for this project, the team will have to utilise several soft skills. Table 1 below shows the roles fulfilled by each of the team members and a brief description of what the role entails.

|  |  |
| --- | --- |
| **Project Manager**  **(Joyanto Chanda)** | The project manager role is responsible for tracking the progression of the project. In this project, a GANTT chart was used to determine the current progress of the project against the expected progress. This ensured that if adjustments to the plan needed to be made then they could be made whilst ensuring that project was still expected to be completed on time. |
| **Secretary**  **(Joshua Simpson)** | The secretary ensures that all the minutes and agendas are prepared for each of the meetings, both for individual team members and with supervisors. They are also responsible for ensuring that the minutes and agendas are filed away in a manner that makes them easily accessible. |
| **Auditor**  **(Andrei Buruiana)** | The auditor tracks the current expenditure of the project and ensures that the project is operating within the budget limits. In this project, this was done by having a Google Doc which contained details of all costs incurred to the project. |
| **Hardware Lead**  **(Theodoros Dimou)** | Important to define the approach to any upcoming hardware deadlines. Also, will be used to make decisions when the team is undecided on the approach to certain hardware challenges. Important to have prior experience in designing hardware (e.g. responsible for the bulk of the hardware design on the buggy project). |
| **Software Lead**  **(Francesco Fumagalli)** | The software lead in this project is a crucial role and was assigned to the person with the most experience of various embedded systems and in coding. They are responsible for defining the structure of the software for each of the instruments and how they will communicate with the conductor. |
| **Document Controller**  **(Antons Petrovs)** | Ensures that documents across the project have a consistent format and are mad easily accessible by each team member. They are also responsible for formatting and organising the two reports. |

Table Project Roles

## Required Skills

In a project like this several technical skills will be needed to create the four chosen instruments and the conductor. For example, the keyboard example in Section 2 [3] would require technical skills in analogue circuit design, programming in C and a solid understanding in MIDI to text conversion alongside basic musical knowledge.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Indicates where skills needed to be developed. | | | | | | |
| Degree | **Electronic Engineering** | **Mechatronic**  **Engineering** | | **Electrical and Electronic Engineering** | | |
| Skills | Andrei Buruiana | Antons Petrovs | Francesco Fumagalli | Joshua Simpson | Joyanto Chanda | Theodoros Dimou |
| Analogue Circuit Design | X | X | X | X | X | X |
| Electronic Circuit Design | X | X | X | X | X | X |
| Coding in C | X | X | X | X | X | X |
| Experience with Arduino |  |  | X |  | X |  |
| Experience with MyRIO |  | X |  |  |  | X |
| Experience with Raspberry Pi |  |  | X |  |  |  |
| 3D Modelling |  | X |  |  | X | X |
| PCB Design | X | X | X |  |  |  |
| Basic Music Knowledge |  |  |  | X |  |  |
| Experience in LabVIEW |  | X |  |  |  | X |

Table Skills required for project

During the design phase of the instruments several different embedded systems were considered for each of the instruments. From looking at Table 2 above the team had previous experience in working with the Arduino and working with Raspberry Pi. However, there was no previous experience when working with the National Instruments’ (NI) MyRIO. There was deliberation as to whether to include the MyRIO as a part of the project because of this reason however, ultimately it was decided that it would form a part of one of the instruments since it provided several benefits. It has significantly more digital IO pins than any of the other embedded systems as well as an in-built Wi-Fi module. In order to address the skills gap an application was submitted to the NI Student Sponsorship Project. Although we were not successful with the application, once hearing about our project NI still provided a support engineer to help the team in incorporating a MyRIO with the orchestra, which the team was able to contact through email or by phone.

## Interdependencies

Figure 1 below shows the main tasks for the project that had to be completed. The black arrows show which tasks are dependent on the other tasks. For example, for the “Receiving data” task to be completed for the conductor the software testing for each of the instruments had to be completed. Likewise, to synchronise all of the instruments together the construction of both the hardware and software for each of the instruments had to be completed.

A screenshot of a cell phone

Description generated with very high confidence

Figure Interdependencies amongst tasks

The team was split into four sub team in the initial stages of the project, each sub team consisted of the members that had performed research for that instruments during the initial research phase. The conductor team was constructed of the people that had the most experience with software, since it was a predominantly software-based part of the project. As can be seen in Figure 1, the tasks for each of the instruments were run in parallel, which was possible because the design of no one instrument required input from the other instruments. For example, the keyboard required no input from the xylophone to be constructed. This ensures that if a team member is stalled on one of their instruments (e.g. parts are being delayed) then the whole project does not stall. Instead that person can assist any of the other members on their instruments whilst the parts are being delivered. Some of the tasks for the conductor run in parallel with the construction of the instruments, up until the third main task which is *receiving data from the conductor to the instruments.* This task relies on the software of the individual instruments being completed so that they can be connected to the conductor and the songs can be passed to them. Once this task is complete the next major interdependency is the completion of the conductor and the instruments to the synchronisation task. This task is there to make sure that all the instruments are playing in time with one another and so would require the construction of all four of the instruments and the conductor to be complete. Preparing for demo day relied on this task being completed so that it was possible to determine what needed to be added to the orchestra to deliver an effective presentation.

### Report structure

# Technical Chapters

## Conductor

### Conductor Overview

Part of the specification for the project required the team to build and develop a conductor. The only requirement of the conductor is to synchronise the instruments together. However, the team added additional functionality to allow it to act as a ‘bridge’ between the user and the instruments. The choice of hardware for the conductor was a Raspberry Pi single-board computer, and it was made on account of it being able to run Python scripts, having built-in WiFi and Bluetooth as well as it being small and cheap. Python was the programming language of choice for the conductor, as it allowed the team to quickly develop scripts using the vast collection of libraries available. The following section will explore the various components of the conductor and their implementations. It will follow the order which the user needs to follow to play a track on the orchestra.

A picture containing vector graphics

Description generated with high confidence

**Figure 3.?.** reference tesla

### SDManager

SDManager, is the name given to a collection of Python scripts used to prepare the orchestra to play a track. Initially the user selects a song in a MIDI format and edits it using AnvilStudio to remove/add/modify channels of the track. However, MIDI files are too complex for a microcontroller to work with, and too tedious for the team to work with, so a script called *MIDI2Text* was written which takes as an input a MIDI file, parses it and returns to the user a stripped out version of the MIDI file with only the core information needed for the orchestra to recreate the track. The information returned are three arrays: a time array; with the timing information, a note array containing the note to play/release and finally a status array which tells the user weather to play a note or release it. The script also handles other things such as converting the time from ticks (MIDI notation for time) to seconds and was also used in the development of the instruments as it can return data to the developer such as the number/notes each track uses. Furthermore, the script can separate the MIDI file into separate channels which correspond to each instrument so only the portion of the MIDI file which corresponds to the selected instrument in extracted. MIDI2Text is crucial to the development of the instruments as it saved the team a substantial amount of time and effort which would otherwise had to be put in converting the MIDI file manually.

The team chose to use SD cards to store the various tracks on the microcontrollers since they are a cheap non-volatile memory solution. Another script was developed which takes the output of *MIDI2Text* and transfers it onto the SD card in either a text format or a binary format (depending on how the instrument interprets the data). The script creates a folder with the name of the song, as well as four subfolders, on containing the timing file, note file, status file and one containing some metadata required by the instrument. Although most instruments use the binary format, the text-based format is used in the xylophone and is also useful when checking that the data written is correct. Additionally, another script was developed to allow the user to easily check the contents of the SD card and delete any tracks directly through SDManager.

Once these pieces of software were developed and tested they were merged together, so the user needs to follow five simple steps to take an edited MIDI file and save it onto an SD card in .bin or .txt format for the instrument to play. These steps are:

1. Select path to SD card.
2. Select path to MIDI file.
3. Select .bin or .txt format.
4. Select name to save as.
5. Select instrument.

### Bithoven

The second collection of scripts are used when playing music on the orchestra. The team opted to exploit the wireless capabilities of the Raspberry Pi to create a network of instruments with the conductor acting as a central server. This removes any wiring between the conductor and the instruments resulting in a more professional and refined product. It also given the project an Industry 4.0 feel, something which has been of great interest lately. The conductor can then transmit commands to the instruments wirelessly, and fulfils the requirement of synchronising the instruments.

### Wireless Network

The choice of telecommunication method was narrowed down to three choices: WiFi, Bluetooth and Radio (2.4GHz); the pros and cons will be examined and a decision will be made in this section.

Most of the instruments the team have built are based on microcontrollers rather than computers and therefore do not have built-in WiFi/Bluetooth/RF so the modules have to be bought. The Raspberry Pi has built-in WiFi and Bluetooth but not RF. However, the price for the modules is relatively low (£3-5) compared to the budget so it did not play a major role in the selection. The second aspect the team examined were the libraries and documentations available for each module both for the Raspberry Pi and the various microcontrollers used. All three modules are heavily documented and have libraries available, however, only the WiFi had many protocols for building networks. Bluetooth, is the harder to build a network with and RF does not have a specific protocol for networks. However due to the nature of the modules (nRF24L01+) a network can be easily built by exploiting the various channels available. WiFi, did stand out regarding the availability and simplicity of the protocols for building networks. Moreover, the RPi’s built-in WiFi would speed development and ESP-8266 WiFi modules are cheap and can easily be integrated with any microcontrollers which has UART, more on the WiFi modules can be found …... Due to these reason, WiFi was chosen as a base to build the network on. The protocol used is that called MQTT, which stands for Message Queuing Telemetry Transport. It is a lightweight messaging protocol for small sensors and mobile devices which follows a publish-subscribe messaging protocol and works on top of the TCP/IP protocol. MQTT is becoming an industry standard within the IoT ecosystem due to its simplicity and the fact that it is extremely lightweight and can run on almost all systems. The diagram below depicts an network based on the MQTT protocol.

The RPi has been configured as an access point for the instruments to connect to, this removes the need for an external WiFi network, which might not always be available or secure. Messages in the MQTT protocol contain ‘topics’ which nodes can subscribe to. A node can publish to any topic and any other nodes subscribed to the topic will receive the messages. In more detail, the RPi runs the MQTT ‘Broker’**,** all messages which are published go through the broker, which sorts them and retransmits them to each node which has subscribed to the topic of the message. Each instrument has a topic associated with it and is subscribed to that, furthermore all instruments are subscribed to a BROADCAST topic. The conductor itself is also a node within the RPi; it is subscribed to all the instruments and the broadcast topic and can transmit commands to each instrument individually or to all instruments simultaneously. The IP address of the RPi has been configured as a static IP and the instruments are programmed to connect to that specific IP once they are powered on, allowing them to receive and transmit messages to the RPi.

The conductor can then publish one of three commands to the instruments. A PLAY command followed by a song name causes the receiving instrument to download the song from the SD card and play it once it receives a START condition from the conductor. A STOP command from the conductor cause the instrument to halt any song that is currently playing, this allows the user to play another song or simply stop the song. Finally, a LISTtracks command causes the instrument to read the tracks stored in the SD card and returns them to the RPi, this allows the user to see which songs have been saved without having to remove the SD card and putting it in a computer. Since all the configuration for the MQTT protocol has already implemented, adding additional features mean they only need to be implemented on the microcontroller, allowing the developer to expand the functionality without prior knowledge of the protocol.

Add mqtt diagram/code snippets etc

### Conclusion

In summary, the conductor contains various scripts to handle any interaction the user might need with the conductor. It allows the user to easily convert MIDI files and load them onto an SD card without any knowledge of the MIDI protocol, and in a format which is easily read by any microcontroller and computer. It can also be used to read the contents of an SD card and delete tracks easily. Since the conductor was designed with the intention to add additional instruments to the orchestra it can also be very useful for developers to obtain various information about the songs their instrument will need to play. The network side of the Conductor makes remote interaction with the instruments simple. The MQTT protocol proved to be an excellent choice for the network, and keeping with the theme of modularity, adding a new instrument to the network requires very little work from the developer. To wrap up this collection of scripts into a neat and easy to use way, a graphical user interface (GUI) was developed to run on the RPi with the goal of simplicity in mind. The GUI in combination with a touchscreen connected to the Pi allows the user to have a fluid experience with the orchestra.

## Software Structure

## Xylophone

### Xylophone hardware

### Solenoid Software

#### Concept

The concept of the xylophone software, is to have three text files from each song, generated by the conductor, in order to define not only the physical condition, but also the rhythm, of the solenoids; i.e. which solenoid needs to be turned on and off at a specific time and play a particular note. Therefore, the first text file contains the notes that need to be played for a specific song, the second text file determines the time delays between the notes that are being played and the final text file, defines the state of the solenoid; either on or off.

#### How it works

To begin with, the three text files for each song, which are generated by the conductor, are saved in a file directory inside the MyRIO.

The WIFI module produces a number of different commands (“PLAY”, “STOP”), along with the song that needs to be played. In order to make these commands interact with the xylophone, the UART of the MyRIO is used. Therefore, the transmitter of the WIFI module is connected to the receiver of the UART of the MyRIO.

In total, two while loops are used; one which contains the UART and thus continuously “reads” the commands received from the WIFI module, and the second loop which contains the generation of the status, notes and time delays. Emphasising on the second while loop, a case statement exists which executes according to the command that is read by the UART. If the command is “STOP”, then there are no text files loaded in the program and therefore inertia prevails. On the other hand, if the command is “PLAY”, then the appropriate text files are loaded from the c file directory of the MyRIO. Three arrays begin to form; one containing the time delays between the notes in microseconds, a second one containing the notes of the song that need to be played and finally the third one, containing the state in which the solenoids need to be, i.e. on or off (add image).

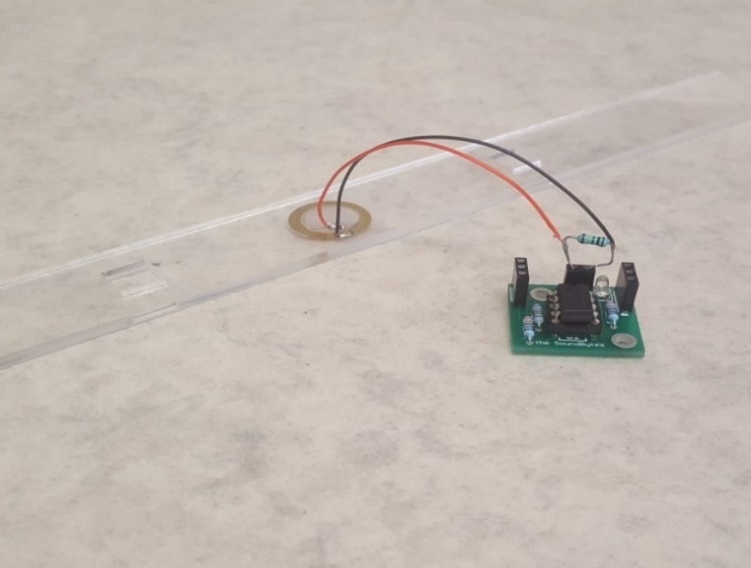
The next step of the procedure is to minimise the number of notes used in the songs, since the xylophone has only 12 notes. In order to do so, a formula node is used, in which the preferred octave is selected and all the notes (from that octave) are used. For instance, the Game of Thrones theme song, which has 19 notes from three consecutive octaves (octave 2,3 and 4), is configured to have 12 notes all of them from the same octave. The appropriate octave, is selected either according to the most frequent notes that are being used within the song, or in the case of having three consecutive octaves, the middle one is selected in order to minimise the deviation from the original song. Finally, 12 digital output ports are used from the MyRIO in order to control the solenoids of the xylophone.

For that reason, a for loop is used, which runs that many times as the number of notes in each song. Inside the for loop, there is a case statement which reads the status text file and acts accordingly. In more detail, if the status is 0, the digital output pins are not enabled and therefore the solenoids are turned off, whereas when the status is 1, the pins are enabled and according to the note array, the appropriate solenoid turns on for an amount of time which is determined by the time array (enter pic). However, the for loop has a conditional terminal; when the UART reads the value “STOP”, then the loop is forced to stop executing and start again when the command send to the UART (from the WIFI module) is “PLAY”.

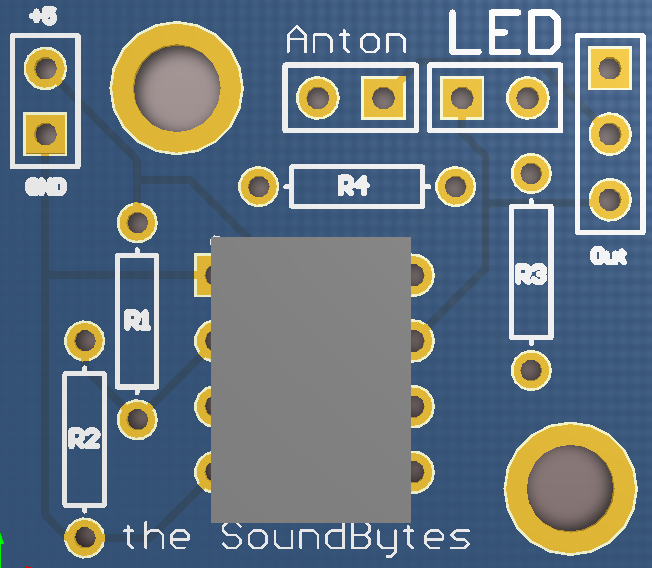
### Piezo sensors

Piezo sensors were put on top of the xylophone keys to detect the solenoids hitting the keys. They were connected to a comparator circuit that compared the voltage from the piezo sensor and the reference voltage of the voltage divider at the non-inverting input. The output of the comparator circuit was connected to the MyRIO so whenever the output was high, the LabView program on the MyRIO played the corresponding note through a speaker. The MyRIO was the preferred microcontroller since it already had a built-in audio output.

**Figure 1.1a.** Piezo sensors on the keys connected to the comparator



**Figure 1.1b.** Comparator PCB



**Comparator circuit**

After making the initial comparator circuit using a LM741[1] and a voltage divider, several values of resistors were tested and it was found that a sensitive reference voltage was found to be around 0.2V that would detect the impact of the solenoid upon the piezo sensor. An LED was also placed at the output as an indicator. This circuit is seen in Figure 1.2a

R1 = 8000 Ohms and R2 = 330 Ohms.

Where V is the supply voltage (5 V) there for .

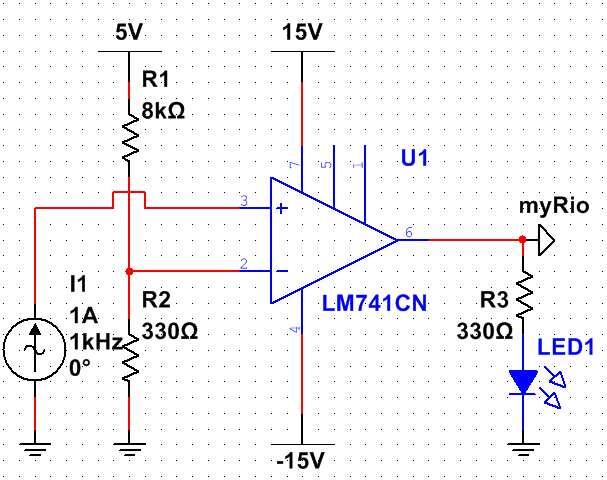
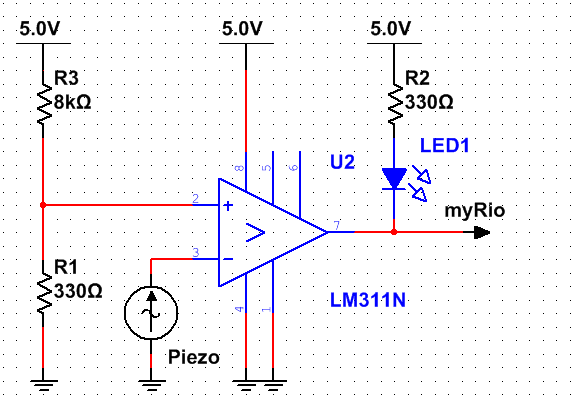
The circuit worked very similar to the final PCB and circuit design shown in Figure 1.1b and 1.2, the only important difference being that the LM741 is powered by -15 and +5 volts whereas the LM311 can be powered by Ground and +5 volts. There were 12 of these circuit connected to each other and to the 12 xylophone keys and in turn connect to 12 individual inputs on the MyRIO.

After tests were done, there were problems with the piezo sensors when they were used for around 20 seconds, the LEDs indicated that the comparators stopped responding to the solenoid hits. This was fixed by adding a resistor in parallel with the piezo sensor to get rid of any build-up of static charge. Later it was noticed that there was -15 V at the output of the comparator when it was in the off state, which could damage the MyRIO, therefore the circuit was slightly modified and a new comparator was added: LM311. This comparator could function of off ground as negative voltage supply so the danger of damaging the MyRIO was no longer there. The new circuit can be seen in Figure 1.2b.

This leads to the LabView code that was used on the MyRIO to play the different sounds for each individual key.

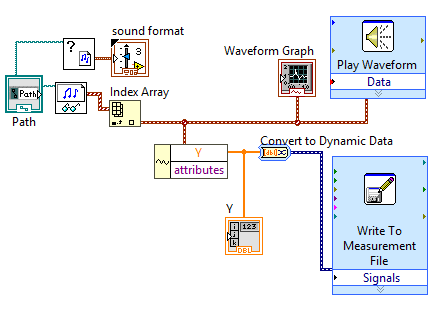
**LabView code**

**Figure 2.2b.** LM311 comparator circuit

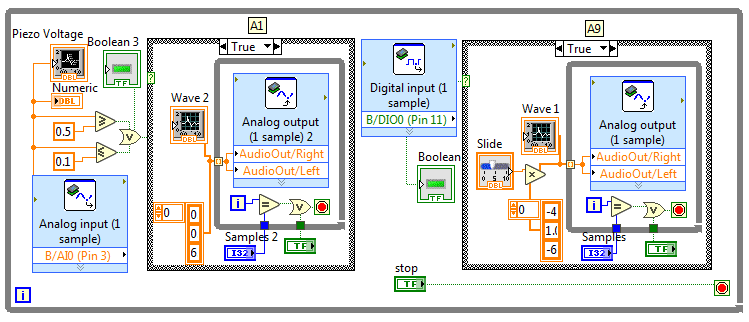


**Figure 1.2a.** LM741 comparator circuit

The code to play the different notes was made in LabView. The MyRIO cannot directly output a wav. or mp4 format files to the audio output therefore the wav. files of the xylophone notes which were obtained from soundpacks.com[2], were converted to a waveform and stored as an 1D array Y seen in Figure 1.3. This was all done offline since the MyRIO cannot do the converting in real-time according to this forum post [3], which was helpful in the development of the converter.



**Figure 1.3.** Converting wav. files to waveforms with help from [3].



**Figure 1.4.** LabView code for analog and digital piezo inputs

This array was then converted to a constant and moved to the main program seen in Figure 1.4.

This code looked at the input from the pin to which the comparator was connected to and whenever it was high, it played the waveform from the 1D array through the audio output. The waveform was played at 44,400 samples a second, the number of samples played could be set to any number and the volume was changed by multiplying the amplitude of the waveform. The front panel (interface) for testing can be seen in Appendix A.

This code was then expanded to 12 inputs and 12 waveforms for all the xylophone keys. The real-time module of the MyRIO was then used to store the code onboard and execute it every time the MyRIO was started up.

**Speaker**

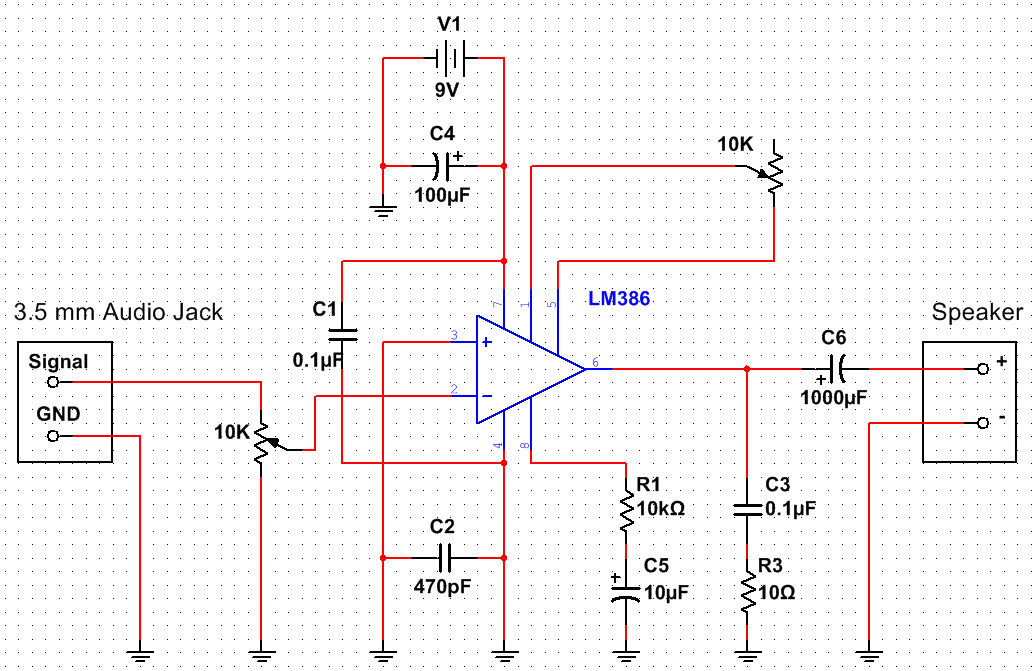
A speaker which would play the xylophone notes was installed and an amplifier circuit connected it to the MyRIO output. This circuit allowed to tune the volume and gain of the speaker since it did not come with any inbuilt controls.



**Figure 1.5.** Visaton Speaker [5] for the xylophone.

The datasheet [5] stated that the speaker was capable of outputting 86 dB between 80 and 20000 Hz frequencies which was loud enough to play with the rest of the orchestra. The issue was that the signal from the MyRIO needed to be amplified.

An amplifier circuit was built following the instructions from [6]. This circuit allowed the user to control both volume and the gain of the signal using potentiometers. This circuit can be seen below:



**Figure 1.6.** Amplifier circuit with volume and gain controls [6].

This circuit used the LM386 chip for amplification. A 9 V battery was used as the power supply, but a MyRIO +15 V and ground could also be easily used as the LM386N-4 [7] chip can support up to 22 V as a power supply. The designed PCB can be found in Appendix A.

This circuit was later used to amplify the microphone input for the Tesla coil setup, but it was later found out that the circuit could pick up the signal from the Tesla coil wirelessly, without a microphone.

### Xylophone Testing

### Summary

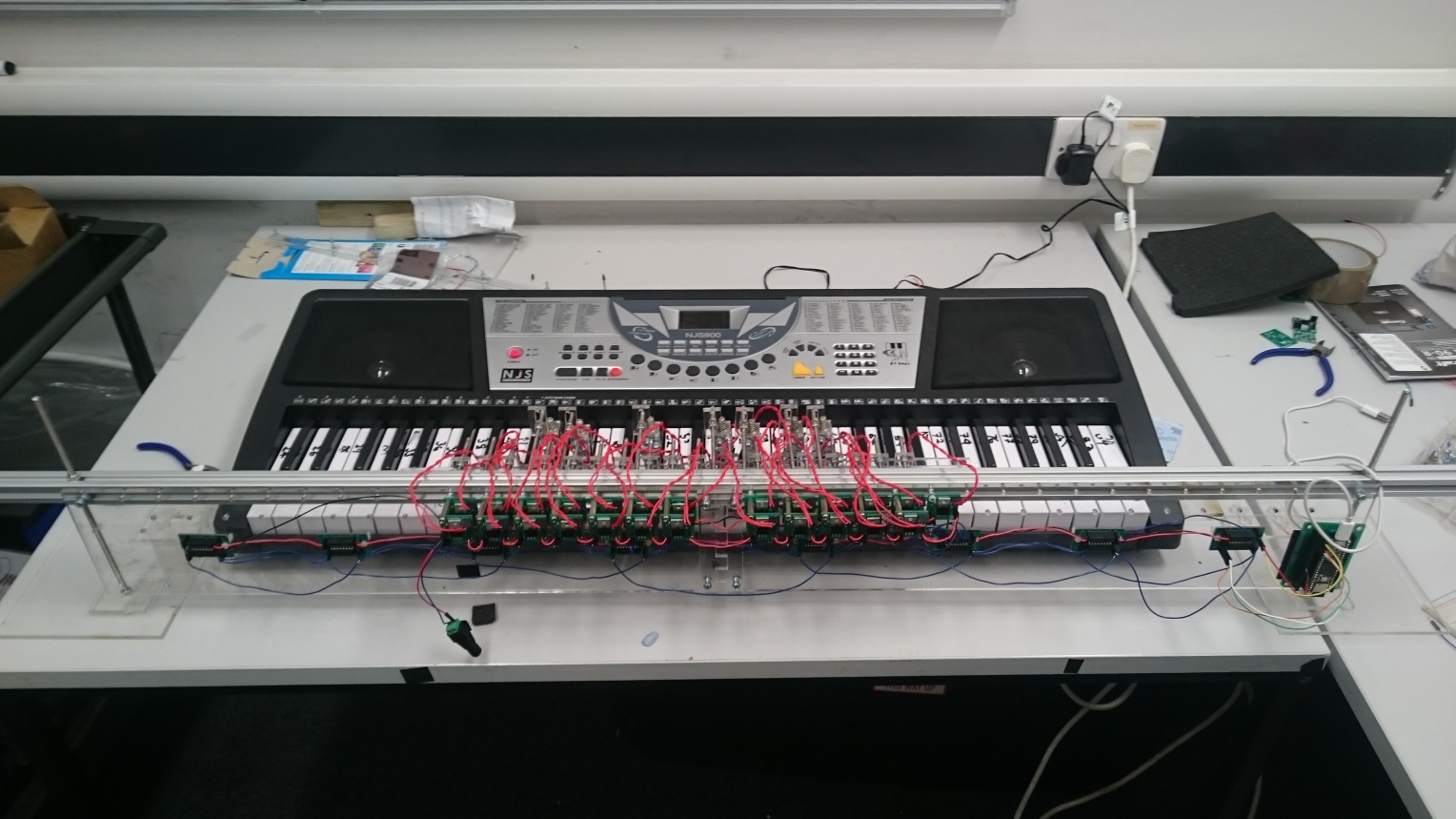
## Keyboard

The keyboard was chosen as an instrument due to flexibility it provides the orchestra to play a wide range of notes. The following sections will first summarise the main components of the keyboard and then describe the design decisions made in the design. To meet the aims of the project, the keyboard needs to meet the following specification:

* play the tracks chosen for the keyboard from the three selected pieces of music. (objective ?)
* communicate with the conductor so it can be controlled alongside the other instruments (objective 4)
* be transportable (objective 3)

### Overall design description (reference literature review)

Figure 2.1 shows the keyboard Design. The robot keyboard design uses solenoids to press the keys and a Teensy microcontroller board to control the instrument. The solenoids are supported by a Bosch bar and are connected to the Bosch bar using threaded rods, which are split into two different lengths: 60mm and 105mm for the white and black keys respectively. The solenoids attach to the threaded rod using a bracket made from clear Perspex, allowing the solenoid to be positioned at any point along the rod and allow it to be adjusted vertically by 25mm. The Bosch bar has a threaded rod at each end to support it, so its position above the keyboard is adjustable. The PCBs to control the solenoids are mounted on a Perspex plate that is attached to the Bosch bar. There are only 25 solenoids set up these cover the notes required for the three chosen test tracks (see appendix??? for table). This decision was made to keep in the project budget of £1500 as each solenoid costs £10.40 so to have 25 is a cost of £260. It would have cost £634.40 to outfit the entire keyboard, which would have been about 42% of the projects budget. The instrument has been designed, so in the future more solenoids to cover the full range of notes can be easily added. For the time being, if when expanding the range of songs to be played by the orchestra the notes required are outside the range of the solenoids set up, the notes can be shifted by an octave into the notes played by the keyboard without too much effect on the sound of the music. The design can be dismantled the threaded rods unscrew from the Bosch bar and the base plates meaning it can be transported in a … box.



Threaded rod supports with base plate

Shift register PCB

Solenoid switching Transistor PCB

Solenoids and brackets

Bosch Bar

Teensy Embedded system

**Figure 4.?.** Keyboard instrument.

### Solenoid Selection

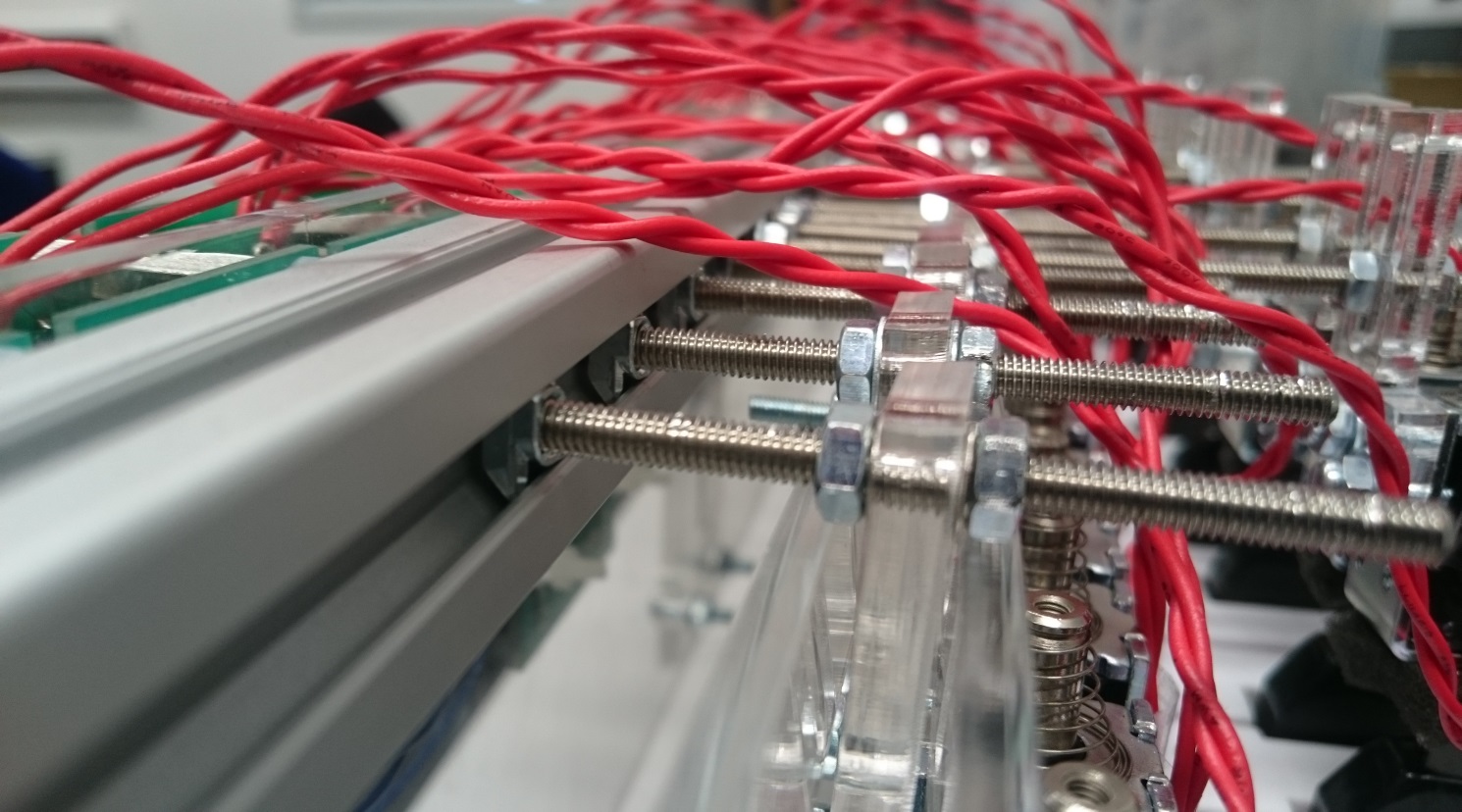
Firstly, a test was carried out to find the force needed to cause the keys to play a note the table is given in Appendix. A mass of 70g is required to press down allowing using a safety factor of 25% the keys need a force of 1.5\*0.070g\*9.81m/s^2=1.03N=105gf to be pressed. The distance the key needs to be depressed is 3mm so the solenoid throw needs to be at least 3mm. In the music chosen up to 4 keys can be played at the same time so the solenoid also needs a low current demand so they can be powered using an off the shelf plug. The 12V 3W SD0630 fulfils these requirements it can provide 120gf with a duty cycle of 50% (none of the songs require any one note to be depressed 50% of the time (table to prove)). It also has a throw of 10mm and has a current requirement of 0.25A so 8 could be pressed at once using a 12V 2A supply (datasheet in Appendix…).

One solenoid was bought to test. The solenoid came without a stopper so one was printed in the mechanical workshop with the design shown in Appendix this was a allows the solenoid throw to be adjusted so it was decided this was a positive feature. The solenoid was tested and it was able to press the white and black keys. The main issue with the solenoid was the clicking made when the solenoid switches on and off. This was partially solved by putting foam between the stopper and the solenoid as shown in figure.

Another clicking sound was made when the plunger hits the inside of the solenoid. An attempt was made to lessen this as well by putting a small piece of foam on the plunder figure?. However, this caused issues as it added extra resistance to the working of the solenoid so, it re-acted slower when turning on and caused different delays between each solenoid. It was decided this would cause issues with playing the songs and synchronisation of the instruments so the foam was removed. The volume of the keyboard can be turned up to a level that mitigates the sound of the clicking and the clicking dose give the instrument a mechanical sound which actually fits in with the aesthetic of the orchestra.

### Supporting the Solenoids

To make the construction of the rail supporting the solenoids flexible a Bosch bar was used as it allows the solenoids to be placed at any point along its length. This is achieved using T-slot nuts to secure the threaded rod in the Bosch bar to which the solenoids can be attached. The threaded rods will be 4mm in diameter as the T-slot nuts are M4.



T-slot nut

Threaded rod

Bracket

solenoid

Bosch Bar

**Figure 3.?.** Close view of Bosch bar structure.

A bracket was required to connect the solenoid to the threaded rod. This needed to be easily adjustable so the solenoid could be put into the correct position. It also had to be limited in width as the keyboard is 726mm long with 36 white keys which leaves 20.16mm per key for the bracket. However, space for a 4mm rod also has to be left for rods to pass through to the black keys leaving 16mm. The width of the solenoids is …..

The design went through several iterations. The first design is shown in figure it allows for vertical movement but not movement along the length threaded rods therefore, the rods would have to be made to the exact length and it wouldn’t be adjustable. The next design is in figure this allows for horizontal and vertical movement (25mm) the design is stepped so the threaded rod doesn’t interfere with the solenoid plunger. The third design just adjusted the distance between the step so there was a 2mmm difference between each side which makes it easier to overlap them in the mounting process.

### Solenoid PCB design

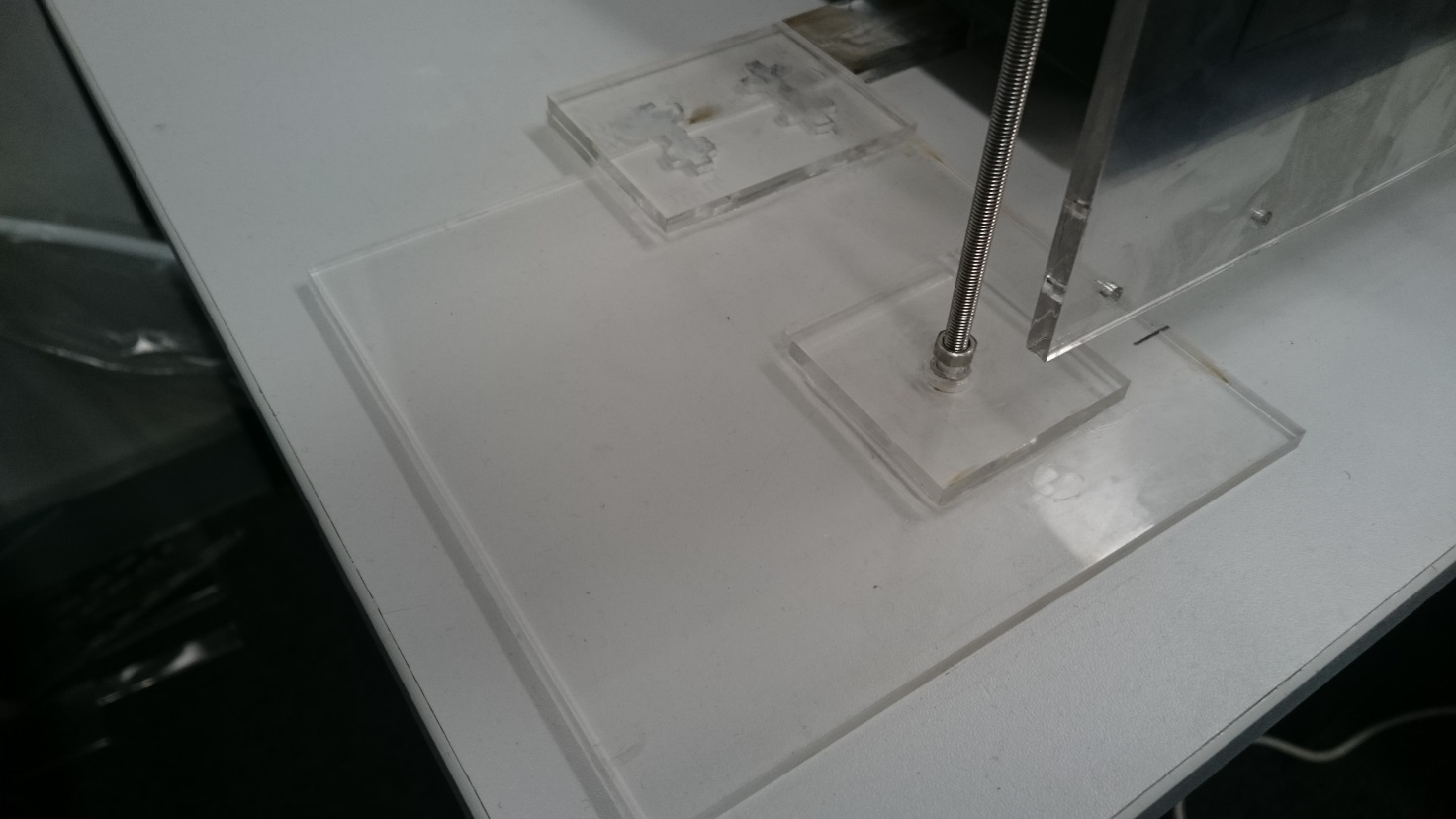
A transistor PCB was designed to turn the solenoid on and off. The required current for the solenoid is 0.25A. The instrument uses shift registers to deliver all the notes to the keyboard at the same time. The shift registers ate powered using one pin from the Teensy which can provide 25mA. If a worse case of 8 keys pressed at the same time is taken the current is 25/8=3.125mA. therefore, the minimum gain required from the transistor is 0.25/3mA=83.3. It was decided to use the same transistor for the xylophone solenoids so only one type of transistor needed to be ordered and the same circuit could be used for both. The xylophone solenoid requires 2A and the MyRIO can provide 150mA per digital output so the gain needed is 13.3. The decision was made to use the BJT, TIP120 which can handle 60V 5A and can provide a gain of 1000. This resistor required for the keyboard to provide the gain is ?? the circuit and PCB design are shown in figure.

### Supporting the Bosch Bar

It was decided to support the Bosch bar using two threaded rods located at either end as shown in Figure 2.1 which allows the solenoid rail to be adjusted vertically. The threaded rods were set at 230mm long this allows for the key height of 70mm, the 20mm thickness of the Bosch bar, 80mm for the solenoid and leaving 70mm to allow the rail to be moved upwards and remain supported while, the keyboard is removed from under the solenoid rail. A 5mm threaded rod was chosen as the Bosch bar has a 5mm gap in its design to attach the thread rod through.

A base plate was needed to hold the threaded rods vertically. It was decided to use Perspex for the various brackets and supporting plates for the instrument. So, to keep with the aesthetics of the design Clear Perspex was used for the support. The design is shown in figure and uses a two Perspex pieces one for which the threaded rod is attached using two nut which clamp the Perspex tight to the threaded rod. The second piece is glued to the other so the holes overlap allowing the lower nut clamping the other plate to be counter sunk into the base so the base can be flush with the surface its placed on. An M5 nut is 3.5mm thick and 9mm wide so the Perspex needs to be thicker than 3.5 and the hole needs to be at least 10mm. The laser cutter in the workshop can only cut 1cm thick Perspex and 5mm was the maximum thickness the workshop has in stock. So, it was decided to buy a 600mm by 600mm sheet of Perspex from the workshop for £40 as it could be used for all the other Perspex laser cutting needed. There is a channel running the length of the keyboard in which a bar can be run to connect the two Perspex stands together this is … away from the edge of the keyboard and the stands need to be located … away from the keyboard. So the larger base plate for the nut to be counter sunk into was made 150mm by 150mm with the centre of the 10mm hole 50mm in from each edge. The top plate which clamps the rod in place was 60x60mm with a 5mm hole in the centre.

To remove the chance of the supports moving while the keyboard is playing a bar connecting the two stands was designed. To do this a channel that runs the length of the keyboard was utilised. The channel is 29mm wide and 20mm high. A Perspex rod was designed to run along it to attach to the two base plates at either end. The design used two lengths of Perspex each 29mmx550mm long that were attached in the middle using a 29x60mm plate to overlap the two pieces so they could be glued together. It was done in two sections, as the laser cutter cannot cut pieces longer than 600mmm long. The bar was then attached to the two supports using a overlapping ….x…. plate at either end to glue the pieces together. Two stopper pieces of perspex were made to stop the keyboard sliding along the bar the design is shown in figure. This means the solenoid rail is held in place by the keyboard.



Large Base plate

Plate clamped to threaded rod with two nuts

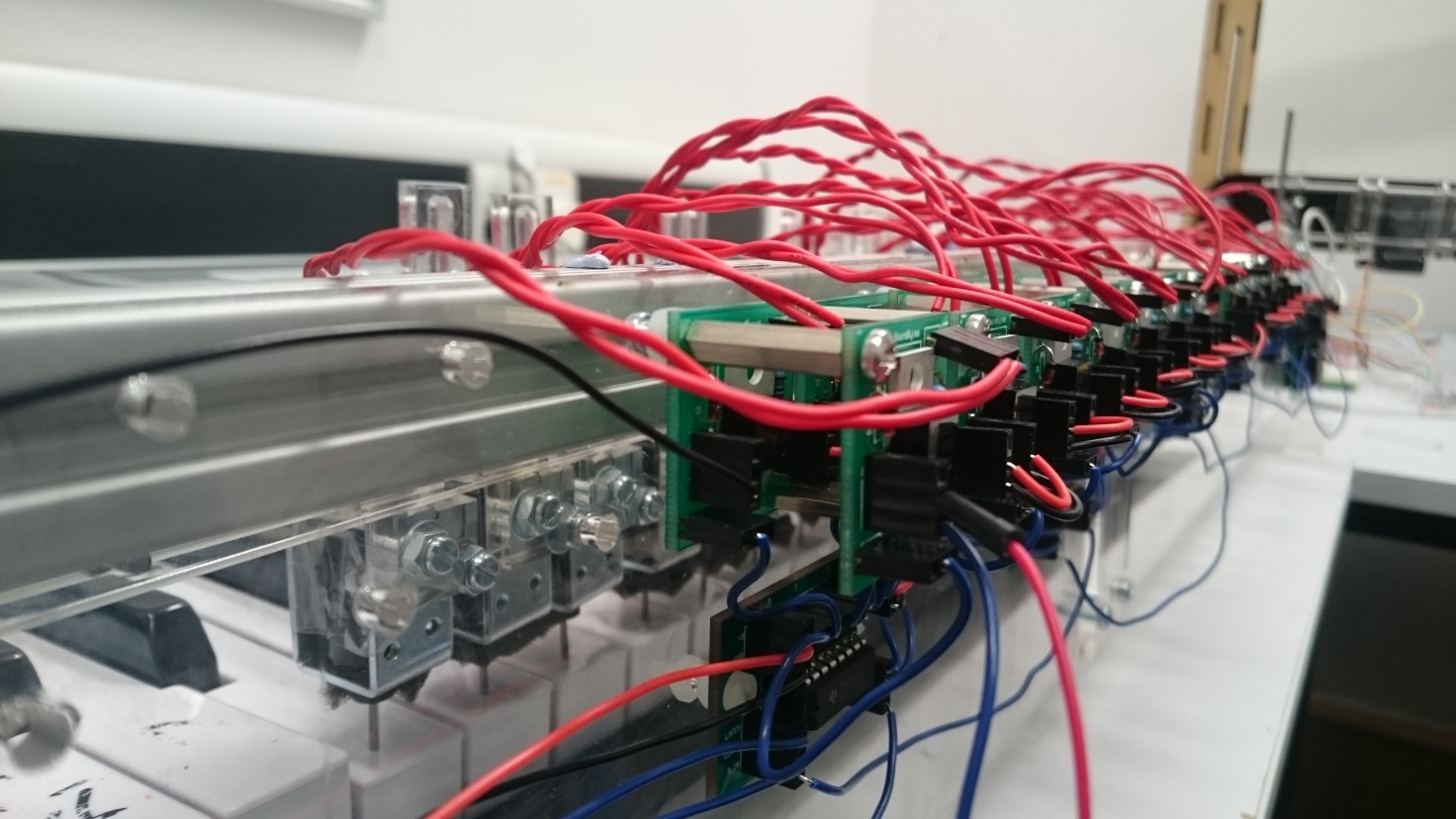
Connection to the Perspex bar that connects both feet each end of the Bosch bar

Threaded rod

**Figure 3.?.** Base of the Keyboard.

### PCB Support Plate

There are 8 shift registers, 25 transistor and 1 Teensy PCB that need to be mounted on a perspec plate. The keyboard instrument is designed is so that it can be expanded to have solenoids for all the keys so space has been left on the plate for more transistor PCBs . The plate can be ...long and …. Wide. To do this in the required space the transistor PCBs have to be mounted as a double layer as shown in figure. They are mounted at the top of the plate so the solenoids leads which are … long can reach the PCBs. The shift registers are mounted below these in the middle of 8 solenoids so they can be easily wired up and the teensy is located at the end as it only needs wiring to the first shift register. The design for the plate is in Appendix 1.



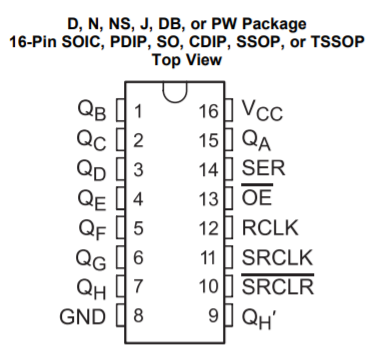
**Figure 3.?.** Transistor circuits.

### Keyboard software

The keyboard is driven by a Teensy 3.5 [1] connected to eight SN74HC595N shift registers [2]. Initially, an Arduino Uno was going to be used to drive the keyboard however it was found that the Arduino did not have enough on-board memory to hold the large arrays needed to describe the note, status and timings of a song. For example, the files needed for *Californication* amounted to 84 kB whereas the Arduino only has 32 kB flash memory. The Arduino was replaced with a Teensy 3.5 which has 512 kB, more than enough to store the files for either of the songs needed for the demonstration.

The keyboard software was designed with the ability to expand in mind. The total available keys on the keyboard is 61, but only 25 will be needed to play *Eye of the Tiger* and *Californication*. For this project, only 25 solenoids will be implemented so that the two songs can be played. However, if the project is to be expanded on in the future then the ability to add more solenoids relatively easily needs to be available. Therefore, wiring the solenoids directly to the Teensy 3.5 would not work, since it only has 42 IO pins [3] and 61 need to be available should all the keys be needed to play a song. SN74HC595 shift registers were used to add 64 IO pins to the Teensy 3.5. In addition to this, a 64-bit variable was used where each bit represents one of the keys.

### SN74HC595 Shift Registers

There are two main forms of shift registers, the SN74HC595 is an example of a serial in-parallel out shift register.

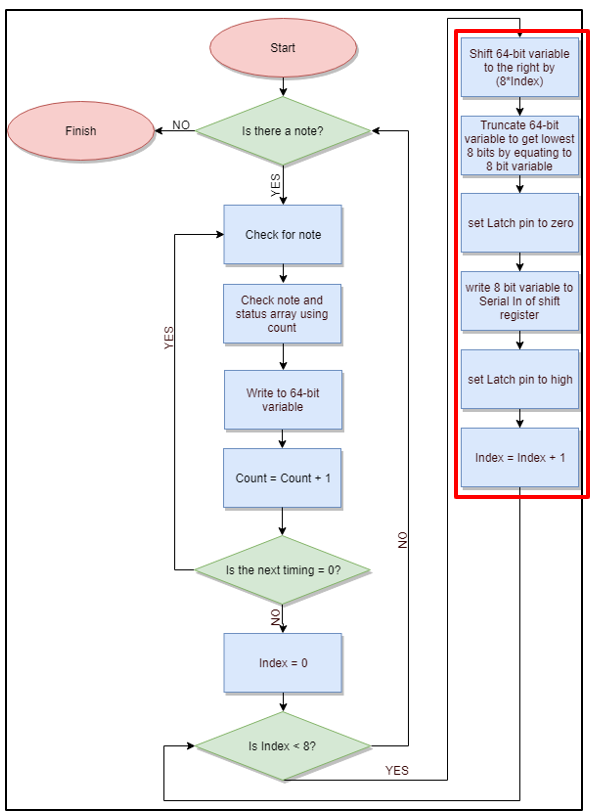
**Figure 1.** Schematic for SN74HC595 shift register from Texas Instruments [2].

From Figure 1, there is one *Serial Input* pin located on pin 14 and there are eight output pins located on pins 1-7 (QB to QH) and 15 (QA). To control the shift register, an 8-bit variable is sent, bit by bit, to pin 14 which writes each bit to the corresponding output pins starting at pin 15 (Q­A) and finishing on pin 7 (QH). In this way an 8 bit variable can be written to the shift register. Once this writing process has been completed, pin 12 (latch pin – RCLK) is set high and all the output pins are written to their connections at once.

The feature that makes it suited to the keyboard is that several shift registers can be connected in series with one another to act as one large shift register, essentially increasing the number of outputs in multiples of eight. To cover the full 61 notes available on the keyboard, 8 shift registers will be needed, with each pin on the chain of shift registers representing one of the notes on the keyboard. This means that, since the solenoids are fixed in position, they will need to be connected to specific pins on the chain of shift registers. The benefit of this is that it is simple to continuously add solenoids to the keyboard so that all the notes are available to play without making any change in the software.

### Structure of the program

To test the shift registers, a circuit consisting of 16 LEDs and two shift registers was constructed. The cathode of each of the LEDs was connected to ground and the anode of each LED was connected to a pin on one of the shift registers. From the testing it was found that, even though there were two shift registers connected in series (providing 16 output pins) it was only possible to write an 8-bit variable to the registers. This means that to write a 16 bit variable and display them on the shift registers, it would have to be split in to two 8 bit variables and write them one after the other. So, for use on the keyboard eight 8 bit variables would need to be written to the shift registers to cover the full 61 keys. As mentioned previously, a 64-bit variable was used where each bit of the variable represents one of the notes from the keyboard. For example, bit 0 of the variable represents the note with MIDI number 24 (which is the lowest note on the keyboard).



**Figure 5.** Structure of keyboard software that occurs within an Interrupt

Figure 2 above shows the structure of the software for the keyboard. Once the program has started the *Count* variable is used to check whether there is a note available in the note array found on the SD card of the Teensy 3.5. Since the lowest MIDI note supported by the keyboard is 24 and the 64-bit variable starts at bit 0, then 24 needs to be subtracted from the note number obtained from the note array to get a corresponding bit number. For example, if note 54 needs to be played then this corresponds to bit 30 of the 64 bit variable. Once the note number has been achieved the *Count* variable is used again to determine the status of this note from the status array. Using the note and its corresponding status, the 64 bit variable can be updated to represent the current status of the song.

To update the 64 bit variable, the bitWrite() [4] function found in the Arduino library is used. The bitWrite() function takes three variables. First, the variable which will have its bit changes (32 bit), the number of the bit to change and the status to be written to that bit. As can be seen, this function only works with a 32 bit variable but the keyboard uses a 64 bit variable, so the functions had to be redefined in order to make them work with a 64 bit variable. This was done by redefining the bitWrite() function so that it works with an unsigned long variable as opposed to just an unsigned long variable. This makes it so that the bitWrite function can now accept and modify a 64 bit variable.

Once the 64 bit variable has been modified the next stage of the program checks whether there is another change to the notes at this particular time. As the software is controlling a keyboard, a possible of 16 changes are possible at any one time (8 notes turning off, 8 turning on). A simultaneous change can be detected by checking whether the next element in the timing array is a zero. If it is a zero, it means that the change in note is happening with the previous change in notes before progressing. To detect whether these changes are happening at once the program enters a while loop, this increments the *Count* variable and checks the corresponding element in the time array to see whether it is zero. With each loop, it updates the 64 bit variable using the bitWrite() function. Once the loop is complete, the program updates the interrupt timer to the value of the non-zero element in the time array.

Once the next non-zero element is found in the time array this indicates that all the changes to the notes have been for this particular time in the song. The next stage is to take the 64 bit variable, break into eight 8 variables and right these to the shift register. This is done by first setting the latch pin (pin 12) of the shift register and *Index* variable to zero and then entering a for loop. Within the for loop (shown in red on Figure 2) the 64 bit variable is shifted by (8\**Index*) and is truncated by equating it to an 8 bit variable. This truncated version is then written to the *Serial In* (pin 14) of the shift register. This is repeated eight times, each time incrementing the *Index* variable by one. In this way, the entire 64 bit variable is written to the shift registers 8 bits at a time. Once this is complete, the latch pin is set to one which outputs the bits on the shift registers to the solenoids. Since it does this in parallel all the changes at that particular instant in the song are made at once. Once the latch pin is driven high, the program waits for the start of the next interrupt and starts again.

### Keyboard Testing

To test whether the keyboard hardware and software had been built and constructed correctly several tests were run. The first test involved creating a program where the user could specific a specific key and get that single key to press down on the keyboard. This was a way of testing whether each of the solenoids fixed to the Bosch bar can press down and producing a note on the keyboard. The test results showed that each of the solenoids was capable of producing a note, the test results can be seen in **Appendix.** This test showed that each of the solenoids can do so when played individually.

In addition to this test, the *Game of Thrones* theme song was played on the keyboard and timed to see how long it would take to complete. This was compared with the MIDI file version to show in both cases the song completed playing after 1 minute and 24 seconds. Both of these tests combined showed that the keyboard, when being played on its own, was performing as expected.

### Summary

The keyboard software utilises a chain of eight shift registers, totalling 64 output pins to cover the 62 keys available on the keyboard. Each of the pins represents a single note of the keyboard. A while loop is used to modify the 64 bit variable to represent the current status of the song. Once this has been done, a for loop is used to break the 64 bit variable into eight 8 bit variables that are written to the shift registers one by one.

## Stepper Motors

### Stepper Motor Hardware

The stepper motor instrument is controlled by a Teensy 3.5 board. Similar to the keyboard, the instrument was initially designed to be used with an Arduino Uno but as the Arduino board did not have enough memory to store the data required for playing an entire song, the design was adapted for the more suitable Teensy board.

This microcontroller board was used to program a AD9837 Digital Synthesizer (DSS) IC [1] to generate the required frequency for each note to be played as the song progressed. The frequency generated by the DSS chip was then used together with a DRV8825 motor driver board [2] to control the rotation of the stepper motor. The DRV8825 was selected because of its high current rating of 2.5 A that matched the stepper motor current requirement of 1.5A [3]. Additionally, the driver board also had a current limiting potentiometer which allowed for each motor to be tuned individually so that each motor would produce a clear sound. To generate the musical notes, NEMA 17 stepper motors were used as they were rated at 12V and 1.5A [3] which allowed for easy interfacing with the DRV8825 motor driver board. Additionally, as the stepper motors are not loud when rotating, a wooden acoustic box was built for them to be mounted on so that the sound produced by the motors would be amplified and easier to hear in larger spaces such as those that are used for trade fairs and other expo events.

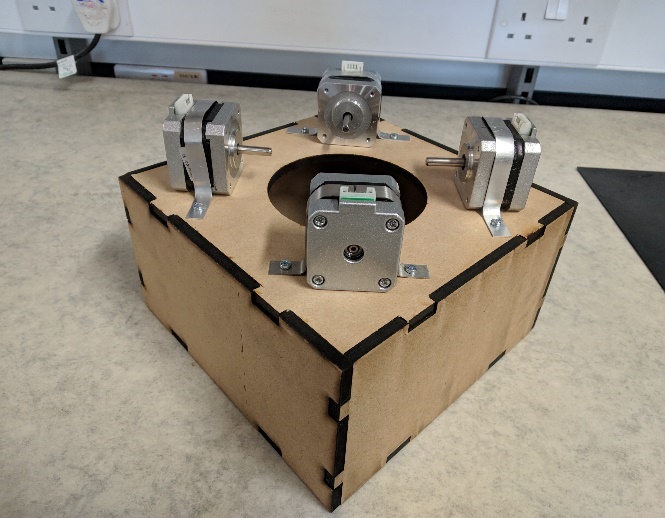
### Acoustic box

Due to the fact that the stepper motors do not produce loud sounds when rotating and as the instrument is likely to be played in crowded areas, a decision was made to amplify the sound produced by the motors by using an acoustic box. After testing the amplification level produced with materials such as Perspex, wood and cardboard, the sound measurements presented in Table 1 have been obtained. As it can be observed, wood produced the highest sound amplification and as a result, the designed acoustic box was built using this material, acting similar to the acoustic box used by guitars to amplify the sound produced by the strings vibrations. The motors are held in place by using aluminium brackets fixed on the top side of the box. The designed box is presented in Figure 1.

|  |  |
| --- | --- |
| **Material Used** | **Measured Sound Level (dB)** |
| Perspex |  |
| Wood |  |
| Cardboard |  |

**Table 1.1.** Sound Level measurements for different materials

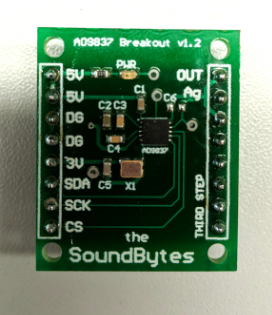
The acoustic box also has attached to it a PCB that contains all the components required for the operation of the stepper motor instrument: a Teensy board, four DSS circuit boards, four motor driver boards and a WiFi module for connecting to the conductor. The PCB that is mounted on the acoustic box was designed in a way that allows for each component to be easily replaced. This was done in order to minimise the impact on the instrument in the situation in which one component might fail and to also reduce the time it would take to fix the instrument in such a situation. Additionally. The slots on the PCB dedicated to the motor driver board are also compatible with other motor driver boards such as the Allegro A4988 which mitigates the risk of some components being difficult to source.



**Figure 1** Stepper Motors Acoustic Box

### Digital Signal Synthesizer

The selected Digital Signal Synthesizer (DSS) chip was the AD9837. This was due to its high accuracy, having a resolution of 0.02 Hz and being able to produce frequencies between 0-3 MHz [1]. As the frequencies in the audible range only go up to 20 kHz [4], the DSS chip was well suited for this application as it was able to produce the frequencies of all musical notes with high accuracy, meaning that the replicated songs sounded as close as possible to the original versions.

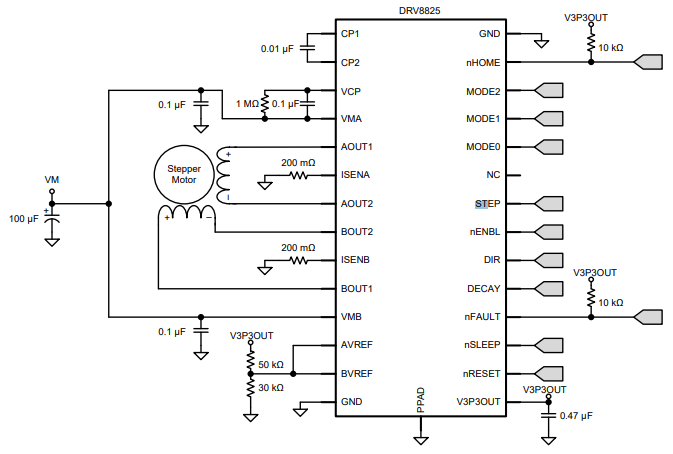


**Figure 2** Digital Signal Synthesizer PCB

Figure 2 presents the PCB designed for the DSS to be used with the other circuit units. As the IC was only available as a surface-mount component, particular care had to be paid when placing the components on the PCB.

### Motor Driver Board

In order to ensure that the stepper motors receive enough current, a motor driver board was used. For the stepper motor instrument, the selected model was the Texas Instruments DRV8825 [3]. This is because this model of driver board was recommended by multiple sources and according to [3], it was able to operate with voltages between 8.2V and 45V while supplying a current of up to 2.5 A, which was suitable for use with a NEMA 17 stepper motor.



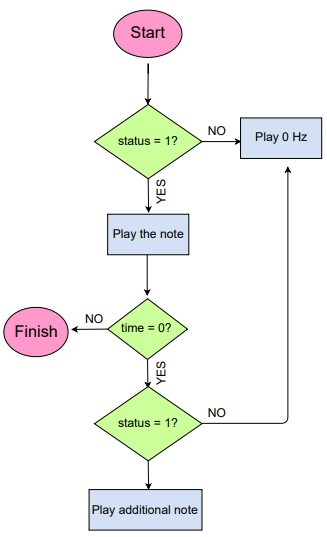
**Figure 3** Schematic for DRV8825 IC [2]

Figure 3 presents the DRV8825 pinout and its required connections. The STEP pin is used for controlling the movement of the motor. Each pulse sent to this pin is converted to a step rotation and the DIR pin is used for controlling the direction in which the motor spins. Additionally, a decoupling 100 µF capacitor has been used as indicated in Figure 3 in order to protect the IC against voltage irregularities. In setting up the control circuit, the wiring diagram presented in [5] has been used for ensuring the wiring was done properly so as not to damage the components.

### Stepper Motor Software

The stepper motor instrument relies on the SD card of the Teensy board to play a particular song. This contains the arrays with the information required to play a song: note (contains the frequencies to be played), time (contains the time until the next action) and status (determines whether a note should be played or stopped).

Figure 3 presents the code execution flowchart of the stepper motor instrument that occurs every time the ISR is executed. The note status has a value of either 0 or 1 which determines whether a note should be played (status is 1) or muted (status is 0). Additionally, a value of zero in the time array indicates that multiple actions have to be executed at the same time. The instrument’s code uses a function named “playnote” in order to play a specific musical note on a stepper motor. This function takes a frequency value as a parameter and encodes it onto the DSS chip which generated the corresponding signal which is further played by the stepper motor. By using this sequence with four stepper motors and four DSS ICs, the instrument can play up to four musical notes at the same time, thus increasing the number of songs that it can replicate.



**Figure 3** Flowchart of stepper motor software

### Stepper motor Testing

### Summary

The stepper motor instrument uses four stepper motors to play a song, meaning that it can support up to four musical notes being played at the same time. As the DSS chip can produce all the frequencies in the audible spectrum, the instrument is able to produce any musical notes that are required for a song, being a versatile instrument that can play both independently as well as in an orchestra, producing good replications of original songs.

## Tesla coil

Intro

Reference to literature review

It was decided to make a Tesla coil as it demonstrates the power sie fo electrical engineering and can provide the orchestra with better visuals.

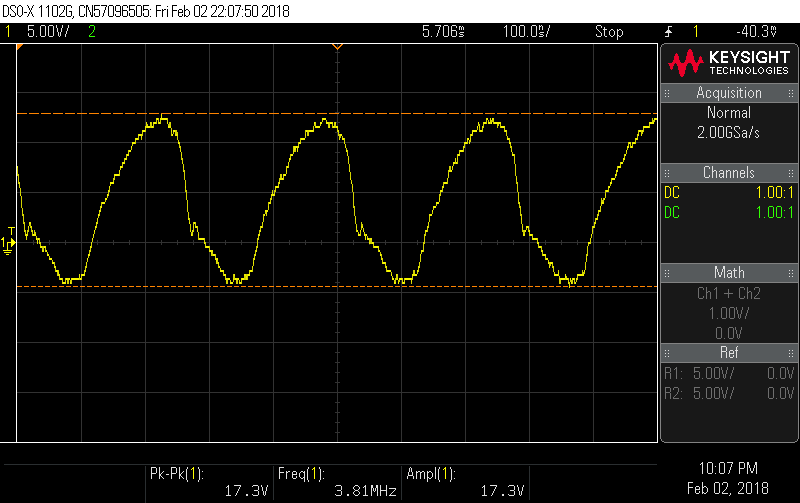
To start tesla coil circuits were investigated and it was found small tesla coil kits could be bought from amazon. Three tesla coil kits were bought to test.

The circuit for the tesla coil kits was provided with the kits but the description was in Chinese as can be seen from Appendix. Therefore, to progress with the tesla coil the kits were tested to figure out how the circuit works. Circuit diagram below.



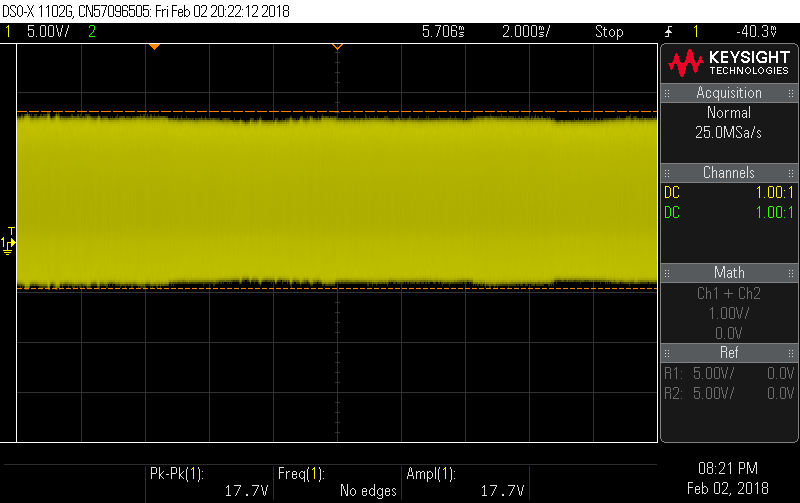
**Figure 3.?.** Tesla circuit.

The circuit works as a high frequency oscillator where Q1 and Q2 switch on and off which creates an approximate sinusoidal waveform (figure) over the primary coil causing a voltage to be induced in the secondary which produces sparks/corona discharge at the open end of the secondary coil. When a music waveform is applied at V1 it modulates the power supply to the coil causing the switching waveform to be amplitude modulated.



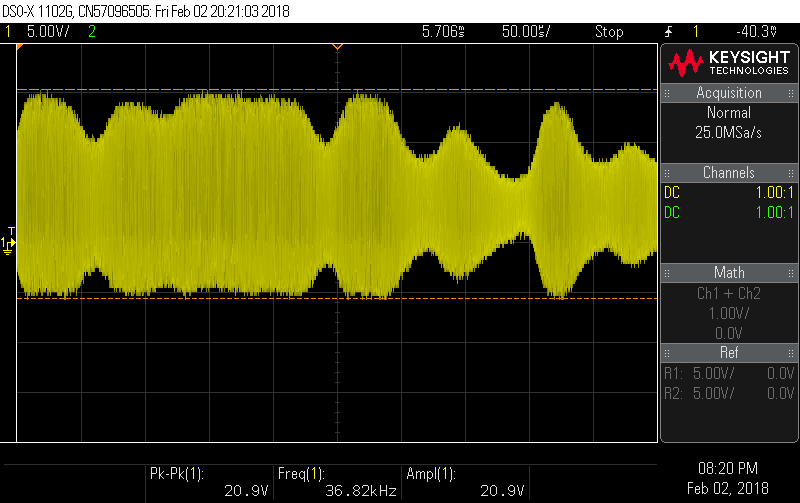
**Figure 3.?.** Single oscillation over the primary coil.

As it can be seen the switching frequency is about 1/250n=4Mhz. as the turns ratio is 350 there will be a voltage ranging from +13 to -4V over the primary and 4550 to-1400V



**Figure 3.?.** Voltage wavforme over the primary coil with no music playing.

The frequency of the audio wave form is about 1/75us=13khz this is as expected as it is in the audible frequency range.



**Figure 3.?.** Voltage waveform over the primary coil with ‘eye of the tiger’ playing.

Issues found while testing.

After continued use the sound produced by the tesla coil reduces in volume and the sparks become smaller. This is suspected to be due to the BJT heating up and so the losses increasing. The amazon description does warn about using the tesal coil for more than 3 minutes. Reducing the current that flows in the bjt from C to E which will reduce the field produced by the tesla coil and thefore the voltage induced in the secondary.

The coil is also sensitive to the primary coil position as the primary coil can move as there is nothing supporting it. As it moves further down the coil it stops the flux linkage linking the secondary coil this can be mitigated by inserting a ferrite coil into the secondary coil.

The limitation of the tesla coils is that they are currently not that loud. Using phone app it come in around … dB this is about the same as a… as can be seen in figure 3 the amplitude of the sound is controlled by the voltage amplitude. Therefore, to increase the volume the voltage need to be increased this can be done by increasing the number of turns in the secondary coil or increasing the input voltage. increasing the input voltage has the limitation that it cannot exceed the amplitude of the power supply the components are rated for a 24V supply but this would only increase by a factor of 30%.

Currently the sparks produced by the tesla coil are 2mm long as the breakdown of air is 3kV/mm. if the number of turns was increased to 10000 from 350 it woul increase the length of the streamers to 10000\*20/3000=67mm=6.7cm. This would in theory increase the sound to 10000/350=28.6 time the original increasing the volume to …. Db. If this was done the switching circuit for the tesla coil on it would be at risk of being damage by the streamers as they would reach the PCB. So remove this problem the PCB would have to be split so the switching circuit can be removed from the vicinity. A rod would also probably need to be provided to catch the streamers. The secondary coil would have to be placed in a faraday cage so nobody can touch the coil and get injured.

The testing of the new coil would have to take place in the high voltage lab discussion wer had with the operators howerver due to inprovment works and demands on the remaining test spaces the team was unable to get acess for testing in the time available.

The solution decided on was to use a microphone to pick up the sound this used a amplifer circuit already designed for the xylophone to amplify the microphone output to connect it to a speaker. As the spark is to remain small a webcam is going to be used to display the tesla spark on a screen. The tesla coil will be housed in ab box so it cannot be touch by observers so it can be used safely as part of the orchestra. The health and safety for the coil is in the appendix.

To deal with the heat problem a new PCB is to be designed with new heatsink and components. It was also designed so a teensy and DSS could be connected to control it. The design is shown below. The coils were also put on a separate PCB to make testing and transportation easier and make it easier to expand in the future.

Two new PCB’s ned to be made the components chosen for this are specified below (insert table+specs)

BJT

MOSFET

Capasitors

Other

Heat sink calculations.

Coil of wire spec

During testing several issues were found with the PCB. Several errors were found on the PCB there was a connection missing to the collector of the BJT (Q1). It was also found that the circuit provided in the tesla kits was not the same as the PCB layout on the test kits. This mean that a LED had to have its position changed. It was also found that the new BJT specified didn’t work properly with the circuit as it didn’t start to oscillate with the MOSFET so the original transistor was tried and it worked. These errors were fixed and a new PCB was printed as the testing was proceeding an alternative method was explored to use the original kit PCB with adjustments to use larger heat sinks and to be used with the Teensy. So, if the new PCB still had faults when reprinted this would be mitigated the risk. There was also a power circuit made so the tesla can be turned off in between songs to limit the amount of heat the heatsinks must deal with. This includes a manual switch and a PNP transistor that can be turned on and off by the microcontroller.

The micro controller works in the same way as the stepper with a few changes such as different tracks to be played a switch to turn off the tesla between uses.

### Tesla coil Testing

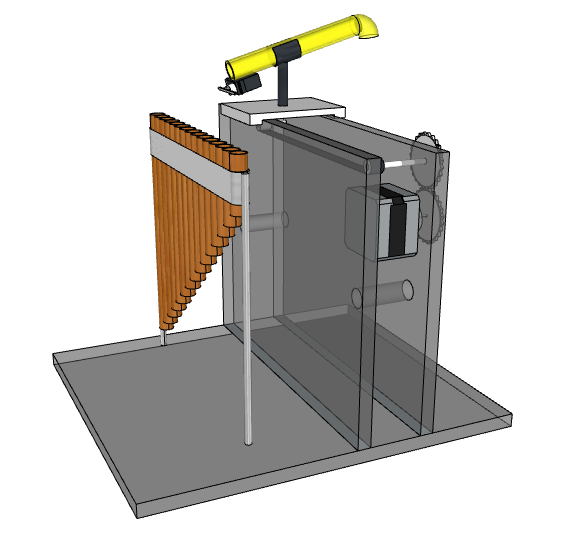
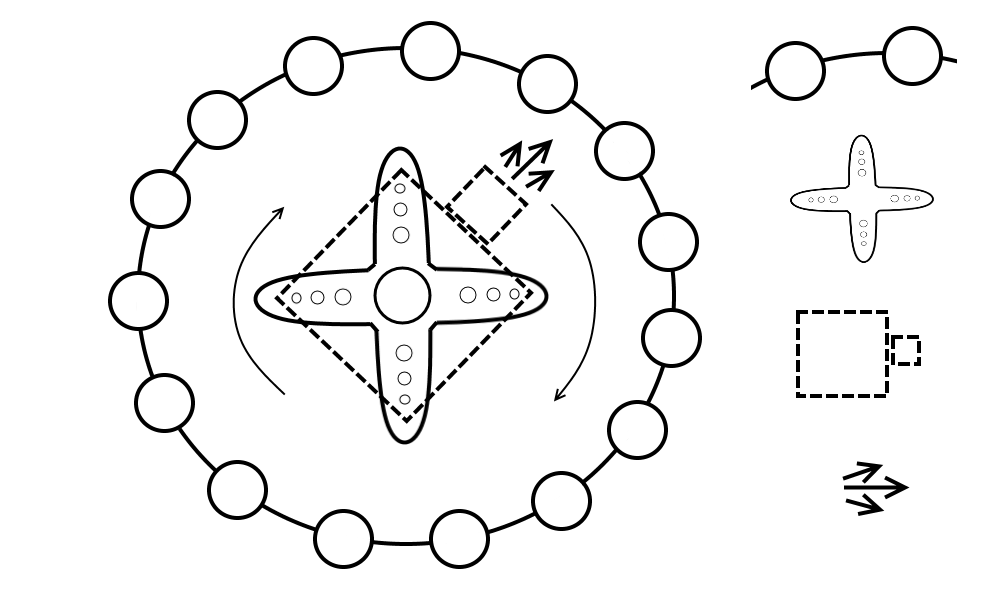
### Summary

## Panpipes

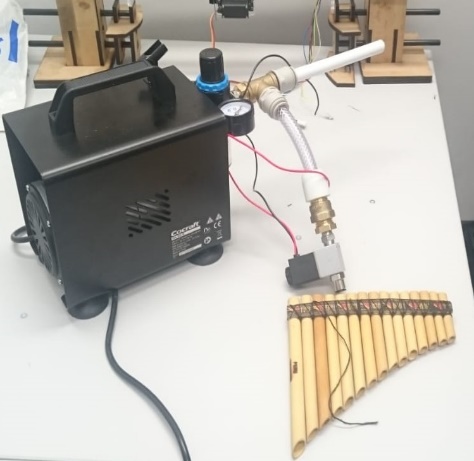
The panpipes were one of the instruments considered. They consist of 15 pipes and notes would be played by blowing air across the openings of the pipes. According to research done in [1], the pipes would require about 8.5 litres per minute [1] of airflow to play higher notes.

After testing was done both on the panpipes and Tesla coils, the Tesla coils were chosen over the panpipes, but there was a considerable amount of work done on the panpipes before this decision, which will be discussed in this section.

### Versions

****There were 3 main designs for the panpipes throughout the project.

**Figure 3.1c.** Third Design.



**Figure 3.1b.** Second Design.

**Figure 3.1a.** First Design.

**First design**

The initial design seen in Figure 1.1a had the panpipes fixed to a base and the nozzle that directed the air from the pump (mattress pump at the time). The nozzle was on an elevated platform that would move from side to side to play different notes and had a breaker that would cut off the airflow when the platform would be moving from pipe to pipe. This platform would be powered by a stepper motor allowing it to move to individual pipes.

This design was later changed to the second design shown in Figure 1.1b which would simplify the moving mechanism.

**Second design**

Instead of a stepper powered track moving sideways, the idea was to mount the nozzle onto a servo motor which could rotate 360 degrees. The pipes would also be taken apart and mounted in a circle around the servo with the nozzle.

This design had flaws due to the moving components and how there might been and increasing error over time of where the nozzle is aiming, to the point where the nozzle will be blowing air past the pipe. This will lead to the final design which removed most of the moving parts.

**Third design**

The final design seen in Figure 1.1c would have an air compressor providing the airflow since after tests were done it was established that the mattress pump did not provide enough pressure to produce a loud enough sound. There were to be 15 ducts to each panpipe and at the end of each duct there would be a solenoid valve that is normally closed, and when a specific note needs to be played, the valve would be turned on to let the air through.

After the final tests it was found that the pressure was still lost along the pipes and the sound made by the air coming out of the valve was still too quiet. This meant that a more expensive air compressor or better pipes needed to be bought which could lead to the risk of being over budget.

## Outreach

# Summary

## Overall achievement

## Future work

### Conductor

### Keyboard

### Xylophone

### Stepper motors

### Tesla

## Progress against Gantt Chart

## Management

# Conclusion

# References