**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

# Executive Summary

This report presents the work undertaken by MEng Team 11 on the “New core infrastructure for the Robot Orchestra” project. The project combines engineering with music and art into a single application that can appeal to a wide audience and advertises engineering to people who might not normally consider it. This final year project was motivated by the rising public and industrial interest in the existing Robot Orchestra which, when expanded with the new core, will have greater flexibility, reliability and showcase industrial-standard equipment in an exciting non-traditional way.

The aim of the project was to increase the existing orchestra’s capabilities and allow for a wider selection of musical styles to be played while also allowing for the new instruments to be used as a standalone band. The objectives of the project were to select, design, manufacture and integrate four new instruments, as well as an electronic conductor, and assemble them into a core orchestra that can play together at least two recognisable songs. The instruments that have been developed for this project are: a keyboard, stepper motors, Tesla coils, all using a Teensy board as a controller and a novel xylophone that uses National Instruments’ myRIO. The electronic conductor is based on a Raspberry Pi that sends all the necessary information to each instrument via WiFi. The instruments can play both individually as well as jointly songs such as *Californication* by the *Red Hot Chilli Peppers, Eye of The Tiger* by *Survivor* and the *Game of Thrones Main Theme*. The system has been designed with expandability in mind, allowing for future songs and instruments to be easily added. Additionally, it has been designed to allow for easy transportation and set up, making it suitable for use in many different locations and venues.

The project showcases the operation of multiple embedded systems to perform a shared task and it has a dual commercial applicability. First, the orchestra can be used at trade shows and other expo events by technology companies such as National Instruments (myRIO), ARM (Teensy Board) or Broadcom (Raspberry Pi) to showcase their products that are being used in this atypical application. Secondly, as the project can raise the profile of engineering, it can support corporate social responsibility agendas as well as the government’s STEM initiatives.

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, 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 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.

Table of Contents

[1. Executive Summary 2](#_Toc513467987)

[2. Introduction 5](#_Toc513467988)

[3. Aims and Objectives 5](#_Toc513467989)

[3.1. Aims 5](#_Toc513467990)

[3.2. Objectives 5](#_Toc513467991)

[4. Literature Review 6](#_Toc513467992)

[4.1. Conductor 6](#_Toc513467993)

[4.1.1. MIDI literature review 6](#_Toc513467994)

[4.1.2. Choice of network 7](#_Toc513467995)

[4.1.3. MQTT Protocol 7](#_Toc513467996)

[4.1.4. MQTT Protocol 7](#_Toc513467997)

[4.2. Keyboard 8](#_Toc513467998)

[4.3. Xylophone 9](#_Toc513467999)

[4.4. Stepper Motors 9](#_Toc513468000)

[4.5. Tesla coil 10](#_Toc513468001)

[4.6. Panpipes 11](#_Toc513468002)

[4.7. Guitar 12](#_Toc513468003)

[5. Motivation 12](#_Toc513468004)

[6. Project Roles 13](#_Toc513468005)

[7. Required Skills 13](#_Toc513468006)

[8. Interdependencies 14](#_Toc513468007)

[9. Report Structure 16](#_Toc513468008)

[10. Technical Chapters 16](#_Toc513468009)

[10.1. Conductor Overview 16](#_Toc513468010)

[10.2. SDManager 16](#_Toc513468011)

[10.3. Bithoven 17](#_Toc513468012)

[10.3.1. Wireless Network 17](#_Toc513468013)

[10.3.2. Summary 18](#_Toc513468014)

[10.4. Keyboard 18](#_Toc513468015)

[10.4.1. Overall design description 18](#_Toc513468016)

[10.4.2. Solenoid Selection 19](#_Toc513468017)

[10.4.3. Supporting the Solenoids 20](#_Toc513468018)

[10.4.4. Solenoid PCB design 21](#_Toc513468019)

[10.4.5. Supporting the Bosch Bar 21](#_Toc513468020)

[10.4.6. PCB Support Plate 23](#_Toc513468021)

[10.5. Keyboard Software 23](#_Toc513468022)

[10.5.1. SN74HC595 Shift Registers 24](#_Toc513468023)

[10.5.2. Structure of the program 24](#_Toc513468024)

[10.6. Keyboard Testing 26](#_Toc513468025)

[10.7. Summary 26](#_Toc513468026)

[10.8. Xylophone 26](#_Toc513468027)

[10.8.1. Xylophone Software 26](#_Toc513468028)

[10.8.2. How it works 26](#_Toc513468029)

[10.9. Stepper Motors 28](#_Toc513468030)

[10.9.1. Stepper Motor Hardware 28](#_Toc513468031)

[10.9.2. Acoustic box 29](#_Toc513468032)

[10.9.3. Digital Signal Synthesizer 29](#_Toc513468033)

[10.9.4. Motor Driver Board 30](#_Toc513468034)

[10.9.5. Stepper Motor Software 30](#_Toc513468035)

[10.9.6. Stepper Motors Testing 31](#_Toc513468036)

[10.9.7. Summary 32](#_Toc513468037)

[10.10. Tesla coil and Panpipes 32](#_Toc513468038)

[10.10.1. Panpipes 32](#_Toc513468039)

[10.10.2. Versions 32](#_Toc513468040)

[10.10.3. Research and testing for the tesla coil 33](#_Toc513468041)

[10.10.4. Issues found while testing. 36](#_Toc513468042)

[10.10.5. Possible Solutions 36](#_Toc513468043)

[10.10.6. Summary 36](#_Toc513468044)

[10.10.7. Decision between Tesla coil and Panpipes 36](#_Toc513468045)

[10.11. Tesla Coil progress 36](#_Toc513468046)

[10.11.1. Heat sink calculations 36](#_Toc513468047)

[10.11.2. Component specification 37](#_Toc513468048)

[10.11.3. Testing PCBs 37](#_Toc513468049)

[10.11.4. Testing 38](#_Toc513468050)

[10.11.5. Summary 38](#_Toc513468051)

# 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]. Recently, the Manchester Robot Orchestra has been accepted to appear at the competitive MOSI Maker Fest on May 26th. 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.

# Aims and Objectives

## 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 after sending the “Play” command to the conductor from the GUI.

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”. [1]

## Objectives

After each member of the team got their general roles the following objectives were set:

1. Select four new instruments:

* Look at different songs that could incorporate 4 robot instruments using Anvil Studio which looks at the MIDI file break down.
* Pick two songs as well as the 4 instruments that will be playing those songs.

1. Design and construct the four instruments:

* Propose designs for the instruments as well as ideas for their communication with the conductor while considering transportation and modularity.
* Use CAD to design hardware and then produce prototypes.
* Design and test breadboard circuits and convert to PCBs.
* Print the PCBs make sure they work with the software like the breadboard circuits.
* Construct the hardware.
* Test the instruments and make sure they play all the notes that Anvil Studio shows they need to play and at the right tempo.

1. Design a conductor:

* Decide on an embedded system for the conductor.
* Design the conductor which will communicate with the instruments.
* Communicate with each instrument using the conductor.
* Receive “Play” and “Stop” from the GUI and send those commands to the instruments.

1. Synchronise the 4 instruments to play coherent music.

# Literature Review

## Conductor

### MIDI literature review

MIDI stands for Musical Instrument Digital Interface and is used to encode all the data required to replicate a song under a common format. MIDI files specify which note to play, for how long and with what velocity the instrument is to play the note [3] [4]. MIDI has become a standard interface used in the music industry and will be used in the project to acquire songs. MIDI files for a vast number of songs can be easily found online, allowing the team to easily choose any song available.

Each MIDI file consist of a *Header Chunk* and *Track Chunk*. The header chunk specifies various parameters needed to decode the rest of the file. It is constructed as follow [2][3][5]:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Length** | *4 bytes* | *4 bytes* | *6 bytes* | | |
| **Value** | *‘MThd’* | *Length (6bytes)* | *FF FF* | *NN NN* | *TT TT* |

Table 1 Header chunk

The top row in Table 1 indicates the size allocated to each section of the header. The bottom row contains the data.

* The first 4 bytes of the header chunk correspond to **‘MThd’**  in ASCII (4D 54 68 64 in HEX), this specifies that the incoming chunk is a header chunk.
* The next 4 bytes specify the length of the data which is to come. For header chunks this is always 6.
* The next 6 bytes of data specify three things [5] [6] [7]
  + **FF FF -** The file format of the MIDI file. Three formats are accepted:
    - FF FF = 1: One track in the MIDI file.
    - FF FF = 2: Multiple synchronous tracks in the MIDI file. Which means that the file contains multiple tracks which all start at the same time.
    - FF FF = 3: Multiple asynchronous track in the MIDI file. Which means that the file contains multiple tracks all with different starting points
  + **NN NN -** The number of tracks in the MIDI file.
  + **TT TT -** The timing parameter The MIDI standard defines timing in its own way, which is *delta ticks per quarter note* and essentially defines the timing for the rest of the MIDI file

The remaining MIDI file consists of the *Track Chunk* Table 2 shows its layout.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Length** | 4 bytes | 4 bytes | Variable depending on specified length | |
| **Value** | *‘MTrk’* | *Length* | *Delta time* | *Event* |

Table 2 MIDI Track Chunk

* The first 4 bytes of the track chunk are **‘MTrk’** denoted in ASCII (4D 54 72 6B in Hex) this specifies that the incoming chunk is a track chunk.
* The next 4 bytes specify the length of the data which is to follow, which varies depending on the data.
* The following 4 bytes specify the *Delta Time,* this is the amount of time that needs to pass prior to executing the MIDI event.
* The last section of the header specifies the event which is to be executed.

The track event can either be a MIDI, meta or sysex events. For this project, only MIDI events need to be considered. MIDI events contain messages sent to each individual channel (instrument) and contain the note and status information. MIDI events consist of three bytes, the first byte is known as the status byte and its function is to specify the type of command, for our purpose only two commands will be used, *press* and *release* which tell the instrument to either play or release the note. Although other commands exist for instruments with additional features like pedals, they will not be considered as the instruments chosen do not require them. The second byte specifies the note to which the status byte is applied to and the third byte specifies the note velocity (loudness or softness). A table showing the musical note to which each binary value corresponds to can be found in **Appendix X**. [5] [6] [7]

|  |  |  |  |
| --- | --- | --- | --- |
| **Length** | 1 byte | 1 byte | 1 byte |
| **Function** | Status Byte | Note Number | Note Velocity |

Table 3 Track events

### Choice of 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 decision. 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 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 WiFi modules are cheap and can easily be integrated with any microcontrollers which have *Universal Asynchronous Receiver Transmitter* (UART), more on the WiFi modules can be found …... Due to these reason, WiFi was chosen as a base to build the network on.

### MQTT Protocol

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 stack. 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. MQTT is based on a client-server model where the client is connected to a server and receives messages from it, an MQTT server is known as a ‘broker’. Clients have the ability to *‘publish’* and *‘subscribe’* to *‘topics’*. Any client can publish to any topic and other clients which have subscribed to the same topic will receive the messages. The Raspberry Pi is capable of hosting an MQTT broker as it very efficient and has a small code footprint. The diagram below depicts an network based on the MQTT protocol.

play a major role in the decision. 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 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 WiFi modules are cheap and can easily be integrated with any microcontrollers which have *Universal Asynchronous Receiver Transmitter* (UART), more on the WiFi modules can be found …... Due to these reasons, WiFi was chosen as a base to build the network on.

### MQTT Protocol

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 stack. 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. MQTT is based on a client-server model where the client is connected to a server and receives messages from it, an MQTT server is known as a ‘broker’. Clients have the ability to *‘publish’* and *‘subscribe’* to *‘topics’*. Any client can publish to any topic and other clients which have subscribed to the same topic will receive the messages. The Raspberry Pi is capable of hosting an MQTT broker as it very efficient and has a small code footprint. The diagram below depicts an network based on the MQTT protocol.

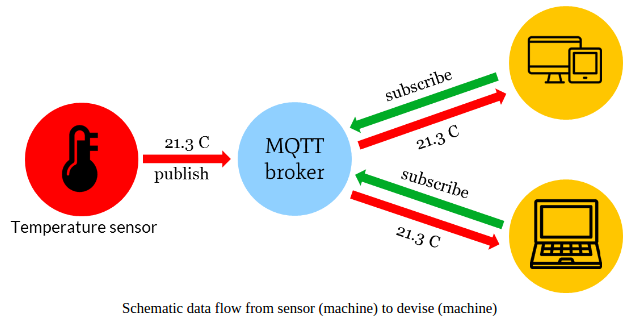


Figure 4.1 Operation of MQTT Protocol

## Keyboard

During the first project block in Semester 1, the team was split into individual sub teams where each team was responsible for researching an instrument. Once the research period was over, another meeting was held to discuss potential instruments. One of the first instruments that was decided on was the keyboard. To make this decision, several designs were reviewed and a paper by *Sugano* and *Kato* was reviewed [8].

In their paper, the authors aim to progress in the field of “soft robotics” [8] by creating two robotic hands that are capable of playing the keyboard by moving like human hands. To design a robot that can do this it is important to consider the degrees of motion that a robot must go through to play the keys. As *Sugano* and *Kato* explain, there are four primary degrees of freedom which occur at the three joints in the finger (MP, PIP and DIP in Figure 4.2) and the wrist.

|  |  |
| --- | --- |
| Figure 4.2 Degree of freedom found in the finger and thumb in a human [8] | Figure 4.3 Robotic hand proposed by *Sugano* and *Kato* [8] |

The DIP joint can be neglected because it is only used whenever the PIP joint is active, and it serves no purpose when pressing a key down on a keyboard [8]. The PIP joint differentiates between a white key and a black key on the keyboard with the MP joint being responsible for pressing down on the selected key. The wrist is responsible for rotating the entire hard to give the fingers access to more keys. Given that in this project 4 instruments were being designed, it was important to make the design of each of the instruments as simple as possible, which can be achieved by reducing the degree of freedom each of the joints described in [8] has. The design shown in Figure 4.4 is an example of a robot playing a keyboard where the PIP and wrist joint have been limited and fixed in place, with the MP joint controlled by solenoids that push down on the keys.



Figure 4.4 Teotronica robot keyboard [9]

The benefit of this type of design is that is significantly simpler to construct, both in terms of hardware and software, than the design proposed in [8]. By fixing the PIP and wrist [8] joint in place, only the MP joint has to be controlled through the software. However, fixing these two joints in place significantly reduces the flexibility of the instrument. The design shown in Figure 4.4 can only play songs that the solenoids are fixed above, any other song and additional hardware must be constructed. In addition to this, the design will look less human like which reduces the aesthetic appeal. For application in this project, simplifying the hardware is more important than creating a design that is human-like as the sound the orchestra produces is the most important metric whereas in [8] the aim was to create an anthropomorphic design. Also, simplifying the design reduces the time spent on each instrument allowing more time for others to be designed and reducing the chance that instruments will not be completed.

## Xylophone

One of the instruments that was also reviewed, is the xylophone. It is considered to be part of the percussion family, and is consisted of wooden bars that differ in size. Each bar is an idiophone (musical instruments which have a resonant solid material that vibrates in order to produce sound [10]) and produces various sounds which depend on their length. For example, shorter bars produce higher notes while longer bars produce lower notes [11]. According to the country of origin, the xylophone is tuned in different musical scales and therefore has different number of notes per octave. For instance, in Africa and Asia, the xylophone belongs in the pentatonic and heptatonic scale having in that way, five and seven notes per octave respectively [12], [13]. When it comes to orchestral use, the xylophone is tuned in the chromatic scale and hence, it is constituted by twelve notes per octave.

There were several designs found online and reviewed, but the team decided to create a customized autonomous xylophone, which is able to produce sounds according to the team’s preferences. That idea is based on a video found on YouTube, where a person created a customized xylophone with eight keys. Each key had a piezoelectric sensor mounted at its centre and was programmed (through an Arduino) to produce a specific sound when it was hit.

## Stepper Motors

As stepper motors produce noise when rotating at different frequencies, the possibility of using them as an unconventional musical instrument was investigated. It was discovered that multiple videos of such an instrument were posted online such as those presented in [14] and [15] which confirmed that the instrument was feasible.

A stepper motor converts digital pulses into steps of a full rotation [16] and by varying the frequency of the pulses, the rotation of the shaft can be modified. The key difference between a stepper motor and a servo motor is that the stepper motor does not require a feedback mechanism since there is a direct relationship between a digital pulse and the amount by which the motor shaft rotates [17].

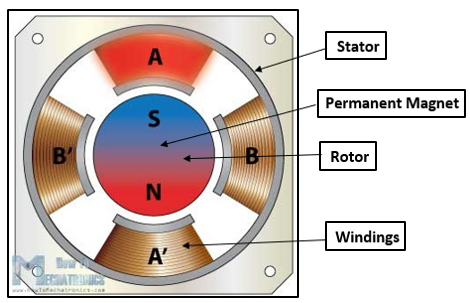


Figure 4.5 Cross section of stepper motor [18]

Figure 4.5 illustrates the cross section of a stepper motor. It can be observed that the permanent magnet in the centre acts as the rotor of the motor and is surrounded by stator’s windings. To understand how the rotational movement is generated, the rotor and stator will be analysed separately. The rotor consists of two discs placed back to back: one of them acts as the north pole and the other acts as a south pole. The windings act as electromagnets that can be independently controlled and when a winding is activated, it attracts the disc of opposite polarity of the rotor. As the windings get activated in turns, this results in the rotational movement of the motor’s shaft being generated [19]

A stepper motor can be controlled in three different ways, depending on the precision that is required. These modes are full step, half step and microstep and the difference between them is represented by the order in which the windings are energised. For example, one digital pulse sent to a motor operating in half step mode produces an angular movement that is half of the one that is obtained by the same pulse when using a motor operating in full step mode.

The key aspect of stepper motors for them to be used as a musical instrument is represented by their ability to generate frequencies in the audible range (20 Hz – 20 KHz [20]) that correspond to musical notes. A song can be replicated by the stepper motors by sequentially generating the frequencies of the musical notes required and ensuring that they are generated for the correct amount of time.

## Tesla coil

The Tesla Coil was invented by Nikola Tesla in 1891 as a way to transmit electrical power wirelessly [21]. A tesla coil consists of a primary coil with a low number of turns and a Secondary (tesla) coil of which one side is connected to ground and the other is left as an open circuit [21]. A typical tesla coil circuit is shown in Figure 4.6. In figure (a) the voltage rises to a point at which a spark can form over the spark gap, at this point the combination of the capacitor primary and secondary inductance causes a high frequency current. This is created the energy oscillates between the capacitor and the primary coil inductance [22]. When there is no spark formed over the spark gap the capacitor charges. There are several disadvantages to this design as when a spark is formed there is a high power and current flow [21]. For DC operation the spark gap is exchanged for a switch which can be controlled to charge the capacitor and then to let it discharge over the primary coil [22].

|  |  |
| --- | --- |
|  |  |
| Figure 4.6: Spark gap Tesla coil circuit (left) and a switching Tesla coil circuit (right) [23] | |

The building of a tesla coil can be broken down into several parts: the primary coil, secondary coil, spark gap or switching circuit, capacitors and Toroid [24]. To make a tesla coil play music the switching circuit which charges up the capacitor is controlled using PWM [25]. There are also several description on how to build a tesla coil from amateur enthusiasts [26] [23] [27].

A tesla coil instrument could also be a good idea for the robot orchestra as it can be used as an educational tool. As it can be used to demonstrate electrical fields and the wireless transfer of energy this can be done by holding a bulb close to the coil and it will light up with no connections [28]. Tesla coils can also be used for simulating lightning i.e. testing aircraft however, currently Impulse Voltage generators are more commonly used [29].

There are examples of tesla coils being used to play songs on YouTube, one such example can play ‘the Hall of the Mountain king’ which produces streamers several meters in length. [30]

## 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 [31], the pipes would require about 8.5 litres per minute [31] of airflow to play notes the octaves above C6 (if C1 is the first C).

Research was done on other wind instruments such as the Waseda Flutist Robot [32] but due to the complexity of some components such as the artificial lips and lungs this was considered an unfeasible option due to the time constraints of the project. Another example of a wind instrument was the Saxophone Playing Robot [33] but that too required complex moving

parts mentioned before.

|  |  |
| --- | --- |
| Figure 4.7 Waseda Flutist Robot [32]. | Figure 4.8 Saxophone Playing Robot [33]. |

Proof of concept can be seen in video [34] where a set of panpipes is being played by a robot that moves the panpipes across a nozzle that blows air across the pipes producing the corresponding notes. The panpipes are easier to automate since there are less moving parts and Degrees of Freedom (D.o.F) involved compared to the other wind instruments. The panpipes were therefore the preferred instrument due its simplicity and range of notes.



Figure 4.9 Panpipe playing Robot (Dare, 2011).

## Guitar

The guitar was another instrument that briefly researched as a potential instrument. There are few research papers on 6-stringed guitars therefore videos were the primary source of analysis. A paper on the history of robot instruments [35] shows a guitar with “72 finger left hand” [35], and a video of Compressorhead preforming the *Ace of Spades* [36].

|  |  |
| --- | --- |
| Figure 4.10 Sergi Jorda’s Afasia from (Kapur, n.d.) | Figure 4.11 Compressorhead 6-String guitar robot preforming Ace of Spades (Nomate, November 2017). |

Both robots have a relatively simple plucking system but a complex fret mechanism consisting of multiple solenoids or other moving parts which could be well take up a large portion of the budget and time which needs to be given to other instruments. This instrument was later discarded due to these reasons.

# Motivation

The motivation behind this project is represented by the rising interest in the university’s existing Robot Orchestra as the orchestra has recently been part of an engineering tour sponsored by the Royal Academy of Engineering and it has also been part of a BBC programme [37]

Due to the fact that the existing orchestra has a narrow selection of songs it can play, increasing the orchestra’s capabilities by adding new instruments would allow for a wider selection of musical styles to be played and would lead to the orchestra appealing to a broader audience. In addition to its outreach aspect, the orchestra also showcases the operation of multiple embedded systems to perform a shared task, making use of platforms such as MyRIO, Raspberry Pi and Arduino to play recognisable songs.

As there is a high interest in attracting students to study STEM degrees and as it can be difficult to explain the benefits of such a degree to an audience of different backgrounds and age categories, the orchestra can serve as a good aid for universities to promote their courses and advertise engineering to people who normally would not consider it. Additionally, the orchestra can also be used by companies in the technology industry at trade fairs to illustrate the creative aspects of engineering and to showcase the applications of their products in an atypical project.

# Project Roles

In addition to the technical skills required for this project, the team will have to utilise several soft skills. Table 4 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 4 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 4.2 [38] 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 were 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 5 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 5 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 8.1 below shows the main tasks for the project that had to be completed and how they relate to one another. The four instruments and the conductor were developed in parallel with each, so they were not interdependent on one another. Within each of the instruments, the software and hardware were developed in parallel with each other. This was possible as the first block of the project was spent solely on developing designs for each of the instruments. As the designs were solidified in the first block, the software could be developed without having the hardware present, just based off the designs created in the first project block.

A screenshot of a cell phone

Description generated with high confidence

Figure 8.1 Interdependencies amongst tasks

The *“Instruments Communicating with Conductor”* task shown in Figure 8.1 was used to create and test the communication channels between the conductor and the instruments. This task was dependent on the preliminary software testing on the instruments and conductor being complete as there needed to be a host (conductor) and a node (instrument) to test that the messages sent by the conductor are being received. Fortunately, since the Keyboard, Stepper Motors and Tesla Coil use the same embedded system and communicate with the conductor in a similar way there is not a heavy need for the software for all three of these instruments to be completed at the same time. A delay in finishing the software for any one of these instruments would not have caused significant delays in the project as long as one of the three was completed on time. However, since the Xylophone uses a different embedded system to the others, a delay in the completion of the software for this instrument would have had significant impact on the project. It was important to identify this at the beginning of the project as it meant that if the Xylophone was falling behind, resources could be moved from the software development of one of the other instruments to try and keep the Xylophone on track.

The *“Synchronisation”* task is dependent on the *“Instruments Communicating with Conductor”* since the synchronisation can not begin until a solid connection has been established between the instruments and the conductor. A delay in establishing a connection between the instruments and the conductor may have had a significant impact on the project. It was difficult to predict, in the early stages of the project, how long the synchronisation of the instruments would take so it was important to stick to the deadline in the previous task to enable sufficient time to synchronise the instruments, should they take a significant amount of time to synchronise and play in time with another.

The final task, *“Demo Day”*, is dependent on the orchestra being completed and synchronised with one another. This task is to largely to increase the aesthetic appeal of the project as well as sorting out logistics behind moving the orchestra, how many plugs were necessary etc. At the end of the *“Synchronisation”* task, the orchestra should be functioning as expected but is unlikely to look as professional as needed. That is why the *“Demo Day”* task exists, to give time for the team to adjust the aesthetic appeal and prepare to showcase the orchestra.

# Report Structure

The report is structured as follows: Section 2 focuses on the aims, objectives, motivation and interdependences of the project. Additionally, it presents the literature review underlining the research done on the conductor and on each instrument of the orchestra. This is an enhanced version of the literature review presented in the Interim Report which is presented in Appendix X. Section 3 presents the technical work undertaken for the conductor and for each musical instrument in the orchestra. Following, Section 4 offers a summary of the work done during the project and presents the available options for future work that can be done for each individual element of the orchestra. Lastly, Section 6 draws a conclusion on the work presented in this report.

# Technical Chapters

## 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 10.1: Conductor with four instruments

## 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 accurately. 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 instrument 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 the portion of the MIDI file required by the instrument 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.

|  |  |
| --- | --- |
| **Name** | **Description** |
| **MIDI2Text** | Converts a MIDI file into either a text file or a binary file  containing the three arrays needed by the instruments. |
| **SDManager** | Saves the output of MIDI2Text for into an SD card. |
| **bitHoven** | Handles the communication between the Raspberry Pi and the instruments. Used to PLAY and STOP a track. |

Table 6:Summary of scripts

## 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 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.

### Summary

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. 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. 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.

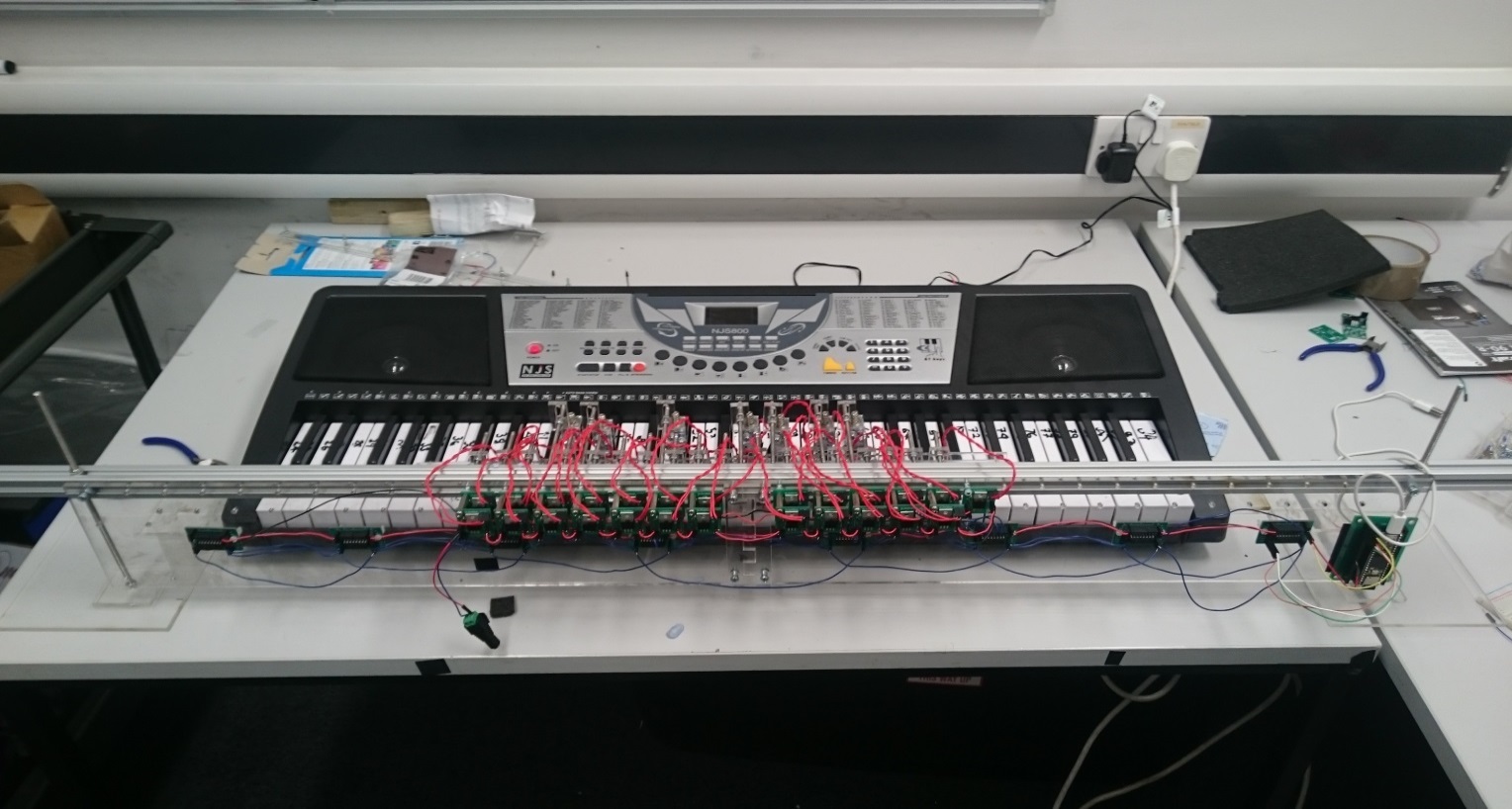
## Keyboard

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 ?). the main step to doing this are
  + be able to play 25 notes
  + design and manufacture PCBs to power solenoids
  + design and build a stand to hold the Solenoids and PCBs
  + write software for the Teensy microcontroller to control the instrument
  + integrate the software with the hardware
* communicate with the conductor so it can be controlled alongside the other instruments (objective 4)
  + write software so it can be controlled with the conductor
  + integrate the Wifi module with the Teensey
  + integrate the conductor with the instrument
* be transportable (objective 3)
  + design the hardware so it can be dismantled for transport.

### Overall design description

Figure 10.2 Keyboard hardware overview 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 within the project budget of £1500 as each solenoid costs £10.40 so to buy 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 that 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(need to put it in a box to test).



Threaded rod supports with base plate

Shift register PCB

Solenoid switching Transistor PCB

Solenoids and brackets

Bosch Bar

Teensy Embedded system

Figure 10.2 Keyboard hardware overview

### 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 so using a safety factor of 25% the keys need a force of 1.25\*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 power supply. 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 (section 4.3 of the interim report calculates the maximum to be 30%). 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 (solenoid datasheet in Appendix…).

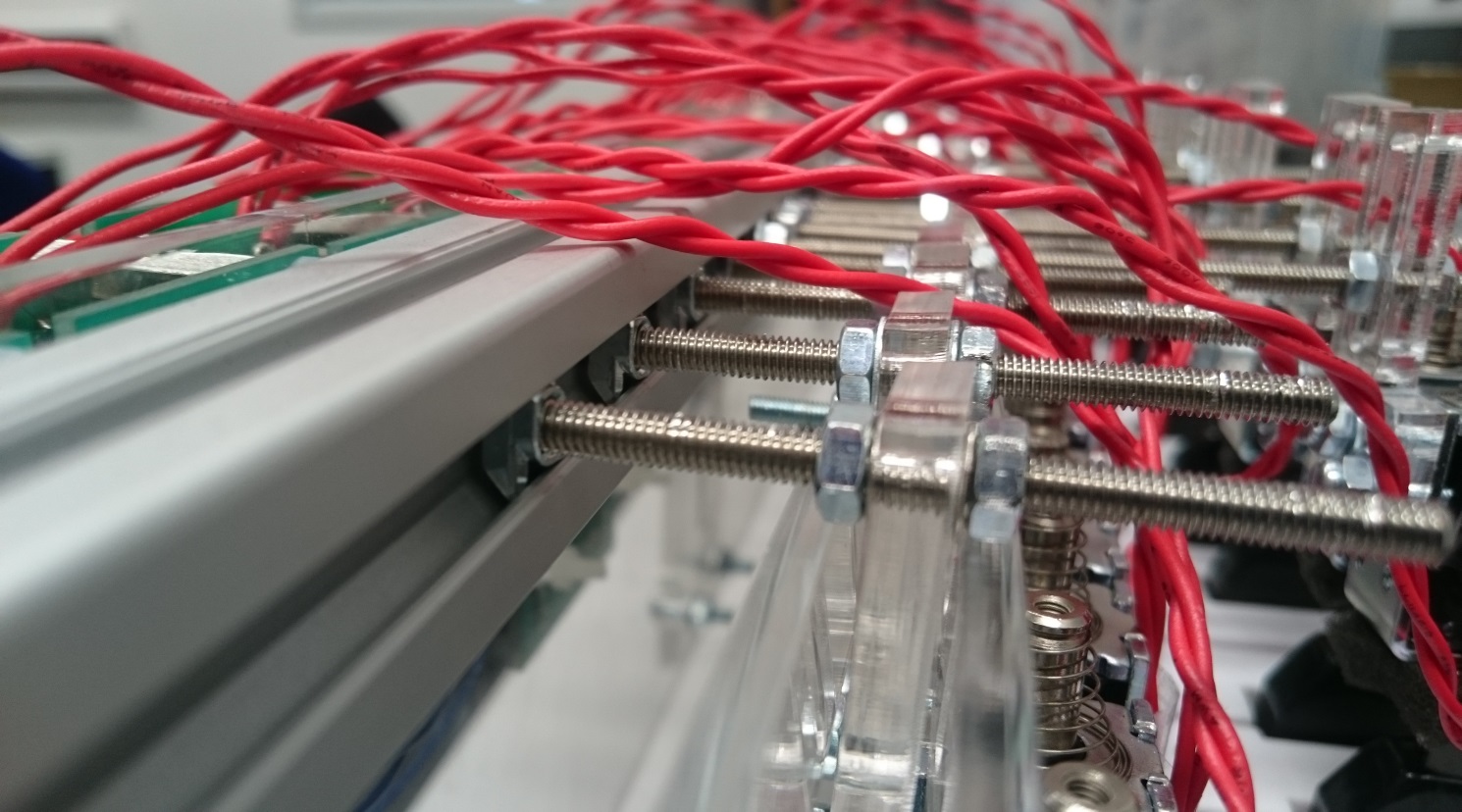
Initially, one solenoid was bought to test. The solenoid came without a stopper so one was laser cut in the mechanical workshop this allows the solenoid throw to be adjusted. It was decided this was a useful feature as the throw could be adjusted to the exact value necessary. The solenoid was tested and it was able to press both the white and black keys. The main issue found when testing 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 10.3.

|  |  |
| --- | --- |
| C:\Users\Josh\Downloads\DSC_0092.JPG | C:\Users\Josh\Downloads\DSC_0095.JPG  Stopper  Stopper Foam  Plunger Foam  Plunger |
| Figure 10.3: Solenoid with foam | |

This lessened the effect however; another clicking sound was made when the plunger hit the inside of the solenoid. An attempt was made to lessen this as well by putting a small piece of foam on the plunger Figure 10.3. This caused issues to the working of the solenoid as it added extra resistance for the solenoid to work against. Therefore, 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 the synchronisation of the instruments so the foam on the plunger was removed. The volume of the keyboard can be turned up to a level that mitigates the sound of the clicking. The clicking is not entirely a negative as it gives the instrument a mechanical sound which fits in with the aesthetic of the orchestra.

### Supporting the Solenoids

To make the assembly 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. The assembly is shown in Figure 10.4



T-slot nut

Threaded rod

Bracket

solenoid

Bosch Bar

Figure 10.4: 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 15mm meaning that the bracket must be the same width as the solenoid

The design went through several iterations. The first design is shown in Figure 10.5 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 10.5 this allows for horizontal and vertical movement (25mm) the design is stepped so the threaded rod can pass the solenoid plunger. It also means that the solenoid connection part can be greater than 16mm as the threaded rod for the black keys will run past the upper section. The third design just adjusted the distance between the step so there was a 1.5mm difference between each side which makes it easier to overlap them in the mounting process.

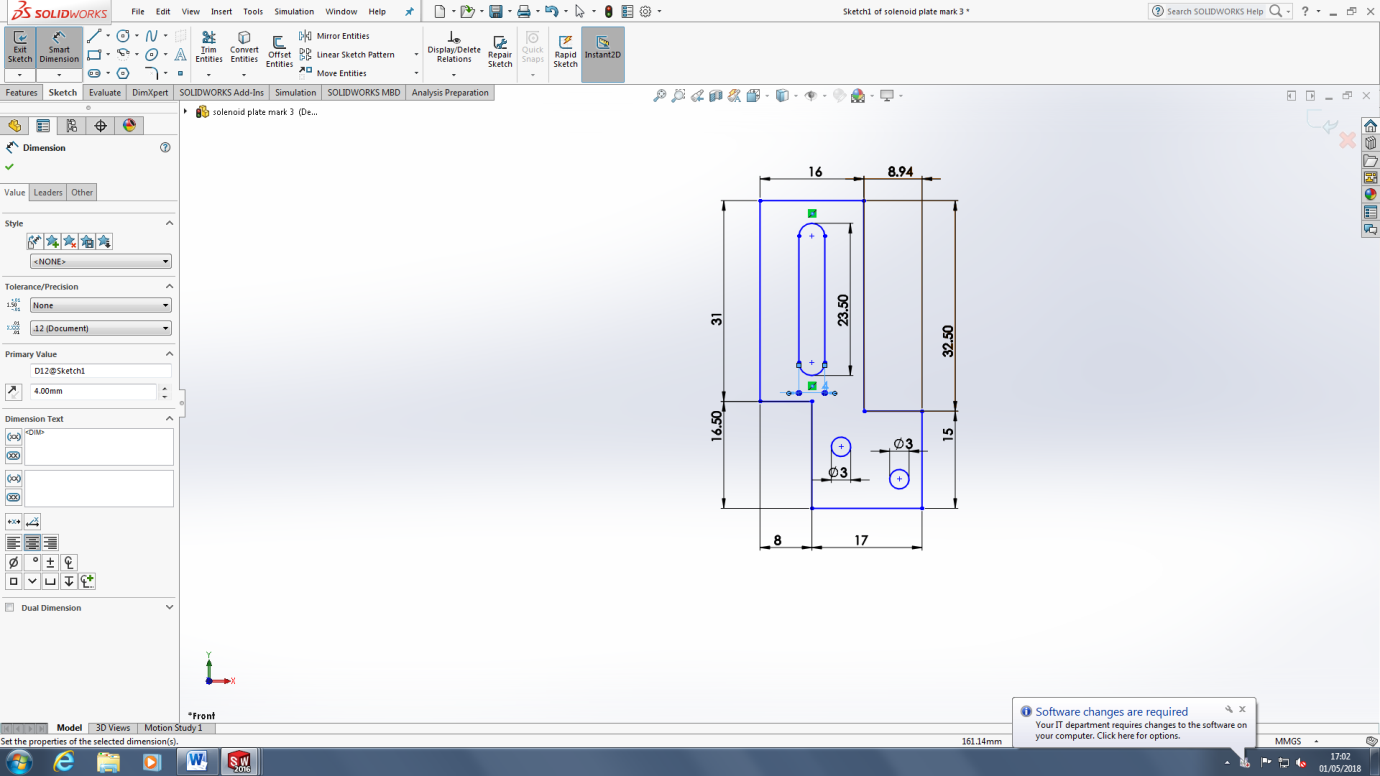
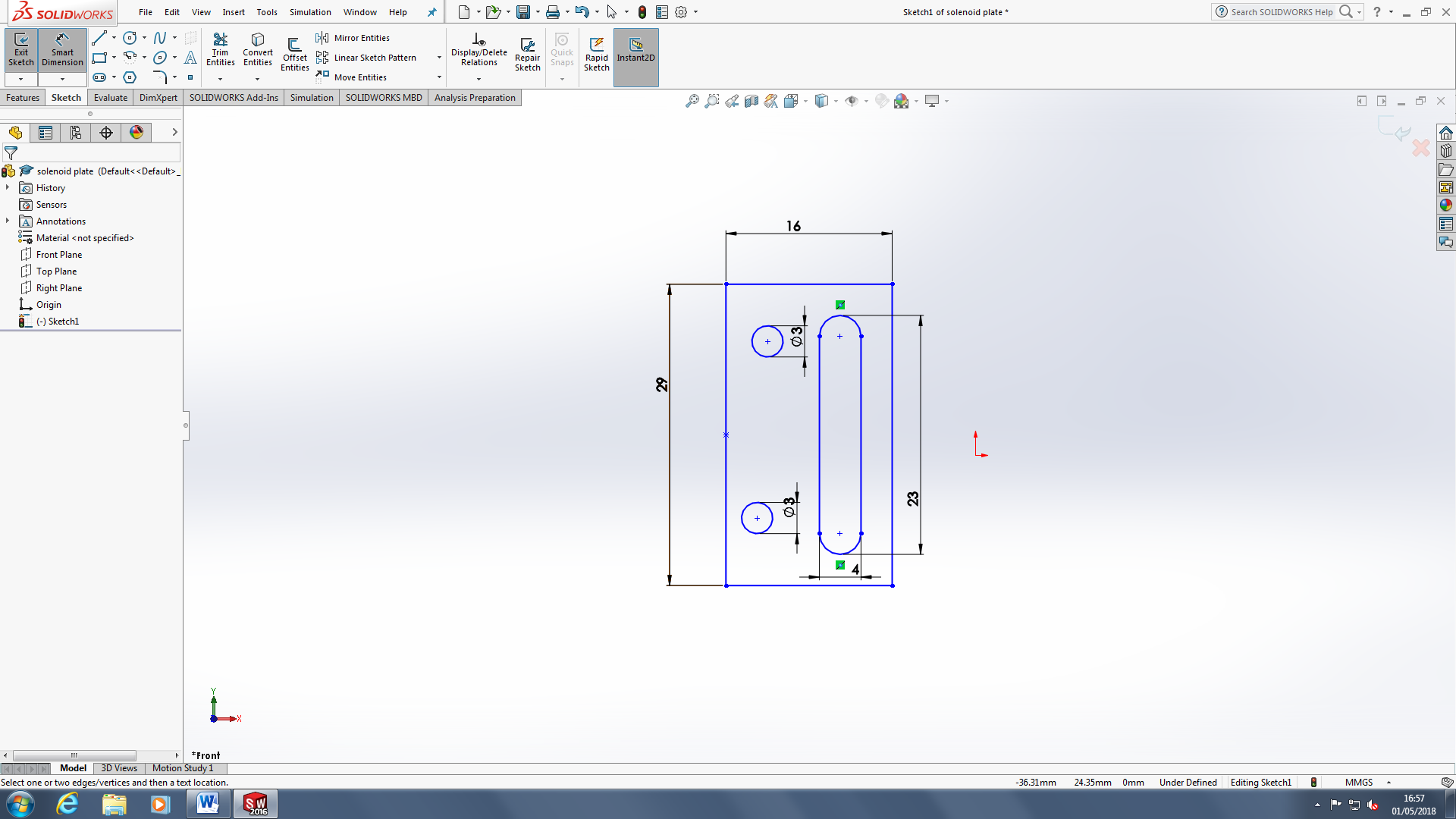


Figure 10.5: Solenoid bracket iterations

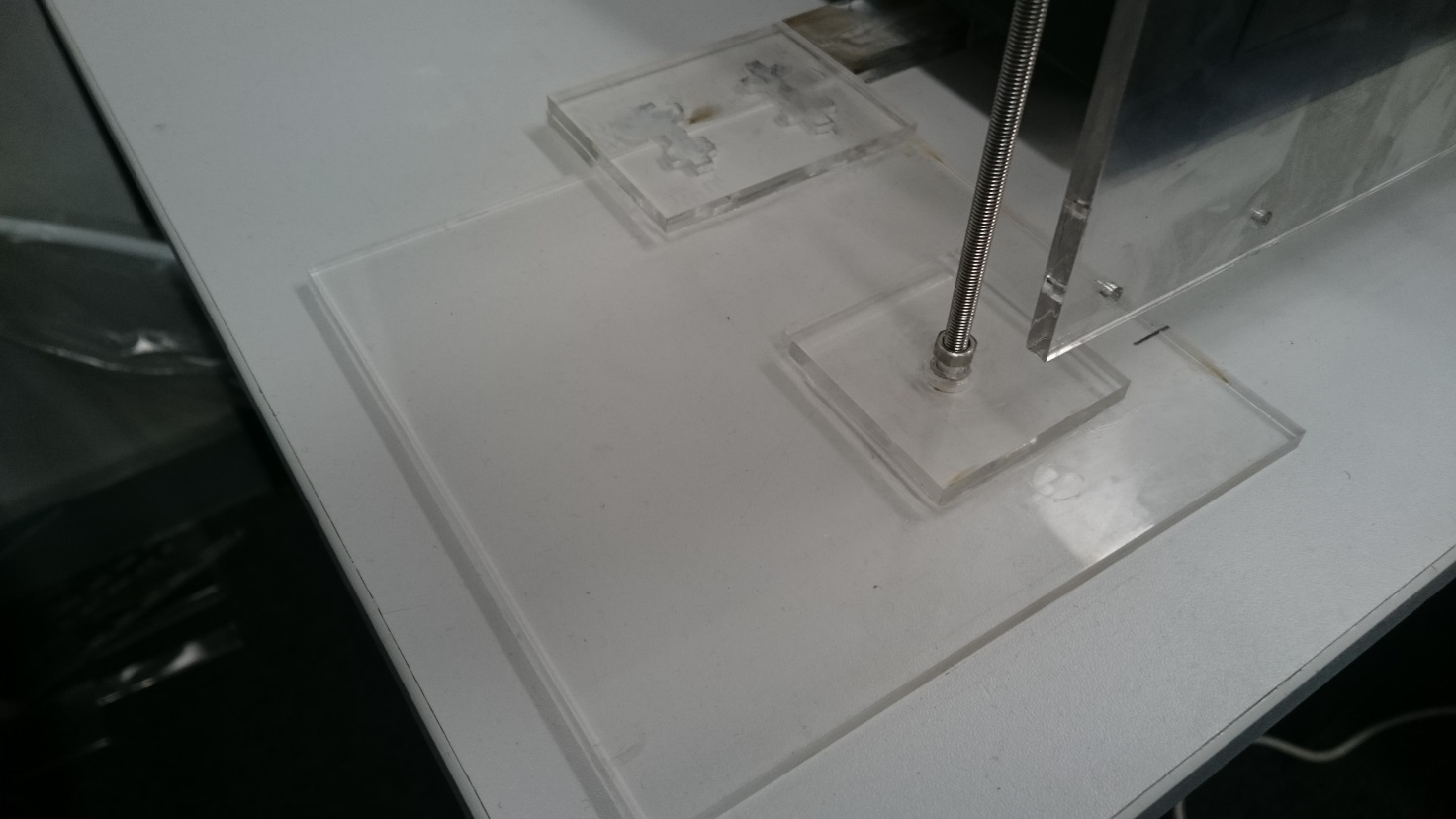
### Solenoid PCB design

The solenoids are controlled by the shift registers which use the Teensy’s 3.3V supply as the solenoids require 12V a transistor is needed to switch the power supply to the solenoids on and off. 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 Teensy can provide 250mA with its 3.3V power rail, a worse case of 8 solenoids playing at the same time would split the current into a maximum possible 31.25mA per transistor. So the gain required is 8.3. 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 8A and can provide a gain of 1000 when Ic=0.25A. The circuit is shown below figure. The equation is Rb=(Vin-Vbe)/ib =(Vin-Vbe)/ic/Hfe=(3.3-1.5)/0.25/250=1.8kΩ. This calculation contained a mistake as the gain is actually 1000 at Ic=0.25A however, as the real gain is higher the circuit still provides a collector current to meet the requirements of the solenoid.

### Supporting the Bosch Bar

It was decided to support the Bosch bar using two threaded rods located at either end as shown in **Error! Reference source not found.** 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 which allows a hole for the threaded rod to go 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 10.6 and uses a two Perspex pieces one for which the threaded rod is attached using two nuts which clamp the Perspex tight to the threaded rod (label 1 in Figure 10.6). 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 needed to be thicker than 3.5mm and the hole needs to be at least 10mm wide. 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.



Large Base plate

1. Plate clamped to threaded rod with two nuts

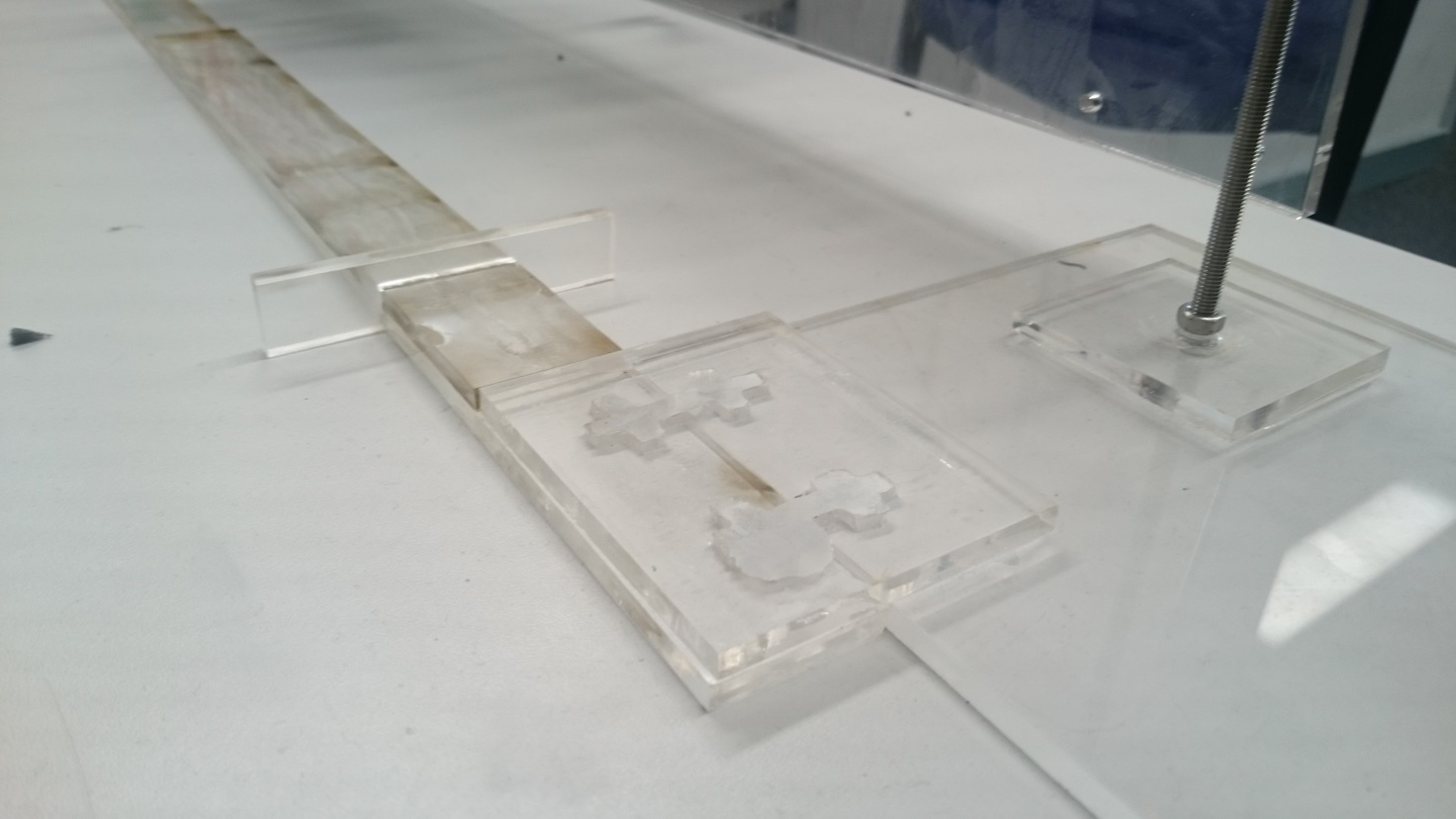
Connection to the Perspex bar that connects both feet each end of the Bosch bar

Threaded rod

Figure 10.6: keyboard solenoid rail stand

With the two stands at either end of the keyboard there is a possibility the stands will be knocked and move the solenoids out of position. A bar was designed to connect the two stands together. There is a channel running the length of the keyboard in which a bar can be run to connect the two Perspex stands together. It is 60mm away from the edge of the keyboard and the stands can be located at any distance from the keyboard as the lengths of the threaded rods holding the solenoids can be made any required length. However, to keep the design compact the rail should be located a maximum of 10cm away and a minimum of 2cm away to allow for room to mount the threaded rods. Ti was decided to locate the rail 4cm away from the keyboard as it is a compromise between the two. 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 locating the threaded rod 100mm from the edge of the channel running the length of the keyboard. The top plate which clamps the rod in place was 60x60mm with a 5mm hole in the centre.

The bar connecting the two stands can be seen in Figure 10.7. 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 four lengths of Perspex two 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 other two were 29xxxmm these were glued directly on top of the other two bars to create a bar 10mm in height to make sure the bar was secure in the channel. The bar was attached to the two supports using an overlapping 60x85mm plate at either end to glue the pieces together. Initially the stand and the bar were connected using a screw and nut (the locations can be seen in Figure 10.7) however, this method was found to loosen over time. Two stopper pieces of Perspex were made to stop the keyboard sliding along the bar the design is shown in Figure 10.7. This means the solenoid rail is held in place by the keyboard.



Threaded Rod

Connecting Bar

Keyboard stopper

Stand and connection bar attachment plate

Figure 10.7: Stand and bar connection

Previous screw connection points

### PCB Support Plate

The instrument has 8 shift registers, 25 transistors and 1 Teensy PCBs it was decided to mount them on the Bosch bar using a Perspex plate to display them. 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 size limits of the plate were set at 1100 mm long and 120 mm wide so it would fit neatly along the front of the keyboard. The transistor PCBs were mounted along the top to make wiring up the solenoids easier, to fit all the PCBs on they had to be mounted as a double layer (shown in Figure 10.8) as each PCB is 25mm wide and with a 3mm gap between PCBs there was space for 1100/28=39 PCBs not 61. The shift registers are mounted below these in the middle of 8 transistor PCBs that they control 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.

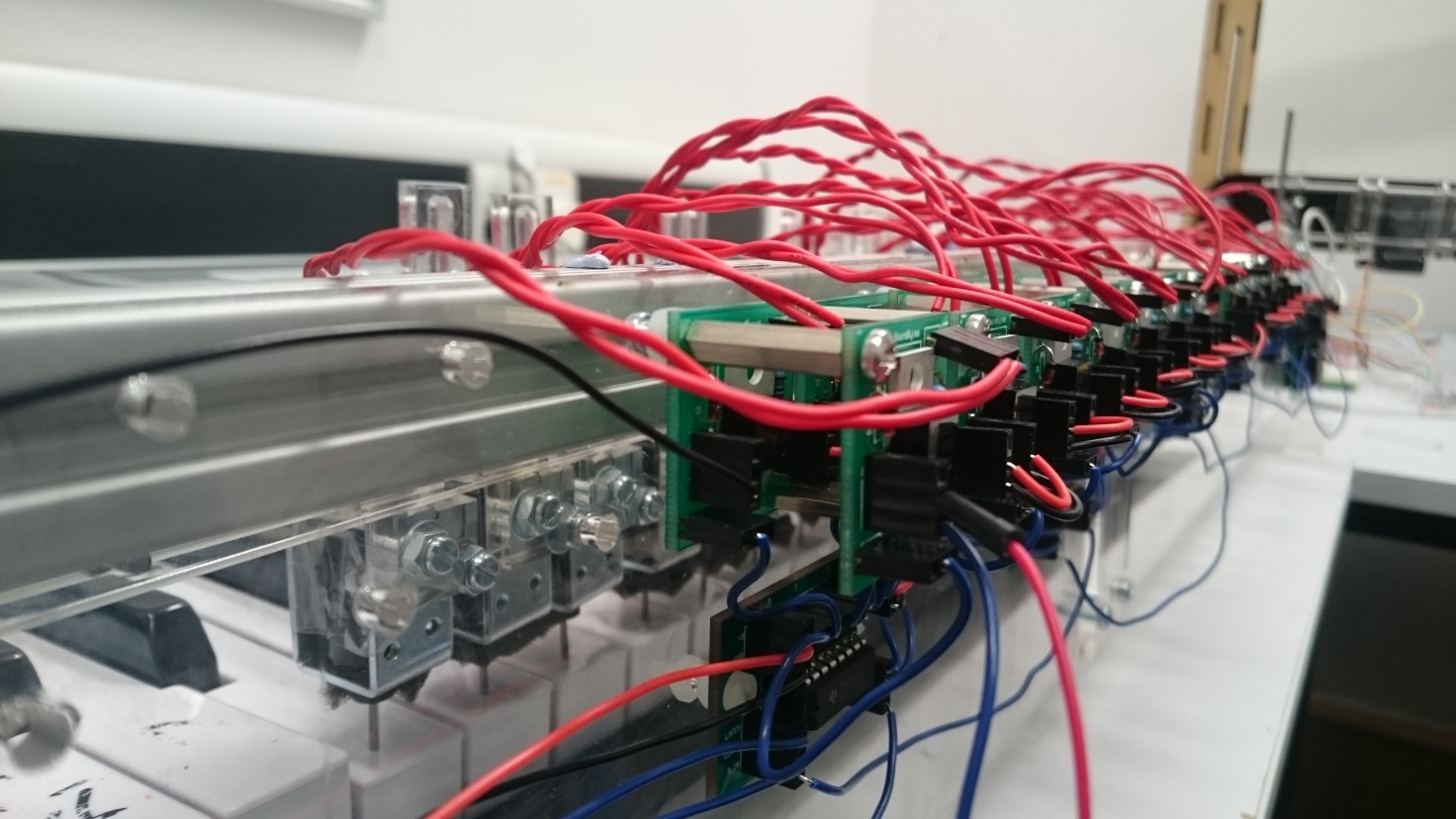


Figure 10.8: Transistor PCBs mounted on the Perspex plate

## Keyboard Software

The keyboard is driven by a Teensy 3.5 [39] connected to eight SN74HC595N shift registers [40]. Initially, an Arduino Uno was 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, 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. In this project, only 25 out of the 61 keys will be needed to play the two songs. However, the software will be designed so that songs containing more than just the 25 keys can be played without having to change the software, just adding the additional solenoids. As described in the previous section, each of the solenoids are connected to a transistor circuit, where each transistor circuit is connected to the output of one of the shift registers.

### SN74HC595 Shift Registers

There are four ways that shift registers operate, the SN74HC595 is an example of a serial in-parallel out shift register (SIPO). The other three are, Serial-In to Serial-Out (SISO), Parallel-In to Serial Out (PISO) and Parallel-In to Parallel Out (PIPO).

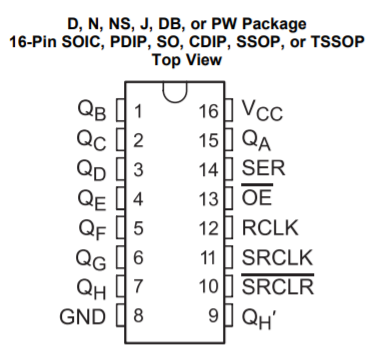


Figure 10.9 Schematic for SN74HC595 shift register from Texas Instruments [40]

From Figure 10.9, 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.

In this way the shift registers allow for control of more output pins that are available on the Teensy. Several shift registers can be connected in series with one another to act as one large shift register, increasing the number of outputs in multiples of eight. This is done by connecting the *Serial Out* pin (QH’ – pin 9) from one shift register to the *Serial Input* pin (SER – pin 14).

To cover the full 61 notes available on the keyboard, eight shift registers will be needed. This means that, since each solenoid represents a specific key, they must 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. The shift registers will be represented by a 64-bit variable in the software, where each bit of the variable represents a specific solenoid and note. For example, bit 0 of the of the 64-bit variable represents the lowest note on the keyboard, which is the MIDI note 24.

The latch feature will be very useful when the keyboard must play a chord (multiple notes played at the same time) as it means the solenoids will be pressed down at the same time. If the commands for each note in the chord were sent sequentially then it would not sound like chord, but a series of notes played quickly after each other.

### 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 which will have to be written to the shift registers one after another. 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, meaning that the shiftout() function (the function that writes to the shift registers) would need to be called eight times. This means that more instructions will have to be executed, which needs to be taken into consideration when synchronising different instruments together.

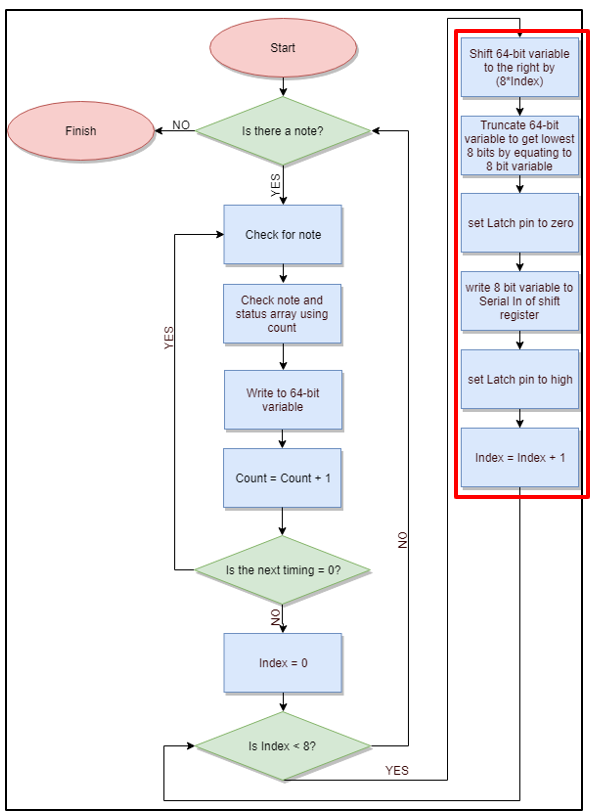


Figure 10.10 Structure of keyboard software that occurs within an Interrupt

Figure 10.10 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. 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() [41] function found in the Arduino library is used. The bitWrite() function takes three variables, the variable which will have its bit changed, the number of the bit to change and the status to be written to that bit. Each of these parameters is represented by 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 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. If it is a zero, it updates the 64-bit variable using the bitWrite() function. Once the loop is complete (i.e. when the program detects that the next element in the time array is non-zero), the program updates the interrupt timer to the value of the next 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 it into eight 8 variables and write 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 10.10) 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 high, 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 specify 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 was capable of doing 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 it own, was performing as expected.

## Summary

The keyboard software utilises a chain of eight shift registers, totalling 64 output pins to cover the 61 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.

## Xylophone

### Xylophone Software

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 within 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. The UART (Universal Asynchronous Receiver/ Transmitter) is a piece of computer hardware (a microchip) which translates data between parallel and serial forms [42] [43]. It is mainly used for serial communications that contain a receiver and a transmitter; each clocked differently [44].

Focusing on the program structure, a flat sequence is used which has two frames. The use of the flat sequence is to ensure that a sub-diagram/ frame executes before or after a sub-diagram/ frame. The first frame contains the interrupt which may occur while running the program, whereas the second frame encloses the main program. In more detail, the interrupt is directly connected with the UART of the MyRIO and therefore when the UART reads a command, the interrupt is triggered. The commands sent by the WiFi module are two: “STOP” and “PLAY”, where the second is followed by the song which needs to be played, for instance “PLAYGoT” if the Game of Thrones theme song is selected. In order to make the UART interact with the rest of the program, two global variables are used; the first one contains the action that need to be performed by the xylophone, i.e. “PLAY” or “STOP”, whereas the second one has the song which need to be played.

Within the second frame, a while loop is created which repeatedly executes the main program. This while loop is fed from the first global variable which contains the action, and therefore either stops or plays the specified track. Inside the while loop, a case structure exists which acts according to the action command 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 second global variable comes to use and informs the program which song needs to be played. Additionally, the appropriate text files are loaded from the c file directory of the MyRIO and 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).

Since the xylophone has only 12 notes, the next step of the procedure is to minimise the number of notes used in the songs. 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 while loop is used, which runs that many times as the number of notes in each song. Inside the while 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 (Figure 10.11). However, the while loop will only terminate if one of the following two cases occur. The first one is when the number of iterations of the while loop exceeds the number of the notes in each song and the second, is when the UART reads the value “STOP”, where it forces the loop to stop executing and start again when the command send to the UART (from the WiFi module) is “PLAY”.

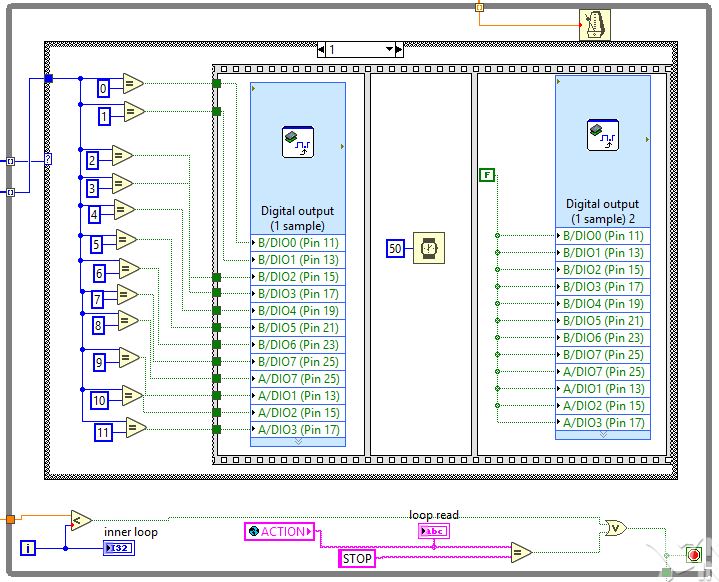


Figure 10.11: While loop which enables solenoids according to status

## Stepper Motors

The stepper motors were chosen as an instrument as they were able to play each musical note without requiring hardware modifications and as they offered a low-cost option for a musical instrument to be added to the orchestra. The stepper motors instrument contributes to the overall project objectives as follows:

* It can play all the required tracks for the selected songs (objective X)
* It is easy to transport
* It can communicate with the conductor

### 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 [45] 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 [46] 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 [47]. 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 [47] 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 sound level produced with materials such as Perspex, wood and cardboard, the data presented in **Error! Reference source not found.** has 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 10.12

|  |  |
| --- | --- |
| **Material Used** | **Measured Sound Level (dB)** |
| Perspex | 48 |
| Wood | 52 |
| Cardboard | 46 |

Table 7 Sound Level measurements for different materials

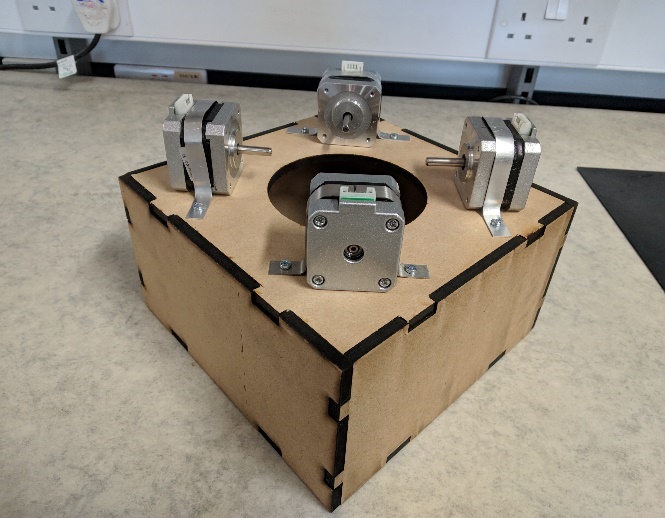


Figure 10.12 Acoustic box built for stepper motors

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. (photo to be added after the pcb is manufactured)

### Digital Signal Synthesizer

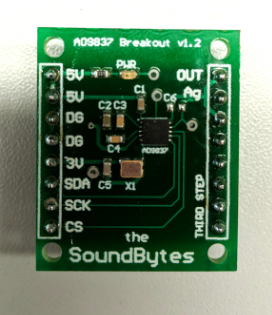


Figure 10.13Digital Signal Synthesizer PCB

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 [45]As the frequencies in the audible range only go up to 20 kHz [48], 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 10.13 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 [47]. This is because this model of driver board was recommended by multiple sources and according to [47], 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.

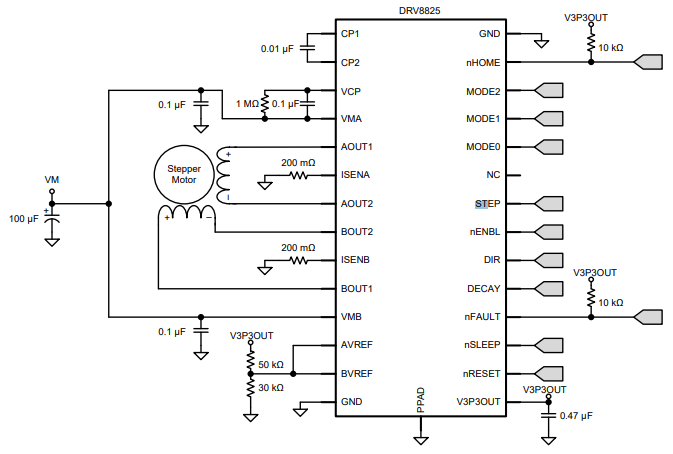


Figure 10.14 Schematic for DRV8825 IC [46]

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 [49] 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).

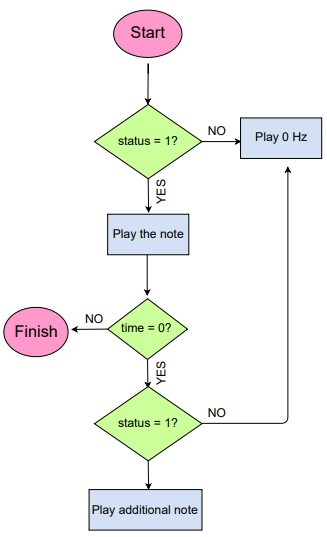


Figure 10.15 Flowchart of stepper motor software

Figure 10.15 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.

### Stepper Motors Testing

In order to confirm that the stepper motors instrument was designed properly, a series of tests was run. The first test consisted of a program designed for Arduino that would send square waves of different frequencies directly to the driver board of the motor. This confirmed that by varying the frequency of the pulses, the stepper motors could produce different musical notes. Following, a test code for the DSS IC was written in order to confirm that the IC could be programmed with a specific frequency to generate with high accuracy which could then be further transmitted to the motors. Next, the *Game of Thrones* theme song was played on one stepper motor and timed in order to compare with the duration specified by the MIDI file. As the instrument finished the song at the time specified by the MIDI file, it was confirmed that the stepper motors instrument was performing as intended.

Additionally, a test code was written for two DSS ICs to run in parallel in order verify that the instrument could play different notes at the same time. Figure 10.16 presents the output of two DSS ICs that are playing at the same time which confirms that the stepper motors instrument is able to play different notes simultaneously.

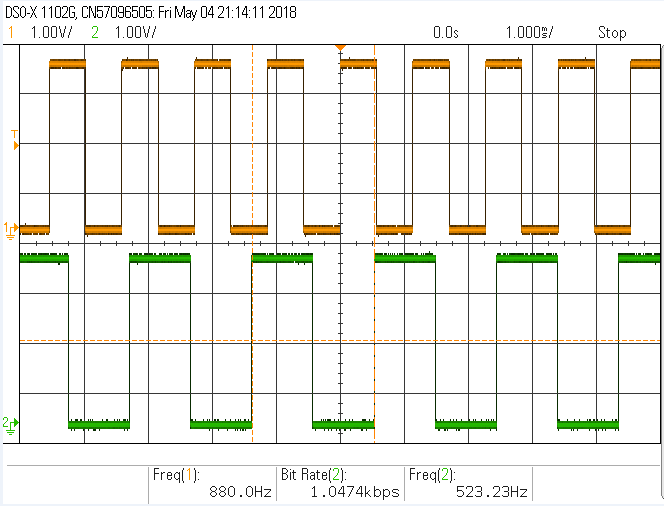


Figure 10.16 Signals of independent motors playing simultaneously

### 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 and Panpipes

The selection of the fourth instrument came down to two either a tesla coil or panpipes. Both designs were considered and feasibility testing for each instrument was done in parallel with a deadline of the 14th of February set to make a final decision.

The selection of the fourth instrument came down to two either a tesla coil or panpipes. Both designs were considered and feasibility testing for each instrument was done in parallel with a deadline of the 14th of February set to make a final decision.

### Panpipes

After testing was done both on the panpipes and Tesla coils, the Tesla coils were chosen to be the fourth instrument for reasons that are later discussed, 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 10.17 First design | Figure 10.18 Second design | Figure 10.19 Third design |

**First design**

The initial design seen in Figure 10.17 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 10.18 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 10.19 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 pipes providing airflow to each of the 15 panpipes and at the end of each airflow pipe 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. Plumbing fittings would be used to distribute the pressure and airflow between all the panpipes.

### Research and testing for the tesla coil

A tesla coil is a transformer with a high turns ratio between the primary and secondary coils to produce a high voltage over the secondary coil. One end of the secondary coil is left as an open circuit and when the voltage goes above the dielectric breakdown of air 3kV/mm [50] it produces a spark. This spark can be modulated to change amplitude and frequency to vibrate the air and produce music.

As mentioned in the literature review there are many examples of a tesla coil being used to play music with producing streamers with lengths greater than 1m. It was decided that building this type of tesla coil would not be feasible in this project. Firstly, the testing would have to take place in the High Voltage lab, which was undergoing improvement works, so the remaining test spaces were under demand and would only leave a 3 week block to complete the testing when the hardware was complete (ref gant chart??). Secondly, The cost of building a lager one could also be a limiting factor as one estimate to build a small tesla coil was £300 [51]. This estimate doesn’t include a switching circuit to play music and issues in development, adding a further 50% to account for these would increase the cost to £450 which is which is 30% of the budget. Finally, the other factor to take in to consideration is that the orchestra is to be taken to outreach days and having a high power tesla coil could be difficult to meet health and safety regulation. Due to these issues it was decided to buy three low power mini tesla coil kits available from amazon to test.

The tesla coil kits included a circuit diagram with an explanation however 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, shown in Figure 10.20.



Figure 10.20 Circuit diagram for Tesla Coil

The circuit works as a high frequency oscillator where Q1 and Q2 switch on and off which creates an approximate sinusoidal waveform Figure 10.21 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.

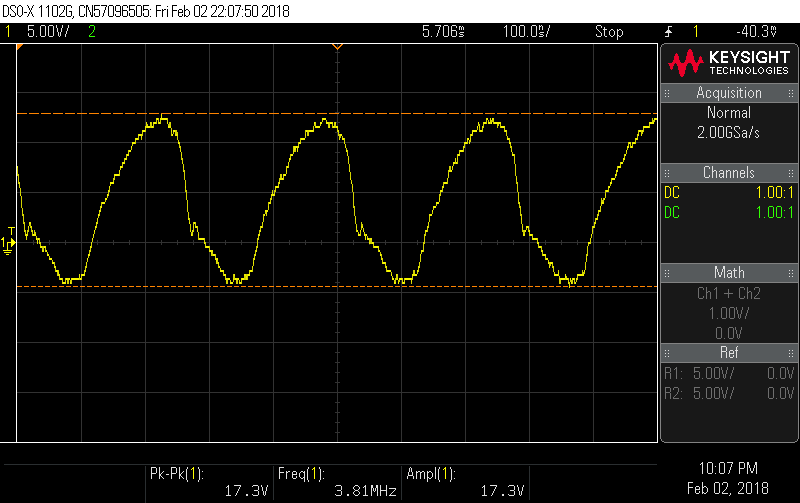


Figure 10.21 Single oscillation over the primary coil

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

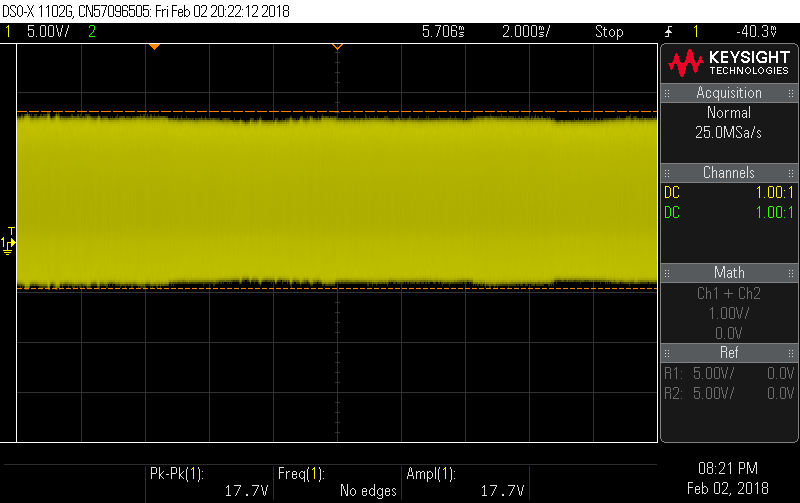


Figure 10.22 Voltage waveform over the primary coil with no music playing

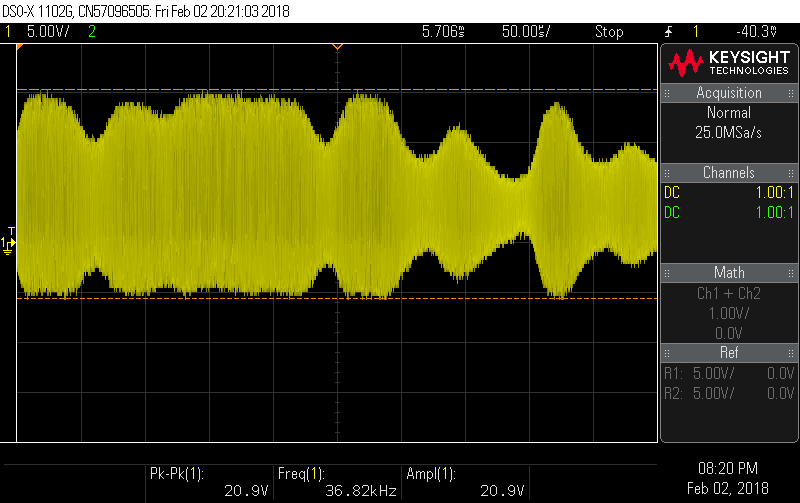


Figure 10.23 Voltage waveform over the primary coil with ‘eye of the tiger’ playing

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

The maximum current in the primary coil is estimated to be 30mA. As the base resistor is R4=10KkΩ and the voltage measure over R4 varies between 16 and 22V so the maximum current is 2.2mA. The gain of the BD243C is 100 so the collector current is 220mA in the primary. In the secondary coil the current is 220mA/350=0.6mA. Therefore, the current in the secondary coil is not dangerous.

The peak to peak voltage over the secondary coil is estimated to be about 6000V as the breakdown of air is 3kV/mm the length of the sparks are about 2mm long. As the spark from the is small that is also not going to cause any serious damage and can be controlled by putting the coil into a box, so it cannot be touched.

### Issues found while testing.

After continued use the heat sinks become hot and the volume produced by the coil reduces 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 tesla coil for more than 3 minutes.

The sound produced by the tesla coil is currently not that loud at only 55DB (using a phone app). This is about the same as conversational speech [52] as can be seen in Figure 10.23 the amplitude of the sound is controlled by the voltage amplitude.

### Possible Solutions

To deal with the heating problem new heat sinks that can deal with the heat better could be selected. The choice will be discussed in the Tesla coil section

To increase the volume the voltage over the secondary coil needs 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 would be possible 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 would increase the length of the streamers to 10000\*20/3000=67mm=6.7cm. 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 be able to reach the PCB. So to remove this problem the PCB would have to be split so the switching circuit can be removed from the coils 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.

As stated in the previous section the testing of the new coil would have to take place in the high voltage lab discussion were had with Dr. Vidyadhar Peesapati however, due to improvement works and demands on the remaining test spaces the team was unable to get access for testing in the time available. Therefore, this could be possible future work to increase the size of the secondary coil. For this project the solution decided on was to explore the use a microphone and play the sound through a speaker. This will be discussed in the tesla coil section.

### Summary

Using the mini tesla coil circuit would be cheaper as the components cost around £10 and safer due to the low current and small sparks (2mm). It would also fit within the time limits of the project as there is already a working circuit so with some developments it will be a viable option.

### Decision between Tesla coil and Panpipes

## Tesla Coil progress

The developments needed for the tesla coil to become a part of the orchestra are as follows: the heat management needed to be improve so it can be played for longer; the tesla coil needed to be integrated with a DSS and Teensy to control it; a new PCB needed to be designed to meet these requirements and a box is to be designed to mount the tesla coil this will make it easier to transport and provide a safety feature as the spark will not be able to be touched. The components will also be specified to cope with higher voltages and currents so in the future the circuit could be adapted to work with a larger coil. The current power supply used to power the tesla coil is 18V capable of supplying 2A so the maximum current the circuit can supply to the primary coil is about 2A.

### Heat sink calculations

The heat sinks were over specified with development with larger coils in the future. Power dissipated as previously calculated the current in the transistor is estimated to be about 0.22A and the voltage Vce= 2V so power is P=0.22A\*2=0.44W

Equations

Δθ =R\*P

Where P is the power dissipated in W, R is the thermal resistance in ⁰C/W and temperature difference between maximum temperature and ambient Δθ in ⁰C.

Δθ =(Rjc+Rca)\*P

Δθ =(Rjc+Rch)\*P

T=70\*0.44+25=55.8⁰C

The temperature specified by the British government state that surfaces acsssible to touch should not exceed 43⁰C [53]. Along with this touching a surface for 17s at 55⁰C will cause second degree burns [54] to avoid this a heat sink was chosen to aid heat loss. The maximum temperature for the surface of the heatsink was set to 30⁰C to ensure ther will be no risk of burns.

Using equation ? 30-25=0.44(2+Rhs) therefore, Rhs<9.36⁰C/W. To meet this requirement the …. was chosen as it has a thermal resistance of 3.4 ⁰C/W. If the circuit was used for 2A the temperature of the heat sinks would become 25+2\*2\*(2+3.4)=46.6⁰C. Which is above the governments recommended maximum temperature so the circuit would have to be contained in a box so it can’t be touched.

### Component specification

Two new PCB’s were made the components chosen for this are specified below:

BJT requires: a colector current >2A; =2V, >24V, DC current gain at 40mA≈100, Switching frequency>4MHz, TO-220 package type.

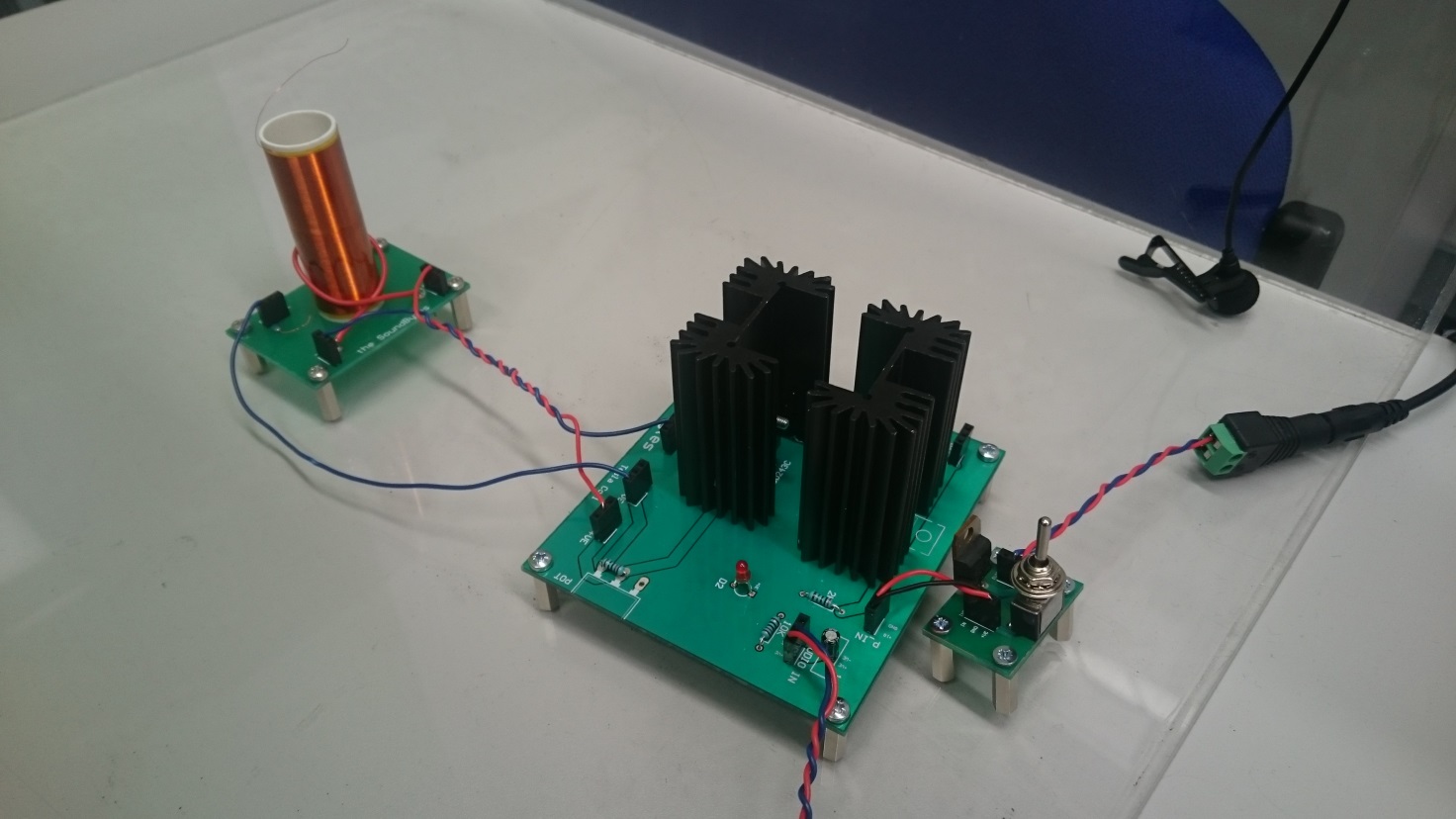
MOSFET requires: Vgs=4V, Vds>24V, Id>2A, Switching frequency =4MHz, TO-220 fitting

Capacitors: C2 is 1uF with a voltage rating larger than 25V, C1 is a 1uF electrolytic capasitor requiring a voltage rating of >5V

The components except for the heat sinks, BJT and the MOSFET were obtained from stores. The selected BJT was the BD243C and the selected MOSFET was the STP75NF70.

### Testing PCBs

Two new PCB was designed to house the new components and are shown in Figure 10.24. One is the circuit producing the oscillations across the primary coil and the other is for the secondary coil. this will allow a larger tesla coil to be exchanged for the current one in the future. During testing several issues were found with the PCB. There was a connection missing to the collector of the BJT (Q1) and it was also found that the circuit provided in the tesla kits was not the same as the PCB layout used in the test kits. This meant that the position of LED1 was changed from the positive terminal connected to the secondary coil and with the negative terminal connected to the base of the transistor. To the positive side connected to the transistor base and the negative to the ground leaving the secondary coil connected to the base of the transistor (the new circuit can be seen in figure) and the old in the appendix).



Tesla Coil

High frequency oscillation circuit

Microphone

Power Switch

Figure 10.24 Tesla coil instrument

These errors were fixed and a new PCB was printed. An alternative method was explored to so it could be used with the Teensy. So, if the new PCB still had faults when reprinted this would allow the instrument to be finished and tested with the orchestra and mitigate the risk. There was also switch for the power-in made so the tesla can be turned off in-between songs to limit the amount of pwer the heatsinks have to dissipate. This includes a manual switch and a PNP transistor that can be turned on and off by the microcontroller. The TIP126 PNP transistor was one available from the school stores and was rated for 80V and had a gain of 1000 so, to make sure the transistor allows 2A to flow a resistor to give a collector current of 2.5A was set at 1.2kΩ

A Teensy microcontroller board and DSS was used to control the tesla coil so it 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.

To amplify the tesla coil a microphone was chosen pick up the sound. This utilised an amplifier circuit already designed for the xylophone to amplify the microphone output so it could be connected it to the speaker. The tesla coil will be housed in a box to keep it from being touched by observers, so it can be used safely as part of the orchestra. The health and safety document for the coil is in the appendix.

### Testing

### Summary