Robotic Whistle

Final Report for CS39440 Major Project

Author: Lukasz Medza (lum22@aber.ac.uk)

Supervisor: Mr Dave Price (dap@aber.ac.uk)

08/05/2017

Version 2.0 (Release)

This report is submitted as partial fulfilment of a BSc degree in
Computer Science (Inc Integrated Industrial And Professional Training) BSc (G401)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Declaration of originality and Consent page removed for the purpose of sharing. It includes a signature on it.

Acknowledgements

I am grateful to my supervisor Dave Price for overseeing the project and sharing his wealth of wisdom on robotic orchestras. I am also grateful to Peter Todd, Barry Thomas, and David Lewis for their assistance in building the physical device.

I'd like to thank my closest companions; Adam, Stewart, Eleanor, Steffan and Jonathan for continuously motivating me and keeping me sane throughout the project.

Abstract

This project is a robotic approach to playing music on an analogue instrument typically played by humans – the tin whistle. The majority of the work focuses on the software component, though some work has been put into the hardware aspect as well.

Running on Linux, the program is built around the integrated ALSA framework which forms a part of the Linux kernel. The program takes in user input in the form of a MIDI file and translates it into instructions for the physical device to execute. The device possesses the ability to recognise the pitch and dynamic of each event and calculate which notes are playable on the physical device. The system accounts for unplayable notes and provides an interface for forwarding unplayable events to an external MIDI synthesizer, thus giving the illusion of a seamless performance.

Execution of notes on the physical device is carried out using a servo board. A Python API provided to move individual servos. A separate, 'free play' mode is available to manually make calls to the API for testing, calibration, and maintenance purposes.

Contents

Figures 8
Tables 8
1. Background, Analysis & Process 9
1.1 Background 9
1.2 Analysis 9
1.3 Relevant Literature 10
1.3.1 Robotic Orchestras 10
1.3.2 MIDI 10
1.3.3 Python and Unit Testing in Python 10
1.3.4 Testing Methodologies 10
1.4 Project Objectives 11
1.4.1 Use Case Diagram 11
1.4.2 Functional Objectives 11
1.4.3 Technical Objectives 12
1.5 Analysis of Technical Risks 12
1.6 Process 13
1.6.1 Upfront Design, Planning and Iterations 13
1.6.2 Board 14
1.6.3 Retrospectives 14
1.6.4 Notable Changes and Adaptations Made 14
1.7 Tool Selection 15
1.7.1 Programming Language 15
1.7.2 Operating System Platform 15
1.7.3 Development Environment 15
1.7.4 Version Control System 16
1.7.5 Documentation and Diagrams 16
1.7.6 Issue Tracking 16
1.8 Security 16
2. Design 17
2.1 Overall Architecture 17
2.2 High Level Design 18
2.2.1 Event Timing 18
2.2.2 Pitch Recognition 19
2.2.3 Range of notes physically playable by the whistle 19
2.2.4 Software/Hardware Interface 19

2.2.5 Volume and Air Control 20
2.3 Execution Algorithm 21
2.4 Other Features: 22
2.4.1 "Free Play" Mode 22
2.5 Software Design Summary 22
2.5.1 Class Diagram 23
2.6 Hardware Design Specifications 23
3 Implementation 25
3.1 Sandboxing and Early Implementations 25
3.1.1 Early Attempts at Note Translation 25
3.1.2 Early Attempts at Timing 25
3.1.3 Hardware Control 25
3.2 Major Refactoring of the Program 26
3.3 Development Sequence 27
3.4 Final Implementation 27
3.4.1 Servo Message Encoding 27
3.4.2 Note Translation 28
3.4.1 FreePlay Mode 28
3.5 Hardware Implementation 29
3.5.1 Hardware Compromises and Omissions 29
3.5.2 Photographs of the Physical Device 30
4 Testing 31
4.1 Testing Strategy 31
4.2 Exploratory Testing 31
4.2.1 Creating a Bug Classification 32
4.2.2 Test Charter 32
4.2.3 Test Review 32
4.3 Analysing the Usefulness of Exploratory Testing 33
4.4 Acceptance Testing and Unit Testing 33
4.4.1 Test Results 33
5 Critical Evaluation 34
5.1 Project Stages 34
5.1.1 Requirements and Research 34
5.1.2 Design 35
5.1.3 Implementation of Software and Associated Hardware 35
5.2 Testing 37

5.3 Methodology....37
5.4 Tools and Development Environment....38
5.4 Project Reflections....39
Appendices....40
Appendix A – Third Party Code and Libraries....40
Appendix B – Ethics Form....41
Appendix C – User Manual....43
System Requirements....43
Preparing the Hardware....44
Running the Program....44
Appendix D – Sample Test Charter....45
Appendix E – IEEE 829 Master Test Outline....47
Appendix F – Abridged Test Results....52
Annotated Bibliography....53

Figures

Figure 1 - Use Case Diagram 11
Figure 2 – Fabio's interpretation of the basic principles behind Musical Robots 17
Figure 3- Early high level design....17
Figure 4 - Execution algorithm for a single ALSA event 21
Figure 5 - Class Diagram 23
Figure 6 - Sketch schematic of proposed physical device....24
Figure 7 - A photograph of a very early hardware prototype featuring only a single servo and powered by batteries 30
Figure 8- A photograph showing a near completed version of the hardware. Note the pump encased in a wooden mute-box connected via a plastic pipe. The servos are connected to the servo board, visible in the plastic box. 30

Tables

Table 1- Servo controller message encoding....27
Table 2 - Sample bug selection 32
Table 3 - Breakdown of objectives compared against the final implementation 36
Table 4 - Exploratory testing log....45
Table 5 - Test Plan laying out pass and fail criteria 49
Table 6 - Results of the master test session 52

1. Background, Analysis & Process

1.1 Background

Ever since the self-playing player pianos of the 19th century, automated instruments aimed to educate the public in music, computing and robotics. Used on their own or in ensembles, robotic instruments form robust companions for composers, performers, and students of music; able to play indefinitely without mistakes or human errors.

Outside of computing the authors interests and background lie in folk music. Combining the two seemed like an interesting challenge from the computing side but also an opportunity to draw on previous knowledge from the music field. One of the big inspirations for this project was the work carried out by Dave Price on the 'robotic orchestra' for the BBC Royal Institution Christmas Lectures where a glockenspiel and an organ were adapted to autonomously play music from a MIDI file. [1] [2] The purpose of this project was to expand on Dave's work by adapting a new instrument for robotic control.

The tin whistle is an interesting instrument to tackle for this project because in comparison to other instruments woodwinds are rarely digitised. MIDI interfaces are often used (built in) to keyboards or as standalone devices on guitars but rarely are they found in use on whistles, recorders, and flutes. The nature of these instruments requires deceptively complex hardware to control the air flow and ensuring an air-tight seal on the finger holes. Overall, this makes the tin whistle a compelling challenge which on the surface might look deceptively simple.

1.2 Analysis

At a glance, the instrument in question (a tin whistle) seems overly simplistic; however, upon further research the potentially overwhelming depth of this project became apparent. Previous instruments in Mr Price's robotic orchestra have been inherently simpler; e.g. the glockenspiel/xylophone is tuned to only play one sound from each note. In practice this means that the only factors to take into consideration when translating it into a robotic format are a) which note to play and b) when to play it. In the case of robotic whistles (or other wind instruments) controlling the flow of air becomes a significant problem where regardless of holes covered a different note could be played depending on the force and pressure of the air pumped into the reed.

Previous attempts at controlling the airflow involved a complex process of measuring human blowing patterns using a quick response laminar flow sensor and then using the collected data to 'train' their robotic recorder. [3] The time and the technical resources required of this depth of detail were far beyond reach of a one-man team.

This meant that a comprehensive risk assessment was a critical requirement for this type of project to realistically assess which components could be implemented in the time given and what should be left for future development outside of the scope of this project.

1.3 Relevant Literature

As part of the initial research and analysis process, relevant literature was studied to gain understanding of the different components prior to the design stage of the project.

1.3.1 Robotic Orchestras

A lot of early reading was put into getting preliminary knowledge in the making of robotic instruments prior to starting any of the major planning activities. Works by Ajay Kapur (a leading figure in robotic music) formed the foundation of this project. [4] His paper on the history of robotic musical instruments provided valuable leads to other experts in the field such as Trimpin and Makoto Kajitani. [5]

Furthermore, M Fabio provides excellent commentary on high level design of robotic musical instruments; his work formed the basis of this project's software design. [6]

Additionally, deep insight was gained into the wind/volume control techniques on robotic woodwind instruments from studying the work of T Katyo et al. [3]

Analysing other implementations of robotic instruments and orchestras was critical to understand different approaches and schools of thought regarding this field and it helped to guide the design phase of the project.

1.3.2 MIDI

The use of MIDI was inherent to the subject. It was necessary to gain in-depth understanding of how MIDI works on the front end but more importantly how MIDI events are encoded and how programing with MIDI would work. Furthermore, it was vital to understand how MIDI interacted with the Linux operating system via the use of synthesisers and sequencers allowing MIDI traffic to be routed between software components and potentially MIDI hardware (e.g. a MIDI keyboard).

The official MIDI Manufacturers Association provides comprehensive specification documents which proved useful in understanding the backend operations [7]. Additionally, 'Ted's Linux MIDI Guide' presented a good introduction to MIDI programming [8]

1.3.3 Python and Unit Testing in Python

As the chosen language for the project some time was initially taken familiarising with Python. The 'Learn Python the Hard Way' guide proved a useful reference for checking new concepts. [9]

Furthermore, the official Python documentation website was helpful in transitioning from previous experience in JUnit testing into writing unit tests in Python. [10]

1.3.4 Testing Methodologies

A lot of reading was done into different testing approaches to determine the best fit for the scale, team size and duration of this project. Extensive reading was put into the theory behind testing strategies in general. [11] [12]

Having chosen a general approach to testing, more reading was invested into learning about exploratory testing [13] and IEEE 829 manual testing. [14]

1.4 Project Objectives

The objectives of the project are enumerated and explained in this section. Having studied relevant literature and examined similar implementations of robotic musical instruments, logical user stories were drawn and presented with a use case diagram in section 1.4.1. What follows is a set of functional and technical objectives decomposing the very high level user stories into individual objectives.

1.4.1 Use Case Diagram

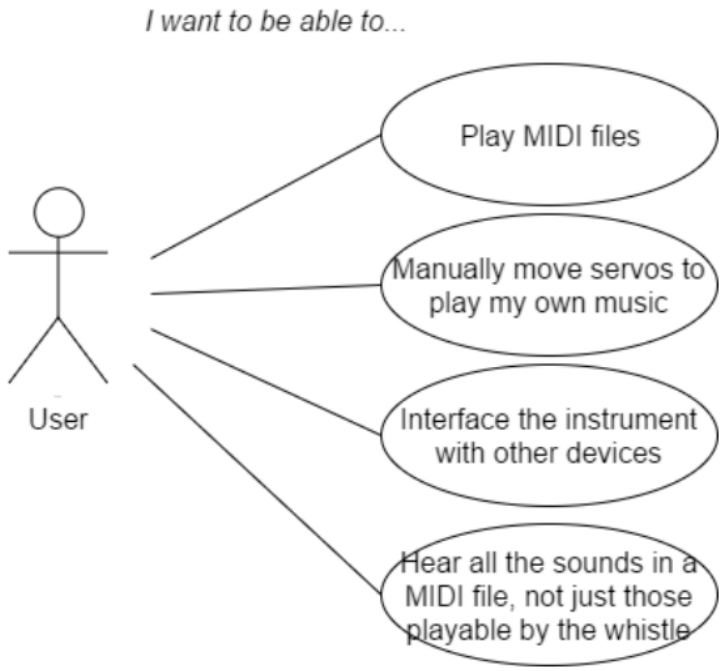

Figure 1 - Use Case Diagram

1.4.2 Functional Objectives

The functional requirements outline the expected experience of an end user

1. The program will accept input in the form of a MIDI file
   1.1. The file will be passed in from a command line interface
   1.2. The program should handle incorrect/broken files being passed to it
2. The program will parse the MIDI file to recover the MIDI pitch and timing values
3. Backend code will control the servo board, providing functions to move servos
   3.1. Functions should exist to reset all servos, move an individual servo, reset and individual servo and also to set the servo's speed

4. The program will determine the ability of the tin whistle to play input pitch
    4.1. The program will maintain a list of playable notes and compare incoming notes against it
5. Unplayable notes will be forwarded to an external MIDI device (synthesiser)
    5.1. A set channel number will be associated with the output letting the user chose their preferred synthesiser (be it virtual or physical)
6. Playable notes will be parsed for playing on the tin whistle by moving the servos
7. An extra servo will mute the reed of the whistle to adapt it for taking in a continuous flow of air

1.4.3 Technical Objectives

8. Open a serial connection to communicate with the servo board
9. Set up a MIDI throughput channel using ALSA to accept incoming traffic, and to forward any outgoing traffic to a synthesiser
10. Provide a way of switching modes (autoplay and free play)
11. Clear documentation of the code
12. Extensible architecture (to fit in with a larger robotic orchestra)

1.5 Analysis of Technical Risks

The key risks of the project stem from its inseparable association with the hardware. The primary purpose of the software is to work alongside the purpose-built device; without which the software becomes redundant.

Although for most parts outside of the technical considerations of this project (and report) the availability and functionality of the hardware potentially presents a major risk in achieving the objectives of the project mentioned in the previous section. In order to mitigate this risk, the project has been planned and design from the start to adapt to the hardware available at hand. Similar projects (see Mubot Recorder) [5] have been in development for over a decade giving them the time to perfect both the software and by extension the hardware design. Accounting for the timeframe of this project some design choices had to be made to make most of the hardware presently available at hand. Where possible suggestions have been made of ideal or best case scenarios where time and funding are not restraining factors.

Furthermore, it has emerged in the early sandboxing phase that hardware precision is of key importance to the project. Whereas a xylophone requires a simple strike or a MIDI keyboard requires a button press; the tin whistle needs precisely calibrated 'fingers' to cover its holes. Early experiments found that despite the MIDI note event translations calculating the correct pitch value the sound produced was not at all pleasant to a human listener because the holes were not being covered correctly.

In summary; additional work outside of software development is being put into maintaining the hardware side of the project to ensure that these associated risks are mitigated where possible.

1.6 Process

Choosing a relevant development model for this project was difficult as the possible work done was limited by the progress of the physical hardware construction process. Although the hardware construction falls outside the scope of this project it did play an integral role in driving and limiting the work that could be done on the software side. This meant choosing a model which can easily respond to change but at the same time puts producing usable code at its core. As a result; Scrumban was chosen to guide the software development process.

The key motivation behind Scrumban was its heavy focus on saving time which was particularly important on a one-man project. The intention was to focus on producing the software and the documentation of the project instead of creating detailed metrics which are only useful for improvement in the long term (far beyond the short scope of this project). By adapting on-demand planning (i.e. without calculating sprint estimates) more time can be focused on improving the overall quality of the produced code and the documentation.

In the spirit of agile software development; Scrumban combines the best elements of Scrum and Kanban [15] which have been adapted for the use with this project:

1.6.1 Upfront Design, Planning and Iterations

A desired feature of the development process was the ability to plan work in the short and long term. Following Scrum principles, the project was divided into two releases; the first between the start of development and the Mid Project Demo, and the second from the Mid Project Demo to the final hand in deadline. High level release planning was carried out before the first sprint of each release, focusing on breaking down the project objectives into end user use cases and objectives which the program should carry out by the end of that release.

Every sprint was benchmarked by its contributions to a) the number of tasks completed on the Kanban board; and more importantly the overall contribution made towards achieving a given project objective.

Although Kanban focuses on continuous development, time is taken to generate new stories. Weekly sprints were borrowed from Scrum; taking place every Tuesday to coincide with the weekly meetings with the project's supervisor. During these Sprint planning sessions; the chosen project objectives for that release were broken down into individual programming tasks. These in turn were used to populate the Kanban board.

A desirable feature of Scrumban is that it doesn't use any time estimates. This was important to give the project flexibility and adaptability which it needed when dealing with all new technology. Estimates can often be wasteful because they don't consider emerging new requirements and often estimates can be made for items which will never be developed. Following Kaizen principles, waste was attempted to be kept to a minimum which was well aided using Scrumban.

1.6.2 Board

A key feature of Kanban; the Kanban board creates a clear visual workflow. This was important to help keep all aspects of the project on track including those out of the direct scope of software development such as the hardware construction and other deliverables on this project. For ubiquity and easy portability, a digital Kanban board was implemented using Trello [16]

The columns on the board were broken up into Current Sprint (Not Done), In Progress, Done, Out of Scope and Future Sprints where items were spontaneously added as they emerged over the course of the project. During Sprint planning sessions stories were taken from the Future Sprints column and added into the Current Sprint column from where they were pulled in to 'In Progress' as required.

A defining factor of a Kanban board is the limit on the number of items in progress. In the case of this project, the limit was at first arbitrarily set to 3 items and this seemed to prove an effective number to keep the balance of work realistic.

1.6.3 Retrospectives

As part of Kaizen (continuous improvement) very short retrospectives were carried out as part of the weekly sprint planning sessions [17]. During these retrospectives, the completed stories from the previous week were analysed and the Sprint story card would ask and answer the following questions:

- What went well?
- What didn't go well?
- What could be improved?

The effectiveness of retrospectives within a one-man team can be debatable hence why they were kept short and to the point (no more than 20-30 minutes). In the long term, it's difficult to analyse their usefulness but they seemed to help keep the overall progress on track by addressing the most pressing issues.

1.6.4 Notable Changes and Adaptations Made

The most critical change which had to be made to a typical implementation of Scrumban was the adaptation of it for a one-man team. Agile development across the board encourages frequent face to face communication between team members; which was difficult on this project. Daily team meetings encourage making valuable changes to the project and ensuring that it is still on track to meet its desired goals.

In the case of this project most meetings were replaced with a 'review session'. As mentioned previously; weekly Sprint planning meetings and retrospectives were merged into a solitary sit down event. Daily Scrums were instead replaced with a daily look at the Kanban board. Sometimes this only took a few seconds to confirm what's been done and to move the relevant cards across but it helped to maintain good disciple throughout the project.

Similarly, it was impossible to implement the different roles typically associated with Scrum as all the work was being accomplished by one person. However, Kanban/Scrumban don't place much emphasis on team roles and code ownership which overall simplified the development process by avoiding trying to juggle wearing many different metaphorical hats.

1.7 Tool Selection

The toolset used for the project was deliberately kept bloat-free. Where possible the needs of the project have been examined, and various tools compared up against the objectives of the project to determine their use value.

1.7.1 Programming Language

The key task of this project (MIDI event translation) is done in real time; making it time sensitive. In practice this meant streamlining the code as much as possible to reduce processing time and naturally the need for a lightweight programming language has emerged making Python one of a few possibilities. To further argument in favour of Python, the language emphasises readability and its dynamically typed behaviour lends itself to the evolving design of this project.

Specifically, Python 2.7 was chosen for this project because the initially used (though later discontinued) Python MIDI Library was in a stable release only for Python 2.7 and not 3.

1.7.2 Operating System Platform

The program has been designed from ground up to work only on Linux in order to make use of the built in ALSA sequencer [18]. While Windows and UNIX (macOS and others) operating systems have their own sequencers, these are typically proprietary; making programming with them difficult and harder still to make one program cross-compatible across various platforms. ALSA on the other hand is open source and is an integrated part of the Linux kernel, making it easily accessible from any Linux machine.

1.7.3 Development Environment

Multiple different development environments have been tested for use prior to the implementation stage of the project. Cloud9 was initially chosen due to its constant availability and ubiquity thanks to running in the cloud/over the internet. While this was fine for some initial testing it wasn't very suitable for the physical interfacing inherent to the nature of the project; establishing a serial connection was tricky and the process wasn't very representative of the intended end use of the software (running on a ubiquitous device like a Raspberry Pi or a laptop). [19]

Instead the JetBrains PyCharm IDE was chosen for its complex toolset (including an integrated debugger, and many quality of life tools such as auto refactoring and integrated unit testing support). [20]

1.7.4 Version Control System

Drawing on previous experiences of using various VCS' in the past, the project was an opportunity to build on those experiences. Research was carried out to compare the two leading VCS's (Git and Mercurial) [21] , and no compelling arguments were found to choose Mercurial over the previously used Git.

A number of Git servers/hosts were also examined (notably Atlassian Bitbucket [22] previously used during an industrial year, and GitHub mostly used at University). While both offered similar features, GitHub was chosen in the end for its simplicity of use without the need to set up a dedicated local server combined with the offer of free private repositories for academic use [23]. Moreover; the chosen PyCharm IDE supports built in VCS support for Git/GitHub making fetching and retrieving data even more convenient. [24]

1.7.5 Documentation and Diagrams

Where possible, most of the simple diagrams in this report have been created in Microsoft Word using the built in 'SmartArt' tool which proved adequate at quickly creating linear diagrams. More complex diagrams were created in draw.io; a web tool specifically designed for creating free flowing diagrams. [25]

1.7.6 Issue Tracking

As specified by the project's development process, a Kanban board was a key feature required to plan and track stories, bugs, features and individual development tasks. An electronic board was favourable over a physical one as it could be accessed anywhere from any device. Two platforms were analysed; Atlassian JIRA [26] and Trello [16].

While more detailed and functionally advanced; JIRA did require a local installation or (more expensively) a cloud hosted server. Although technically basic and barebones, the functionality of Trello was sufficient for this project. The ability to create different columns and populate them with issue was all that was needed of the project which Trello was able to deliver on for free and without any installation.

1.8 Security

The security risk of this project can be classified as relatively low; with only potential for damage on the local machine where the program is running. With all the relevant libraries downloaded the program doesn't require to run on the internet (or any computer network) thus drastically reducing any potential security risks.

A possible security flaw comes from the need to use a Linux/Unix shell to interact with the program. With improper system settings, someone with access to the shell could inflict damage on the local machine. Preventing this from within the program would require performing fairly complex operations on the operating system which itself could open up new security risks. A simple approach would be to restrict user access on the

machine by maintaining proper access control systems in place. Overall, security had not played a part in the programs design or implementation but it was considered at the start of the project.

2. Design

2.1 Overall Architecture

The core of the program's design can be divided into two major parts; the software and the hardware. Even though the focus of this project was on the software engineering aspect, the hardware is an integral element of it without which the software essentially becomes obsolete. With this in mind, the high-level design of the project has always placed the hardware at its heart with the software accompanying it. Doing so ensured parity between the two components and helped to keep track of their progress in relation to the completeness of the project.

As identified by Michael A Fabio in his paper "Chandelier: An Exploration in Robotic Musical Instrument Design" all robotic instruments naturally follow the same high level design [6]:

Figure 2 – Fabio's interpretation of the basic principles behind Musical Robots

Building up on Fabio's principles a similar diagram was produced for this project during the early stages of sandboxing:

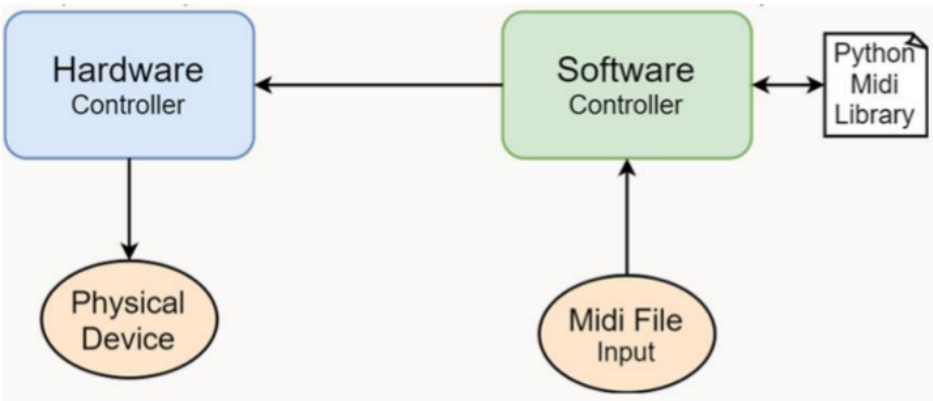

Figure 3- Early high level design

The above design shows the simplest possible way of modelling a robotic instrument; "given an input, the computer will translate it into a set of instructions and execute them by playing notes on the instrument". In a very simplified way the preceding sentence

summarises the scope of the entire project and ideally the minimum deliverable product that would constitute a 'robotic instrument'

The overall design was heavily influenced by some of the early technology choices which form the backbone of the program. Most notable was the decision to base the program off MIDI and subsequently use the Python MIDI Library [27]. Although the field of Robotic Music/Orchestras has somewhat matured over the past 30 years, it is rather niche with well-established trends. Having analysed different possible implementations of the above model, MIDI seemed like the logical choice to deal with sheet music to sound translation.

The immediate implication of choosing MIDI was the need to plan in advance how the program will parse MIDI files fed into it as input. Many programming languages exist solely for interfacing with MIDI, however a second language would then be needed to operate the hardware components of the project (i.e. the servo board). This made Python a very compelling choice to write the entirety of the project. Its simplicity and expandability meant that it was ideal for a project like this involving elements of both physical hardware and software.

However further into the development it has emerged that the Python MIDI Library was discovered to be redundant in the development of the program as the decision was made to shift the design from up front translation of events to real-time translation of notes as they arrived into the program. The decision involved the use of ALSA; the built in sound sequencer which comes pre-packaged with the Linux kernel. The sequencer would take control of timing the MIDI events and so all the relevant data (mostly the MIDI pitch) could be extracted directly from the ALSA event without the need to model a MIDI event in OOP as was the initial plan at the time of the 1st release (prior to the mid project demo) thus no longer requiring the Python MIDI Library.

Instead the Alsaseq Python Library was used to create a new ALSA channel and control anything that comes through it [28]. The design would allow the user to send a midi file straight from the command line by directing it to the ALSA channel:

aplaymidi –port 129 my_midi_file.mid

Which would then in turn be intercepted by the program and available for further parsing as described in the following sections.

2.2 High Level Design

2.2.1 Event Timing

Timing is an important part of every music experience. Every MIDI composition will contain instructions concerning the timing of MIDI events in the form of tempo (given in beats per minute), time signature, and the use of different notes (e.g. semibreves, minims, crotchets, etc…). The total number of different factors affecting the timing of musical events at any one given time is very large which would make timing events correctly a complex task.

The initial design for event timing on this project was to use MIDI's built in ticks which form the connections between MIDI events. In MIDI tracks the individual MIDI events (e.g. NoteOn or NoteOff) run in chronological order by maintaining an associated 'tick'

value expressed in milliseconds. The tick for any given event is the delta time in relation to the preceding event. For example, event1 will have a tick value of 0 (as there are no events preceding it). In turn, event2 might have a tick value of 200 because it occurs 200 milliseconds after event1. [29]

The early design of the project was to use these tick values to manually keep track of time and using the tick values time the events. That way the entire track would be pre-calculated in advance and then passed for execution all at once. In practice this was difficult to implement effectively without bugs due to the complexity of variables affecting the timings

Towards the second half of the project the final design was re-thought to deal with events as they arrive in real time rather than the initial approach of trying to calculate the note values in advance and prepare a playlist for execution. A similar approach was used in some of the studied works and while the initial setup barrier seemed more difficult, the end result was a significant improvement.

2.2.2 Pitch Recognition

Naturally, the bulk of the work on the project would be devoted to recognising the pitch of the note to play. In MIDI NoteOn and NoteOff events this is represented by a value between 0-127 which in turn corresponds to one of 12 notes in one of 10 octaves. [30]

In principle recognising the pitch of the note is a simple task, but major problems can arise:

2.2.3 Range of notes physically playable by the whistle

The tin whistle is a diatonic instrument meaning that it physically cannot play all 12 notes of the octave. Different solutions to this problem would drastically change the design and so had to be taken into consideration early on.

- An easy solution would be to only feed the program pre-prepared MIDI files which are definitely playable on a tin whistle. The program could analyse the MIDI files before execution and reject it if it was unplayable
- An alternative would be to execute any MIDI file but only play those notes playable on the whistle; other notes would be ignored
- Better yet; the program would play those notes playable on the whistle and substitute/synthesise those which are not. This design choice seems to make for the most 'natural' user experience as it exploits the whistle's potential but at the same time fills in any blanks (thus avoiding a forced silence).

2.2.4 Software/Hardware Interface

Having calculated the pitch and timing of the MIDI event the instructions would need to be passed on to the physical device itself. For the purpose of this project a Pololu USB 16-Servo Controller was used alongside 6 mini servos to cover the holes of the whistle

[31]. The device was very simplistic; accepting only encoded binary commands over a serial connection which meant that one of the first elements of design and implementation for this project was creating a frontend API which could be used to move the servos during the initial testing and the development stage of the project.

2.2.5 Volume and Air Control

Volume control is an embedded part of MIDI, expressed as a 'velocity' value. At its simplest, the velocity responds to the dynamics given in sheet music. On a wind instrument like the tin whistle the velocity would be controlled by varying the air volume/pressure entering the mouthpiece of the whistle. An air valve would need to be use along with a built-in pressure sensor to monitor the amount of air going in; this would need to be taken into account in the software when translating velocity from MIDI events.

However, the tin whistle itself is a more complex instrument when compared to other instruments in Mr Price's robotic orchestra. The xylophone for example operates in a binary mode where one of the metal plates responds to exactly one note that note can be either on or off. This is not the case with tin whistles where every note is not one key but instead a combination of keys. Furthermore, the same combination of finger presses would generate a different tone depending on the force of air blown into the reed.

The original design of this project accounted for air control as a necessary feature, but time and hardware constraints meant that this idea was put on hold and eventually removed from the scope of the project. Nonetheless, volume and air flow control form an integral part of the overall design and would make the next logical major feature to factor into design. If the project was taken further in coming years, work could be invested into expanding the current implementation to feature volume control.

2.3 Execution Algorithm

The behaviour of the program is summarised below with the use of a flow diagram. The diagram shows a snapshot of execution for a single MIDI event assuming that the starting point is a stack of events dispensed as input in real-time.

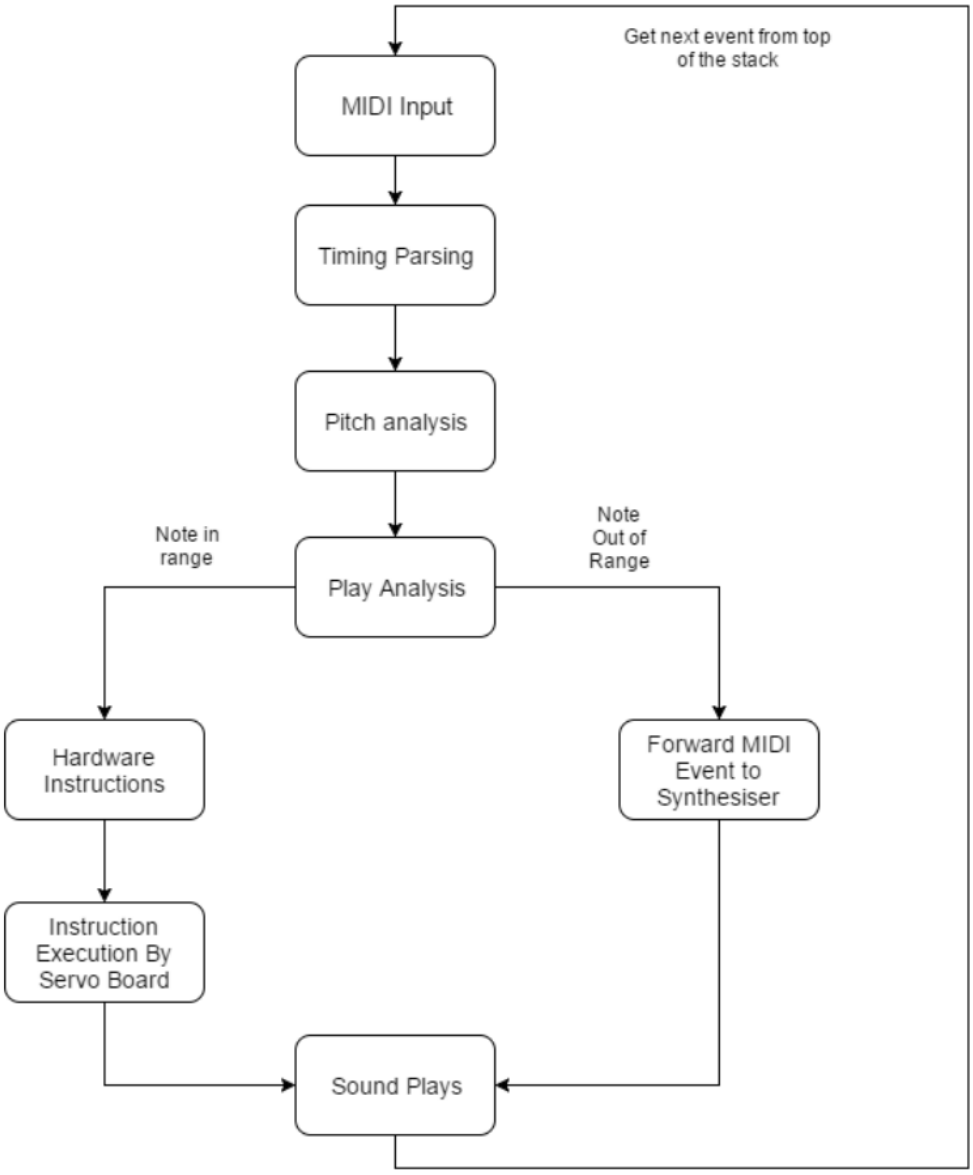

Figure 4 - Execution algorithm for a single ALSA event

2.4 Other Features:

2.4.1 "Free Play" Mode

The idea of Free Play Mode was to deliver a way for a human musician to manually control the robotic whistle. Pressing keyboard buttons would instantly move the relevant fingers up and down on the whistle.

The purpose of this mode was to add use value to the project beyond the basic functionality. In theory, this mode would be very easy to implement as the functions required to move individual fingers would already have been a perquisite for the Auto Play mode which meant that only a small chunk of code would have to be written to implement this feature.

In practice this mode was useful for debugging and calibrating the hardware and individual servos.

2.5 Software Design Summary

In conclusion; the design of software is based around a central AlsaMain class which acts as the 'hub' of the program. It forms the entry point where input is received and interactions are made with other components of the program. The overall design is summarised on the following page in a class diagram; note the presence of the theoretical PumpEvent class which would be responsible for controlling the airflow into the mouthpiece.

2.5.1 Class Diagram

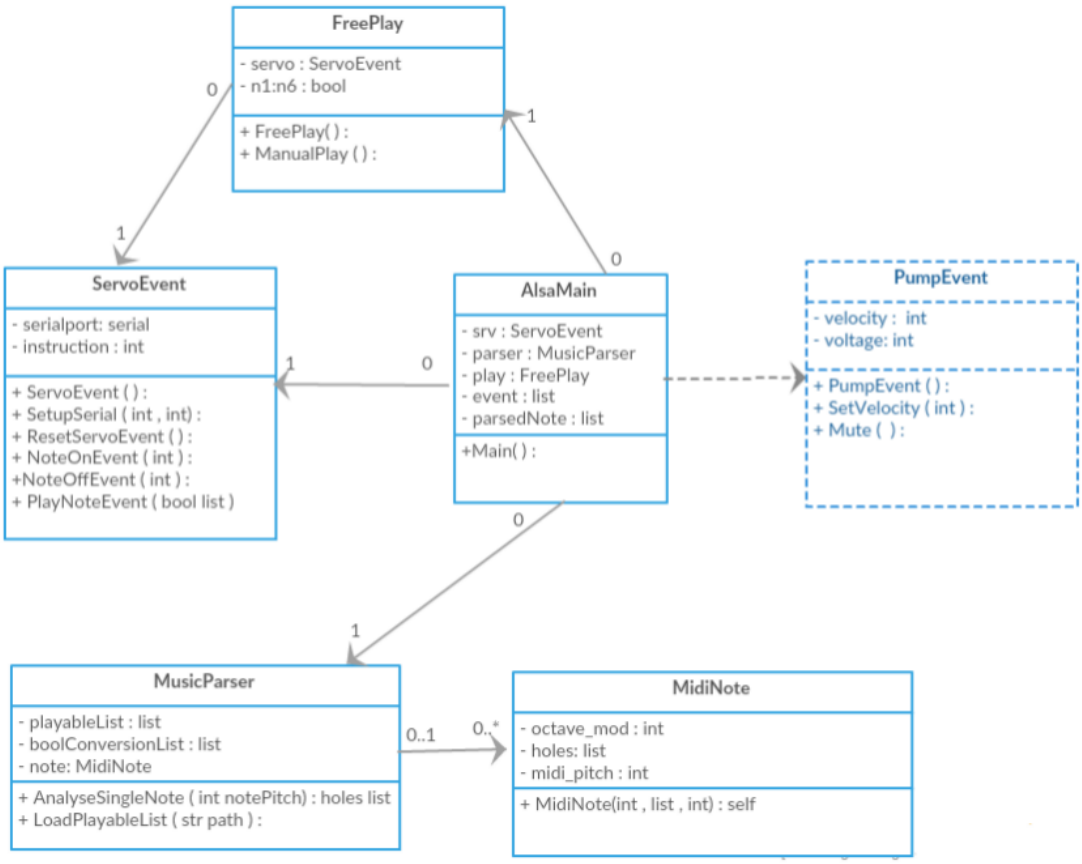

Figure 5 - Class Diagram

2.6 Hardware Design Specifications

As explained previously; while the hardware aspect of the project was not its primary interest, it did form an inseparable element without which the software is rendered useless. To add background and context to the nature of produced software, details of the hardware setup are given below. The majority of work on the physical build has been carried out by IMPACS technicians at Aberystwyth University, but a lot of work was put into designing, specifying and calibrating the device by the author.

The device was built around a WALTONS 1521 Brass Irish whistle in D [32] and for most parts the physical components were designed to work with that particular brand and model. In turn, the software produced for the project was affected by this decision as the list of notes playable by the whistle is hard coded into the program. However, due to the programs modular nature, this could be easily swapped out with a new configuration to accept a different set of notes to allow the instrument to play in a different key (e.g. G or C as is also common).

Interestingly, the initial approach to simulate human behaviour via robotic methods was somewhat misguided. Instead new measures had to be implemented to account for natural human techniques like basic breath control. The biggest difference was the addition of an extra 'finger' to cover up the reed to stop any sound coming out. A human musician would instinctively stop blowing air into the reed but this was impossible without using a computer controlled valve. Instead, to simplify the design the decision was made to cover up the reed and thus allowing for a much-needed staccato effect rather than a continuous drone. In practice, this meant implementing additional code to control the movement of the mute finger; but thanks to implementing a stable API to move the servos, this was as easy as telling the program to move another servo at the right times.

The hand-drawn sketch below shows a very early (sprint 1) prototype of what an expected hardware configuration could look like:

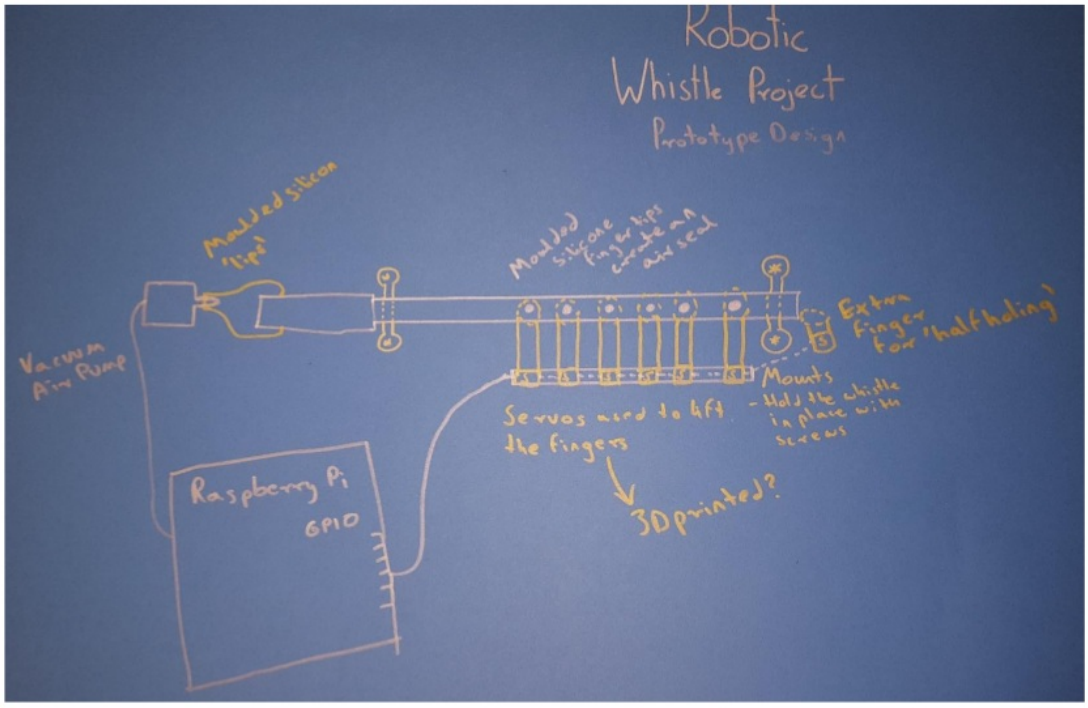

Figure 6 - Sketch schematic of proposed physical device

Ultimately, the final design (and its subsequent implementation) has evolved over the course of the project. Most notably focus was shifted to focusing on the six basic fingers and less priority was given to air control and the extra 7th finger for half holing. See section 3.5 for further analysis.

3 Implementation

This section presents the progress of the implementation stage; explaining the early implementation work and how it eventually prompted a change of direction and a departure from the original design.

3.1 Sandboxing and Early Implementations

As planned in the opening weeks of the project, the program would be split into two releases; one finishing at the Mid Project Demo, and the other at the end of the project in early May. The objectives of the first release were to focus on a) hardware control and b) note translation with the aim to demonstrate fingers moving at the first demo.

3.1.1 Early Attempts at Note Translation

Following initial research and design (see section 2.1) the program was built around the Python MIDI library. A MIDI file was uploaded from the command line upon program execution and parsed in advance before the music was played on the whistle. This approach was functional but in hindsight short sighted due to the emerging difficulties in implementing timing the events correctly which were not apparent until early on into Release 2.

The MIDI file was read in and by using the MIDI Library it was modelled as a MIDI object in OOP; this made it convenient to loop over and access MIDI events one by one for translation. In turn, translated events were added to a playlist for parsing by the MusicParser class which would extract the translated pitch values and associate them with a finger combination to be executed on the whistle. This design worked and it was demoed at the Mid Project Demo with considerable success, but it created drastic problems later on.

3.1.2 Early Attempts at Timing

In spirit of agile development; only limited planning was done in advance at the start of the project. Instead, high-level release planning accounted for timing to be factored in during the second half of the project. Initial research was carried out to determine possible implementation paths; revealing the built-in MIDI ticks as a likely candidate; however, no work was attempted on timing until the first sprint of Release 2.

Further research was carried out and sandboxing began; proving to be a greater challenge than expected. Problems arose due to the nature of MIDI ticks which were expressed as delta times meaning that individual events carried a time in milliseconds from the preceding event. In practice this was useful for stringing events together but proved difficult to calculate the timing of an event for any further parsing. About a week was spent trying to manually implement timing when new literature was discovered; highlighting the use of ALSA to streamline timing processing [33]. Upon further research and sandboxing, this seemed like a viable option however it meant that significant elements of code from Release 1 would have to be removed, rewritten, or otherwise adapted.

3.1.3 Hardware Control

The hardware control component of the program was designed to pass commands to the USB servo controller and move the relevant fingers. Its function is to focus purely on the physical

movement of fingers in the right order without worrying about which note is being played – that functionality would be implemented by other modules.

The hardware control module was implemented in the ServoEvent class, providing the following functions:

- SetupSerial – Used to set the speed of an individual servo. Once the speed is set it will be remembered by the servo controller board which means that this function doesn't get used very much. It is however still useful for testing and maintenance purposes as it bypasses the need for manually encoding binary messages.
- ResetServoEvent - Resets all the servos by moving them to the open position (hard coded 75 degrees)
- NoteOnEvent – Moves a single servo finger down (60 degrees), takes the servo number in as input with values starting from 2
- NoteOffEvent – Moves a single servo finger up (75 degrees), takes the servo number in as input with values starting from 2
- PlayNoteEvent – The most important component of the hardware controller; responsible for parsing a single MIDI event; it receives a list of holes (represented as Booleans; with 'true' meaning the hole is covered' or 'on' and 'false' meaning the hole is uncovered or 'off') and plays them one by one.

3.1.3.1 Hardware Limitations

One of the glaring limitations of the servo board is the inability to send messages in parallel, meaning that instructions must be parsed and sent down the serial line one at a time. In theory, this could become problematic with high speed songs, meaning that the instrument couldn't keep up in changing the finger positions fast enough. The PlayNoteEvent class had to be engineered accordingly to send only one message a time, so in reality there is a tiny gap between each of the fingers moving.

3.2 Major Refactoring of the Program

Following on from the decisions taken in section 3.1.2, most elements of the program were to be rewritten to centre around using ALSA. The new implementation was to be neater (with fewer lines of code where possible), easily extensible and well documented. The initial Release 1 code was functional but up until this point no time was taken to refactor the code; leaving a mess of unused imports, functions and commented out test sections. The major refactoring was an opportunity to take the elements of Release 1 which worked well (like the ServoEvent class from section 3.1.2) and discard the now redundant components like the Python MIDI library.

Not all elements however were discarded. Most notably the entire hardware module (from section 3.1.3) was reused as it was essentially an API to interface with the hardware, and the elements which needed changing were in a separate element of the program. Thanks to the modular nature of the program (and proper design prior to development) it was easy to switch elements in and out. In practice this meant that although the decision to refactor on such a large scale was not one taken lightly, it did prove to be easier than anticipated.

3.3 Development Sequence

Following the decision to refactor the program a sequence of priorities was derived to rank the programming tasks in order of importance; considering any requisite work which needed to be in place.

1. ALSA Setup – The sequencer would be prepared to run on the machine from which the program will be running
2. TiMidity Setup – A backup output would be prepared for any unplayable notes
3. Hardware Control – A set of high level functions to move the servos
4. Serial Connection – Establish a serial connection with the servo board
5. MIDI Through – A MIDI through channel to be set up using alsaseq; this will allow to intercept MIDI events and forward any on to TiMidity
6. Event Handling – Accept a MIDI event from alsaseq, parse it to extract the pitch value
7. Pitch Translation – Compare the extracted pitch with notes playable on the whistle
8. Note Execution – If the note is playable then extract the hole array and forward it on for playing, if not then forward the note to TiMidity via alsaseq
9. Other Extra Features – If given time to spare, additional features will be implemented. Of particular interest would be the ability to integrate the robotic whistle into Mr Price's robotic orchestra.

3.4 Final Implementation

3.4.1 Servo Message Encoding

The ServoEvent class presents an API used to interact with the servos. This is done by sending a chain of binary numbers as seen below. The information on the pattern was derived from page 7 of the official Pololu user manual for the servo board. [31]

Table 1- Servo controller message encoding

start byte = 0x80	device ID = 0x01	command	servo num	data 1	data 2

An example of the above implemented during this project can be seen in the SetupSerial class:

```
  def SetupSerial(self, servo, speed):

        self.serialspeed = chr(0x80) + chr(0x01) + chr(0x01) +
chr(servo) + chr(speed)

        serialport.write(self.serialspeed)
```

The message itself is composed in the 'serialspeed' attribute. In this case the command is pre-set to 1 meaning 'set speed'. In all the other functions of the ServoEvent class the command is set to 4 meaning "Set Position, Absolute".

The speed value (ranging from 1 to 127) and the servo number (ranging from 0 to 15) are then provided as needed to the function.

All the above supplied values are combined into one attribute and sent to the serial port over serialport.write

3.4.2 Note Translation

The note translation algorithm is one of the major components of the program; spreading between the AlsaMain and MusicParser classes. The core structure of the algorithm is explained in this section.

The purpose of the algorithm is to produce a list of six Booleans corresponding to holes which need to be either open or closed on the whistle. Execution begins when a new event is drawn with `event = alsaseq.input()`. The pitch value is captured within ALSA events in list item [7][1].

Having found the numeric MIDI pitch value (ranging from 0-127), a modulo 12 operation is performed on the value. Twelve is a deliberate number here; stemming from the number of notes in a given octave; thus, the remainder of pitch % 12 will always yield the correct note value. E.g. 60 % 12 yields 0 and thus we know that the note to be played will be a C.

The output of the mod function is then fed in as input into the MusicParser. The AnalyseSingleNote function takes the newly calculated pitch value and compares it against a list of playable notes (supplied via the LoadPlayableList function). A strange (and as of the final release unfixable) bug exists in the program where the list is somehow altered during execution; the list of holes would somehow get deleted. Some significant time was invested into debugging this problem with no success. Instead; a workaround has been implemented where the list is re-uploaded for every Note Event. In theory, this is not ideal as it produces unnecessary stress on the CPU and requires more processing power; however in reality this hasn't proved to affect the execution of the program in any meaningful way.

3.4.1 FreePlay Mode

The FreePlay mode was a bonus feature aimed at helping to test the hardware control module and to use during the final demo. The bulk of the implementation was carried out using the curses terminal control library. The notable feature of interest from curses was the ability to capture keyboard input in real time using the *std.scr.getch* function.

The final implementation was somewhat different than anticipated due to two major problems:

- Most standard computer keyboards can only support multi-touch for up to 4 keys – This is problematic when trying to simulate an instrument with 6 different keys (and a potential 7th for the mute). In practice this means that up to 4 fingers can move at any given time but moving all 6 is physically made impossible. The closest alternative is to quickly move the fingers in succession.
- Capturing 'KeyUp' events appears to be impossible from a terminal – The original idea was to have the finger move down on a keyboard press and stay there for as long as the key is down. Once the key is released (thus triggering a KeyUp event) the finger would move up too. However, this appears to be impossible to do from a terminal without

writing substantial amounts of low level Linux code (thus defying the point of this feature).

The final implementation instead uses Boolean flags to keep track of if the key is up or down and flipping it when the relevant key is pressed.

3.5 Hardware Implementation

The act of constructing and implementing the hardware falls outside of the scope covered by this project. It does however present an important factor which steered the design, implementation and testing of the associated software. In this section, the physical construction is outlined, noting its impact on other elements of this project.

3.5.1 Hardware Compromises and Omissions

As outlined previously in section 2.6 , some components of the high-level hardware design had to be carefully selected for removal in order to manage the allocated time. The decision was made to focus on the pitch and timing aspects of MIDI Event Translation and to abandon sound volume/velocity control via controlling the air volume/pressure (see section 2.2.5).

This prompted the construction process to include a simple air mattress pump with the ability to switch on and off. It would take several seconds to power up/down and so it was not viable to rapidly trigger it to play individual notes. Likewise, it would supply a continuous stream of air without any pauses which was rather problematic when attempting to play a number of notes with the same pitch; as a result the sound coming out would appear to be one very long note.

In order to mitigate this, a quick fix was implemented to add a 7th mute finger servo placed over the reed. This is very atypical in simulating a human player of a tin whistle, but it proved to be a necessary decision to distinguish between different notes. It can be seen implemented in the PlayNoteEvent() function of the ServoEvent class.

3.5.2 Photographs of the Physical Device

Figure 7 - A photograph of a very early hardware prototype featuring only a single servo and powered by batteries

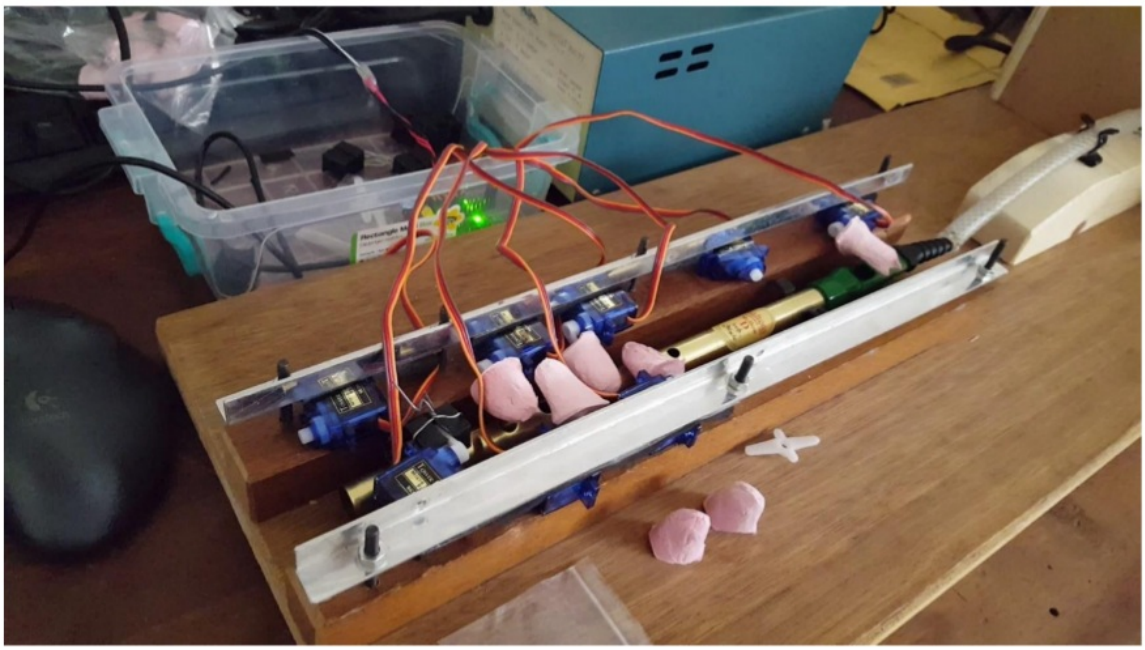

Figure 8- A photograph showing a near completed version of the hardware. Note the pump encased in a wooden mute-box connected via a plastic pipe. The servos are connected to the servo board, visible in the plastic box.

4 Testing

This section provides discussion of the considered, and chosen testing strategies, along with how they were implemented to suit the needs of this project. More detailed test results are seen in Appendix D

4.1 Testing Strategy

In the early stages of research and design, it was noted that testing would always prove to be an interesting challenge for this project, due to its inherently 'human nature'. On the surface, the program looks relatively basic, with limited front end functionality, thus giving us little to test from an end user's perspective. However, the major focus of the project was its backend; the logic behind input handling, note translation, and servo logic, combined together to form the bulk of the assessable work. Therefore, it would make the most sense to employ a testing strategy centred on testing the internal framework of the program, thus making white box testing the ideal option.

Having answered the question of 'how will the program be tested?', the issue of 'what will be tested?' still remained a problem. A lot of apprehension stemmed from the very human nature of music which will either sound 'right' (i.e. in tune, timed correctly, correct volume) or 'wrong' to a human listener. Instilling this behaviour in a computer for testing purposes could be made into a separate project itself; thus, making the limited time allocated for this project even more invaluable. Having analysed the different possibilities, the obvious option was to benchmark the degree of success of the project against its initial objective.

The approach to testing on this project was somewhat non-committal due to the decision to rewrite major elements of the program (see section 3.2). As the project progressed, the unfamiliar (and somewhat experimental) nature of the used technologies became more apparent. More priority was given to understanding the new platforms (Python, ALSA and MIDI) and implementing them, over adapting a test-driven development approach which would take away too much time from development, on writing tests which would possibly have to be discarded anyway. Instead, testing was carried out incrementally in two stages with manual testing done 'on the fly' during development, and more through testing done in the closing weeks of the project after the software has matured.

4.2 Exploratory Testing

The change in direction of the design (as mentioned in section 4.1) could be seen as a significant technical risk. Refactoring major elements of the program relatively late into the development cycle can easily introduce new bugs and break any existing functionality. In order to combat this, some form of testing approach was required to pick out hidden bugs which could hinder the development process in the long term.

However, conventional (i.e. methodical) testing is time consuming and doesn't lend itself well to evolving designs as was the case on this project. Instead, a flexible approach would need to be taken; offering the ability to test quickly, efficiently and with little overheads. Having researched different testing methodologies, exploratory testing emerged as a promising candidate. [34]

Exploratory testing puts a lot of emphasis on the freedom of discovery and investigation by combining the process of writing and performing tests at the same time. This style of testing

leads to exploring the code, finding interesting elements and following into them to explore some more. Naturally, it can be mistaken for simply using the program and waiting for things to break, but in reality exploratory testing can (and should) be made equally rigorous and reliable as scripted testing. This is done by following a formal process; adapted for the needs of the project as seen below:

4.2.1 Creating a Bug Classification

Recurring bugs from the project (and potentially from similar projects as discovered during study of previous implementations and similar works) are categorised in a list. They are analysed to find their root cause and to work out their impact on other components. Some common bugs identified for this type of project include:

Table 2 - Sample bug selection

Bug	Cause	Impact
Wrong octave recognised on note translation	Using the note value alone and not the MIDI pitch value	Notes played are either too high or too low, sounding incorrect
Note timing incorrect	Many possible reasons; the most common was multiple notes trying to play at the same time	Notes try to play too quickly, not sounding right
Messages sent to the servo board in serial	The servo board only supports messages in parallel	Messages are discarded and not played

Having identified some common bugs these were used as the starting point in forming the test charter

4.2.2 Test Charter

The test charter forms the input and output of the test process by guiding the test session in a set direction; it forms an integral part of the exploratory testing process. Prepared in advance, the charter lays out a simple framework for the testing session; giving it an initial starting place. As the session progresses, actions taken are recorder along with their predicted outcomes and the actual outcomes. For every action a short observation is made, indicating its overall success.

Finally, at the end of the session, brief notes are made summarising the result of the performed test. Bugs are outlined and suggested remedial actions are put forward, along with suggestions for further testing.

A separate charter (of some sort; sometimes they were hastily handwritten) was produced for every test session; a typed example can be found in appendix D.

4.2.3 Test Review

Following on from the test session, the charter is reviewed with particular attention given to the debriefing. Test coverage is analysed alongside the results from the debriefing. Future tests are planned and possibly new test charters are written at this stage.

4.3 Analysing the Usefulness of Exploratory Testing

The greatest advantage of using exploratory testing on this project was its ability to produce rapid feedback in a controlled and predictable manner. The emphasis of this was to assist the development process rather than slow it down by writing dedicated scripts. In practice this also meant that every testing session was unique because even the testing methods changed dynamically and evolved to suit the needs of the project at that given time. In total five such testing sessions were held (typically on a weekly basis) and for most parts they helped to steer the balance of 'no tests at all' and 'not devoting too much time to testing'.

However, in the long term exploratory testing falls short on several fronts. The most concerning factor is the lack of stability and predictability. Traditional methodical tests have the ability to produce consistent results every time and if implemented properly can guide the design of the program itself. This becomes particularly problematic on one man projects like this one where only one perspective and point of view is applied to the testing method, regardless of how varied you attempt to make it. As the project matured and fewer changes were made to the code this became more apparent; it got to the stage where a more stable approach to testing was needed.

In summary; the benefits of exploratory testing are not to be underestimated as the value they added to this project was advantageous. However, they are best used as a supplement and not a replacement of traditional testing methods.

4.4 Acceptance Testing and Unit Testing

As was discovered in section 4.3; exploratory testing was a great aid in development by providing quick feedback to the most vulnerable areas of code. However, as the final release drew to the end of its development cycle a more reliable method of testing was needed. Unit testing was studied as a likely candidate however in practice it proved difficult to use without changing existing code. The problem stemmed from the fact that much of what could be tested didn't provide any real use case value. For example, testing the servo control functions would require the function to return something to be tested which would mean adding a return function which isn't required for the implementation itself. A brief attempt was made at unit testing with 3 tests produced testing the functionality of the serial connection.

Instead, emphasis was placed on comprehensive and methodical manual testing of the system. For most parts the testing procedure would take on a process similar to acceptance testing but in reality, it was somewhat more rigorous. The testing process followed the IEEE 829 Test Plan Outline Standard [14] meaning that due care was taken to ensure a clear, through coverage of all requirements. One master testing session was carried out right at the end of the project following pre-defined procedures and ensuring constant environment variables. The test plan and all of its attached deliverables are found in Appendix E.

4.4.1 Test Results

The overall test results yielded a 73% coverage (including tests planned but not carried out) which forms a reasonably reliable foundation to take the program forward for future development and publishing. The main outcome of the testing session was confirming some of the outlying problems with the implementation; namely the unreliability of the fingers covering the holes however this itself forms a hardware issue; the software works as intended.

In conclusion, the test results are satisfactory because they have been carried out in a thought out and logical manner. Although not the same as scripted tests, this testing session proved able to match the rigour of unit testing.

A test result table with comments is available in Appendix F.

5 Critical Evaluation

As is the case with any software development project, some elements have been more successful than others. Although driven at a steady pace, the progress of this project was hindered by the need to learn and experiment with new technologies not used during lectures thus reducing the overall amount of work produced in comparison to what was hoped would be produced at the start. However, despite this the project has succeeded at achieving most its assessable goals and it forms a solid proof of concept piece, possibly able to integrate with a bigger robotic orchestra.

This section provides detailed evaluation of all aspects of the project; assessing the approaches taken and the value of making those decisions weeks/months down the line.

5.1 Project Stages

Although this report is broken into distinctive sections (research/requirements, design, implementation, and testing) these were intermixed in order to facilitate an agile methodology of software development. In this section, the relative successes of each stage of the overall process will be reviewed to examine how successfully they were implemented, and what could have been done differently next time.

5.1.1 Requirements and Research

The opening weeks of the project were dedicated to reading around the chosen subject area and producing a set of objectives to work towards (as seen in section 1.4) For most part this stage seems to have been a success. Significant time was put into studying relevant literature; particularly on previous implementations of robotic instruments. One of the crucial pieces of information to have emerged from this phase was the scale and complexity of similar projects. Music is complex and there are hundreds of years of theory to it. Some of the studied implementations had whole teams of experienced engineers working on them incrementally for years at a time; something which was far outside the scope of this project.

This mean that the requirements for this project had to be manageable and realistic; as only a few weeks could be committed to development with the rest of the time to be divided between the other (equally important) stages of the project.

The big initial worry was that overall the project will seem a bit simplistic in comparison to some of the others, and this is a criticism I would willingly accept. However, this was a conscious decision made to deliver a minimum viable product. As discovered during research; entire papers have been written solely on what is just one element of this project (e.g. properly controlling the airflow into a woodwind instrument) thus making it impossible to produce a fully functional robotic instrument. Instead the efforts could be (and were) focused on developing a solid proof of concept; an early implementation of a solid design which given enough time and resources could be developed into a full instrument.

5.1.2 Design

Thanks to the detailed research and realistic goal setting, the design stage was rather effective and successful. A deliberate decision was made to follow an agile approach and avoid doing too much upfront planning at the start of the project; which in the end turned out to be a good decision. Near enough all of the technologies used on this project were new to me meaning that a significant amount of time was invested into researching, learning and prototyping thus making it difficult to commit to one particular design.

The ability to evolve and street the design of the project as it progressed proved to be valuable in terms of the quality of code and technical work produced, but it arguably had a negative impact on the quantity and the number of objectives that were otherwise implemented. If the project had followed a more linear development model like say the Waterfall where all the planning is done at the start, then perhaps the project would generate more tangible work, but I would argue that its quality and relevance to the set objectives would be lessened significantly. Instead, evolving the design brought the project back on track, even if it meant making the rather daring decision to rewrite major sections of the program.

Ultimately, opinions on design will always vary and there can always be debate over the correctness of any given design. The design of this project was built on the principles of simplicity and easy extendibility which it has succeeded to achieve. The code has been significantly refactored to strip away unnecessary filler, up to the point where it looks like it's lacking. With this in mind the design attempted to be *Pythonic*; that is clear, concise and easy to understand which in my opinion it has achieved.

5.1.3 Implementation of Software and Associated Hardware

The implementation of the program can be considered a success to some extent; though undeniably not all the original requirements have been implemented. For most part the implementation was focused on developing a proof of concept; a minimum viable product that could be the subject of further research; perhaps a next year Major Project could build upon the work by factoring in airflow control and MIDI velocity; a task in and of itself complex enough to form a solid foundation of a separate project.

The biggest recurring problem of the entire project was the disparity between the hardware and the software. As part of my degree in Computer Science, the assessable part of the program was the software component; making it the focus of the project. However, the intended purpose of the produced software was specifically to run on a purpose-built hardware setup without which it could be argued that the software becomes redundant.

The main problem stemmed from the lack of resources available to commit to the physical build. As the sole owner of the project I took responsibility for overseeing the design and some of the construction; but nearly all of the physical construction was done by (or in close collaboration with) technical staff at the Computer Science Department/IMPACS. This decision was taken primarily because the physical construction is not an assessable element of this project and so the valuable time available was allocated to working on the software component of the project.

Over the project's duration, some work was devoted every week to upgrading and calibrating the hardware but it was not enough to achieve the final goal of interfacing the hardware and software together.

To conclude, this brings about the question of what the projects total worth was considering failing to produce music on the whistle itself. In some regard, the project could be seen as a failure because of this, but this would be an oversimplification of the work produced which lays out a very solid foundation for any future work. The software itself fulfils itself fulfils the majority of its requirements as can be seen below:

Table 3 - Breakdown of objectives compared against the final implementation

Objectives	Progress	Objective Met:
1. The program will accept input in the form of a MIDI file	Implemented by sending MIDI files to the program by opening an ALSA through channel	Yes
2. The program will parse the MIDI file to recover the MIDI pitch and timing values	Done by the Python program. Timing values are handled by ALSA in real time	Yes
3. Backend code will control the servo board, providing functions to move servos	API provided in the Python code ServoEvent class; provides full control over the servos from a high-level interface	Yes
4. The program will determine the ability of the tin whistle to play input pitch	Done by the Python program in the MusicParser class	Yes
5. Unplayable notes will be forwarded to an external MIDI device (synthesiser)	Done; alsaseq used in the Python program to establish a MIDI through channel letting any unplayable notes to be forwarded to a synthesiser (e.g. TiMidity)	Yes
6. Playable notes will be parsed for playing on the tin whistle by moving the servos	Done. Notes are parsed as they come in from ALSA one at a time; combining objectives 4 and 5. If a note is playable the program works out the holes which need to be covered on the whistle and passes it on for execution	Yes
7. An extra servo will mute the reed of the whistle to adapt it for taking in a continuous flow of air	Done. Servo 15 controls a finger muting the reed	Yes
8. Open a serial connection to communicate with the servo board	Done; a serial connection with the board is established with the Python serial library	Yes
9. Set up a MIDI throughput channel using ALSA to accept incoming traffic, and to forward any outgoing traffic to a synthesiser	Done using alsaseq within the Python program	Yes
10. Provide a way of switching modes (autoplay and free play)	Done by providing a command line argument to the program at runtime	Yes
11. Clear documentation of the code	Yes; the program was refactored and rewritten from ground up with proper commenting used throughout	Yes
12. Extensible architecture (to fit in with a larger robotic orchestra)	The design of the program is modular, and uses a data flow model via ALSA meaning that it can easily be interfaced to other programs/instruments using ALSA	Yes

In summary, the overall implementation is somewhat problematic in terms of producing a working product. However, this does not take away from the value contributed by the production of good quality software. Ideally; further time would be allocated, to focus purely on the integration aspect of the project and ensuring a seamless transition from the software to the hardware, but as things stand now the work produced presents itself as a strong proof of concept.

5.2 Testing

The testing strategy and execution for the project was overall satisfactory. The decision to use exploratory testing was a calculated risk, but one which has paid off in the end. It fit well with the flexible nature of the project and its relatively small scale. The feedback was instant and the act of carrying out exploratory tests was surprisingly well suited to a one-man team because you don't need to share information with the rest of your development team. On larger teams of multiple developers looking at one codebase it would make sense to have a more advanced continuous integration tool; but given the short duration of this project that felt unnecessary. Instead, I found that exploratory testing forces you to look at your code in a way which you haven't thought of before. This opens new outlooks and viewpoints which otherwise could easily be clouded by over-exposure of a single developer to an entire project for its entire duration.

However, as it was discussed in section 4.3, exploratory testing was only a short-term solution to aid the development process. More methodical testing was needed but as discussed previously, implementing unit tests would require changing some of the source code which itself was a potentially bug-inducing danger. The decision to use manual testing was satisfactory, but it was mostly aided with the use of the IEEE 829 standard. It is all too easy to fall into the trap of playing with the program and proclaiming, 'it worked fine when I used it'. By working towards a regarded industry standard, the testing process was logical, repeatable and accountable. With the use of a tangible written process and a strictly controlled environment the testing was of substantial quality and thus an overall success.

An argument could be made that manual testing will never replace scripted testing and to some extent I will agree with that sentiment. When designing the testing plan for this project, emphasis was placed on achievable goals. Exploratory testing was used throughout the duration and it worked well, but by the time I had planned to implement unit testing the project was on a feature freeze. I was hesitant of making changes to the code to satisfy tests; and so I think the decision to leave unit testing alone and instead embrace methodical testing was a correct one. If the project was to be carried on into future releases it would be logical to refactor elements of the code to make writing unit tests more convenient, however for time being the manual and exploratory tests carried out appear sufficient.

5.3 Methodology

The methodology chosen for this project (Scrumban) was for most part a useful aid in guiding the direction in which the project was heading. The agile nature of it was the defining principle to which the overall success of the project can be attributed by providing it with the much-needed flexibility and the ability to embrace change as an opportunity to produce good quality code rather than 'hack' together a roughly working but buggy. At the same time emphasis was placed on usable code over documentation which is what the intent of the project was; the code

was meant to be releasable at any stage during the project and so particular attention was given to ensure the quality of code was kept up throughout.

However, despite the value of certain elements being very apparent; some elements of Scrumban proved to be a chore more of a cumbrance. In particular, the idea holding any form of meeting with one person proved more difficult than originally intended because every day (for daily 'standups') and every week (for retrospectives) I found myself going into the 'meeting' with the same mindset. The traditional purpose of these meetings in Scrum/Agile is to communicate with your team, your product owner and your colleagues. On a one-man project the purpose of this becomes somewhat redundant as instead of preparing a short bite sized chunk of the really relevant information you carry all of the knowledge of the entire project in your head. In the opening weeks, an attempt was made to try and maintain a fresh and focused outlook on these meetings but they were quite quickly dismissed as a waste of time and instead replaced with a daily look at the Kanban board.

Furthermore, some attempt was made into adapting a Kaizen approach of reducing waste and improving the development process. Over the duration of the project this had a minor impact; notably on the quality of code and comments but it led to the realisation that Kaizen was more suited for ongoing development rather than a short project like this and thus I was unable to make any useful changes to the development process in the time given.

In conclusion; the choice to use an agile development methodology was certainly the correct one because it was able to support the much-needed learning and experimenting process. It was also much more forgiving of mistakes associated with using brand new technology and overall it let the program to adapt to its intended needs. Not all of the tools initially selected proved to be as useful as planned, but in spirit of Agile the process was reviewed (be it very informally) and the elements which didn't contribute real value were discarded, whereas those that helped persisted.

Good discipline was generally maintained throughout the project but it did dip at times, lessening the overall effectiveness. This is a natural risk in any agile project and in the real world it would be addressed by employing a scrum master or a coach; naturally this was not possible on a one-man project and although some attempts were made at Kaizen (and self-improvement) they were largely a waste of time.

5.4 Tools and Development Environment

The time invested into investigating various development tools was significant; which when examined retrospectively was a good decision. The aim with all the tools was for them to be simple to use and where possible to run in the cloud (thus making the platform easily movable regardless of where the program is being developed from). The time taken examining the needs of the project was well spent because the very physical nature of interacting over USB with the servo board became apparent; meaning that certain initially specified platforms (e.g. Cloud9) were dismissed.

Overall, all the chosen tools succeeded in fulfilling their respective roles. PyCharm made developing in Python easier by providing many quality of life features. GitHub allowed for easy backup and version control. Trello particularly stood out, by providing a very easy but functional issue tracking board which was used steadily throughout the project.

As the chosen language, Python proved to be more than suitable through it's very simple syntax but powerful and extendible libraries. I found that Python lent itself really well to just about every challenge this project was able to offer, and I was particularly surprised how far it can be

extended through the different libraries tried during the implementation (Python MIDI, and alsaseq most notably).

As a base platform; Linux was the ideal choice for this program. The ability to interact with physical devices over serial was convenient and easy; and the built in ALSA sequencer formed the backbone of the program upon which my work could be built. Porting the program to a different operating system in the future would be possible but it would require a major rework to the core design which could be seen as a drawback. The program is not very platform 'agnostic' but at the same time it does run off a free and open source operating system meaning that it is easily workable for anyone willing to take up derived work.

In conclusion, little can be said to criticise the chosen programs. Overall, they have been thoroughly researched to aid in achieving specific goals and they have been seen successful in doing so.

5.4 Project Reflections

Above all, this project has been a journey of learning new technologies and putting years of education to practical use. Mistakes were made along the way, but at the end of the day one cannot help but to feel a certain sense of pride in one's work. Overall I would say the project for me has been about learning how to use new technologies in a methodological manner. Up until this point in my university and professional careers you could get away with 'hacking' together quick solutions but my experience with this project took me far beyond that into a new level of professionalism in preparation for a career in the computing industry.

The main factor which somewhat 'hindered' the project was perhaps putting as much emphasis on the research and design stages of the project as I had spent on the development. It had become clear that my programming skills were somewhat lacking to take on a project of this scale meaning that a lot of time had to be spent familiarising myself with Python; a language which I've had an interest in for a while but not one that I had the chance to use properly (and certainly not on a project of this scale). However, if given the chance to start again I think I would still use Python because even though the number of extra features added was very small, the code that was produced was of good quality; attempting to follow good Pythonic principles and avoiding any bloat [35].

The time management on the project was overall satisfactory but as mentioned previously could be seen as a weakness due to not being able to implement more varied features into the program. A large part of this problem stems from a lack of practical experience of participating in such software projects and because of this I felt like it was very difficult to judge my relative progress during any given timeframe of the development period. However, I feel like there's no lessons learned like those learned from your own mistakes and experiences and so I believe that if I was to attempt the project again I would have a significantly better ability to gauge my progress and take proactive action well in advance.

Appendices

Appendix A – Third Party Code and Libraries

This section lists and outlines the different libraries and other software used on this project.

ALSA – The Advanced Linux Sound Architecture framework was used for its sequencer. It is available to download from https://www.alsa-project.org/main/index.php/Download under either the GPL or the LGPL licenses

Alsaseq – The Python library allowing for interfacing with ALSA by opening an ALSA channel. Traffic can be intercepted inside the channel for parsing. Along with ALSA this was of crucial importance to the project. The code was used but not modified. It is available from the official Python repository: https://pypi.python.org/pypi/alsaseq/0.4.1 under the GPLv2 or later

Python MIDI Library - Used in code from release 1; as demonstrated during the Mid Project Demo. Removed from the program for the final release. Available from: https://github.com/vishnubob/python-midi under the MIT license.

PySerial Library – Used to establish a serial connection over USB between the host PC and the servo board. Available from the official Python repository: https://pypi.python.org/pypi/pyserial under the BSD license.

Python Curses Library – Used to implement the Free Play mode by capturing keyboard presses from the terminal. Available included in the Python standard library.

Pytest – A testing framework; used to run unit tests for the project. Available from: https://docs.pytest.org/en/latest/ under the MIT License

Appendix B – Ethics Form

AU Status

Undergraduate or PG Taught

Your aber.ac.uk email address

lum22@aber.ac.uk

Full Name

Lukasz Medza

Please enter the name of the person responsible for reviewing your assessment.

Reyer Zwiggelaar

Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment

rrz@aber.ac.uk

Supervisor or Institute Director of Research Department

cs

Module code (Only enter if you have been asked to do so)

CS39440

Proposed Study Title

Robotic Tin Whistle

Proposed Start Date

January 2017

Proposed Completion Date

May 2017

Are you conducting a quantitative or qualitative research project?

Mixed Methods

Does your research require external ethical approval under the Health Research Authority?

No

Does your research involve human participants?

No

Are you completing this form for your own research?

Yes

Does your research involve human participants?

No

Institute

IMPACS

Please provide a brief summary of your project (150 word max)

The construction of a physical devices consisting of a tin whistle, servos , a pump and a raspberry pi to take in midi input files and play the notes on the tin whistle; simulating a human player.

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?

Not applicable

Will appropriate measures be put in place for the secure and confidential storage of data?

No

Does the research pose more than minimal and predictable risk to the researcher?

Not applicable

Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?

No

Please include any further relevant information for this section here:

If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one.

Yes

Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change.

Yes

Please include any further relevant information for this section here:

N/A

Appendix C – User Manual

The following is a brief guide explaining how to run the software for this project; particularly drawing attention to some of the less obvious components.

System Requirements

The program is only designed to run on GNU/Linux operating systems; it will NOT work on Windows or UNIX because of the need to utilise the ALSA sequencer built into the Linux kernel. For further explanation see sections 1.7.1 and 2.1

The program has been successfully tested on Ubuntu and Debian distributions but should run equally well on any OS utilising the Linux kernel with the built in ALSA framework.

Backup MIDI Output

This is an optional feature and the program will execute and run without it, however it is strongly suggested that a backup output is provided to fill in any gaps in the musical range of the instrument. For additional information see section 2.2.3

A backup output can be either physical (like a MIDI keyboard or a synthesiser) or it can be virtual. By default the program is hardcoded to forward any unplayable notes to **channel 129 which should in turn be set as the input channel of your chosen backup device.**

For testing purposes this project utilised Timidity. Timidity is an Open Source software synthesiser; using ALSA and the PC's sound card to play MIDI files in real time [36]. On Ubuntu based systems it can be installed via: `sudo apt-get install timidity`

In order to set up Timidity for use here it should be made to run as a daemon where it will act as an ALSA sequencer client. To do this the `timidity.service` should be enabled. On dedicated machines it is a good idea to make this service run on boot.

Once running, the output can be confirmed using: `aconnect -o`

A correct setup should yield the following result:

```
client 128: 'TiMidity' [type=user]
    0 'TiMidity port 0 '
    1 'TiMidity port 1 '
    2 'TiMidity port 2 '
    3 'TiMidity port 3 '
```

Preparing the Hardware

The Pololu servo controller requires minor calibration before it is made fully operational. By default when plugged into a USB port the device might not work; displaying a green light and a flashing red light meaning that the baudrate of the USB connection is too high. When testing on a Raspberry Pi this was not an issue, but when tested on a Lenovo ideapad laptop this had to be reset on every boot. Your experience might vary depending on your platform.

The baudrate of a USB port can be checked with the following; typically defaulting to 115200:

```
stty -F /dev/ttyUSB0
```

The device itself supports a baudrate in the range of 2,000 to 40,000 with 9600 proving to be sufficient during testing. The baudrate can be easily changed with the following:

```
stty -F /dev/ttyUSB0 9600
```

Note; in some cases the baudrate doesn't update until the next power cycle meaning that the computer has to be put into suspend mode and then re-activated to let the baudrate refresh.

Full technical details of operating the device can be found in the Pololu USB 16-Servo Controller User's Guide [31]

Running the Program

The program runs two separate components which both need to run at the same time. Firstly; the Python code can be executed with:

```
python AlsaMain.py
```

The program will work in the background; but can be checked that it's working by running the following:

```
aconnect -o
```

The expected output should now show the previously set up Timidity on channel 128 and also a new appearance of a channel 129 labelled 'RoboWhistle PassThrough':

```
client 129: 'RoboWhistle PassThrough' [type=user]

    0 ' RoboWhistle PassThrough ' 0
```

Running the above will set up the 'listener' part of the project; ready to take in MIDI input. The input itself is provided via ALSA by running the following command:

```
aplaymidi --port 129 mymidifile.mid
```

Running the above will then pass the MIDI file for execution to the Python program where it will be parsed one event at a time and played in real time.

Appendix D – Sample Test Charter

The following is a sample test charter from the second half of the project following the re-factoring. A basic template for the test charter was derived from the Qualitest website and adapted for this project. [37]

Tester: Lukasz Medza

Timebox: 25/04/2017

Duration of Execution: 30 minutes

What to test: Testing for this session should focus on the note translation system. As the core feature of the program, it should be given plenty of test exposure ideally in different testing styles and methods. This will prove a starting point for finding any potentially deep rooted bugs in the system

How to test: As a starting point, a blackbox testing approach would be advisable. Simulate an end user running the program and aim to achieve the functionality described in the project objectives. A MIDI file should be fed into the program and the translation of note events should be observed with particular attention given to:

- The pitch of the notes
- The octave of notes
- The list of holes passed on to be played

Test Setup: Follow the usual program setup procedure from the user guide (Appendix C). Use a single-track MIDI file for testing

Test Log

Table 4 - Exploratory testing log

Inputs and Action	Expected Output	Actual Output	Observations
Program set up	Orange light on the servo board indicating correctly set baudrate	Orange light	**Pass**
Feed MIDI file mary.mid into the program	Input is taken in, song is parsed and executed	Notes parsed as intended	**Pass**
Edit the program; add a print statement to display note pitch as its being played	Note pitch displayed upon note parsing in AnalyseSingleNote() function	Pitch displayed as intended	**Pass**
Take first ten pitch values, compare them against a lookup table and compare the values against the MIDI sheet music	Pitch values match	Values match where intended, some return as 0	**Pass** – Events other than NoteOn/NoteOff have a pitch value of 0

Take the previously translated ten pitch values and work out if they are playable on the whistle or not. If playable work out the combination of holes to be covered and compare it with the print statements of the values from item.holes() on the playableList	First few values will be 0 and thus unplayable , others will have matching hole values given in Booleans	Values match as expected	**Pass**

Debriefing: The brief testing session was a success; print statements were effectively used to confirm the expected behaviour of the program. Emphasis was made on events other than NoteOff and NoteOn (such as the various ProgramChange events typically appearing at the start and end of every track). Following on from this test session further testing should be carried out to ensure that these events are parsed to yield predictable/expected behaviour.

Appendix E – IEEE 829 Master Test Outline

Test Plan Identifier: Final Master Plan

References: This test plan references the following sections of the project's Final Report document:

- 1.4.1 (Use Case)
- 1.4.2 (Functional Objectives)
- 1.4.3 (Technical Objectives)

Introduction: This document is the master test plan produced for the master (final) testing session of the project. The scope of this project is the entirety of the software implementation; all of the produced software is to be tested. This document lays out the testing boundaries, conditions and requirements.

Test Items: The purpose of this plan is to test all of the original project objectives; these have been broken down into 17 testable programming features:

1. The program will accept input in the form of a MIDI file
2. The file will be passed in from a command line interface
3. The program should handle incorrect/broken files being passed to it
4. The program will parse the MIDI file to recover the MIDI pitch and timing values
5. Backend code will control the servo board, providing functions to move servos
6. Functions should exist to reset all servos, move an individual servo, reset and individual servo and also to set the servo's speed
7. The program will determine the ability of the tin whistle to play input pitch
8. The program will maintain a list of playable notes and compare incoming notes against it
9. Unplayable notes will be forwarded to an external MIDI device (synthesiser)
10. A set channel number will be associated with the output letting the user chose their preferred synthesiser (be it virtual or physical)
11. Playable notes will be parsed for playing on the tin whistle by moving the servos
12. An extra servo will mute the reed of the whistle to adapt it for taking in a continuous flow of air
13. Open a serial connection to communicate with the servo board
14. Set up a MIDI throughput channel using ALSA to accept incoming traffic, and to forward any outgoing traffic to a synthesiser
15. Provide a way of switching modes (autoplay and free play)
16. Clear documentation of the code
17. Extensible architecture (to fit in with a larger robotic orchestra)

Software Risk Issues:

The main risks of the program include:

1. Unplayable notes
2. Simultaneous events occurring at once

3. Unknown events from the sequencer

Features to be tested:

The features of the program to be tested from a USER's point of view have been identified on the Use Case diagram. Their respective priorities are noted next to them.

A. Play MIDI files HIGH
B. Manually move servos to play my own music LOW
C. Interface with other instruments MEDIUM
D. Hear all of the sounds, not just those playable by the whistle HIGH

Features NOT to be tested:

The following features have been omitted from the implementation of Release 2 although they still feature in the ideal design of the program. They are not implemented to a sufficiently high standard to be ready for shipping (and thus testing prior to shipping)

1. Air flow control
2. Track separation

Approach: This section outlines the approach to be followed during the testing stage.

Tool Requirements: Comprehensive testing requires a machine running the software to be connected to the purpose-built device. Some of the hardware testing can be omitted to perform the testing without requiring the specialist equipment. An external synthesiser on port 128 is required; for this purpose TiMidity will be used. A variety of MIDI files will be needed to use as input into the program.

Collection of Metrics: Metrics are not deemed to be particularly useful for this project due to its overall short duration. Therefore to simply the process all requirements will be weighted equally. Total requirement coverage will be calculated with the following formula [38]:

$$Requirements\ Coverage = \left(\frac{Number\ of\ requirements\ covered}{Total\ number\ of\ requirements}\right) * 100$$

Configuration Management: The configuration of the program will be carried out manually before every test by re-running the program before the start of every test. For

seemingly persistent bugs the hardware will be disconnected and re-connected to allow a fresh config to update.

Hardware: The testing will be done on a Lenovo Ideapad Z570 laptop connected to the Pololu USB 16 servo controller board

Software: The following software will be used:

- Linux Kernel 4.8.0-46-generic
- Xubuntu 16.04
- ALSA version k4.8.0-46-generic
- TiMidity++ 2.13.2

Processing of untestable requirements: The majority of requirements are testable with the exception of 16 regarding the quality of comments/documentation. In this case; the comments will be compared against the official Python documentation guide. [39]

Item Pass/Fail Criteria: Individual criteria will be assigned for each requirement. The fail criteria can vary (as implementations can often fail in multiple ways); given below are some of the expected most likely scenarios of failure. Every test item/project requirement is also matched with an end user feature as expressed on the Use Case diagram.

Table 5 - Test Plan laying out pass and fail criteria

Item	Feature Matched	Pass Criteria	Expected Fail Criteria
1	A	A MIDI file is sent to the program and accepted by ALSA. It is available for parsing from within the Python program as individual elements become added to a queue	Files other than MIDI are accepted; probably causing the program to crash. Alternatively, ALSA doesn't accept the MIDI
2	A	MIDI's are passed in from the command line using the `aplaymidi` command	MIDI's cannot be sent to the program
3	A	Non MIDI files should be rejected. Corrupted or broken files are also rejected	Corrupted/broken/non MIDI files are allowed to play; probably crashing the program
4	A	MIDI pitch is recovered from the ALSA event object from list [7][1]	No pitch value is found, or out of bounds value is found (outside of 0-127)
5	A , B	Classes are able to move all servo fingers from an open to closed position	Fingers don't move; wrong fingers move ;

6	A , B	Servos are able to reset to an open position; individually and all at the same time	Servo position doesn't change
7	C , D	Correct notes are parsed for playing, unplayable notes are forwarded to the synthesiser	Device attempts to play notes physically unplayable by the whistle
8	A	List of playable notes read in from the MusicParser class when every individual note is parsed	List is unreadable or becomes changed during execution; removing items from itself
9	D, C	Port 128 will function as the output, unplayable notes will be passed to TiMidity	Unplayable notes are not heard playing
10	C	Any device can be connected to port 128. For the purposes of testing TiMidity will be used	No device can pick up input from port 128
11	A , B	Relevant servos move to cover the holes required for given notes	Incorrect holes covered, no holes covered at all
12	A	Mute finger moves at the correct time, separating notes	Notes are not separated; one continuous drone persists making multiple short notes sound like one long note
13	A , B	A green light is visible on the servo board meaning that a connection has been established. Fingers of the servo board move when given instructions.	Red or yellow lights flash on the servo board; error is thrown from the terminal. Typically serialutil.SerialException
14	A , C , D	Alsaseq will successfully establish a serial through port which will be visible when `aconnect -o` is ran from the terminal	No MIDI through visible from the command line; or otherwise the channel is broken and MIDI events cannot be intercepted or forwarded
15	B	AutoPlay is accessed by default when the program is first ran. FreePlay is accessed when an argument (any) is passed from the command line	Cannot switch modes, the program defaults to one mode or another
16	N/A	Where applicable, documentation follows most good documentation principles laid out in the Python programmers guide	Documentation doesn't meet good practices
17	D	The program/device can interface with other devices also using the ALSA sequencer	Program can't interface with other devices; or can't do so to an acceptable degree of proficiency

Suspension Criteria: Under normal circumstances the testing will carry on despite any potential failures.

Test Deliverables: The following deliverables will be produced by the end of the testing stage:

1. This test plan document

2. Test table detailing steps taken during the testing process and their associated pass/fail metrics
3. Error/bug list with suggested corrective actions
4. Final conclusion including metrics

Environmental Needs: No special needs are required for the testing session. All tests will be carried out on the same machine as was used for development of the program.

Staffing and Training: All testing will be carried out by the developer/designed of the program, thus requiring no additional staffing or training.

Responsibilities: All roles will be carried out by Lukasz Medza

Planning Risks and Contingencies: The main concern with the testing is time management and the over-exposure to a single person. Ideally multiple people should be reviewing the program.

Appendix F – Abridged Test Results

Due to word count limitations of this report; below is a results table from the end testing session. The process has been recorded in appendix E

Table 6 - Results of the master test session

Test Number	Outcome	Notes
1	PASS	MIDI file accepted into the program, events are added to the ALSA queue and available for pulling off from the queue using alsaseq.input()
2	PASS	MIDI passed in from bash shell using aplaymidi
3	PASS	Non-MIDI file rejected at ALSA level with ' file is not a Standard MIDI File error' returned.
4	PASS	Pitch found for all events; all events other than NoteOn/NoteOff default to pitch 0 in which case they are passed to port 128 for parsing and are essentially thrown away/ignored
5	PASS	Fingers seem to move in the right order
6	PASS	All servos are able to be reset including the mute servo finger
7	PASS	Correct pitch is worked out
8	PASS	List read in for every note parse
9	PASS	Unplayable notes were audible in TiMidity and a message was printed in the terminal when an unplayable note was forwarded
10	PASS	Works using TiMidity, further testing advised for other devices
11	FAIL	All servos move but not all holes are covered fully as intended; need to try using new fingers
12	PASS	Mute finger moves correctly at the right time
13	PASS	Green light visible
14	PASS	MidiThrough visible when running aconnect -o
15	PASS	Most good principles are followed
16	PASS	Documentation appears to match good practices
17	FAIL	Unable to test compatibility with other devices

Total: 14 passes, 2 fails

Total number of requirements (including those in the design spec but not tested): 19

Test coverage: 14/19 * 100 = 73%

Annotated Bibliography

[1] BBC, "'Robotic orchestra' performs Doctor Who theme music," 31 Dec 2014. [Online]. Available: http://www.bbc.co.uk/news/uk-wales-30643544. [Accessed 2017 Jan 10].

A news article on Mr Price's robotic orchestra, demonstrating the inspiration behind the project

[2] B. Four, "Royal Institution Christmas Lectures - "The world's greatest robot orchestra perform the Doctor Who theme tune"," BBC, 23 Dec 2014. [Online]. Available: http://www.bbc.co.uk/programmes/p02ftwfw. [Accessed 10 Jan 2017].

A video from Mr Price's robotic orchestra performing the Dr Who theme song

[3] Y. K. T. F. K. K. ,. T. K. Tomonori KATO, "Control of blown air for a Soprano-Recorder-Playing robot using unsteady flow rate measurements and control techniques," in *8th JFPS International Symposium on Fluid Power*, Okinawa, 2011.

A paper focused specifically on the airflow aspect of robotic woodwind instruments. In-depth analysis of the problems associated with airflow rate control and measurement. Presents a viable yet complex solution which proved too complicated to be implemented on this project

[4] A. Kapur, "A History Of Robotic Musical Instruments," in *International Computer Music Conference*, Barcelona, 2005.

A paper on mthe history of robotic instruments, particularly tracing the developments and trends over the past 30 years. Very useful; this paper helped to form the foundation of this project providing leads to many other useful sources

[5] M. Kajitani, "Development of Musician Robotos in Japan," University of Electro-communications, Chofugaoka Chofu Tokyo, Tokyo, 1999.

A general overview of robotic music in Japan. Of particular interest is the section on the Mubot Recorder; somewhat similar in implementation to that of this project

[6] M. A. Fabio, "The Chandelier: An Exploration in Robotic Musical Instrument Design," Massachusetts Institute of Technology, Massachusetts, 2007.

A paper on robotic music design in general but specifically focusing on producing an interactive music display for public to engage with. Very useful at introducing high level concepts in robotic music

[7] MIDI Association, "The Official MIDI Specifications," MIDI Association, [Online]. Available: https://www.midi.org/specifications/. [Accessed 2017 Apr 24].

The official MIDI documentations website. Useful as reference but also as learning material for understanding the backend technology of MIDI. Used frequently throughout the project, particularly to understand the values contained in MIDI events

[8] T. Felix, "Ted's Linux MIDI Guide," Dec 2016. [Online]. Available: http://tedfelix.com/linux/linux-midi.html. [Accessed 2017 Apr 24].

A guide laying out tips and instructions on fine tuning a Linux based PC for working with MIDI. Particularly useful for understanding ALSA and its built-in sequencer

[9] Z. A. Shaw, "Learn Python the Hard Way," [Online]. Available: https://learnpythonthehardway.org/. [Accessed 24 Apr 2017].

A comprehensive guide to learning Python. Used frequently at the start of the project to learn Python and then throughout the project as a reference guide. Very helpful, containing lots of worked examples

[10] Python Software Foundation, "Python Standard Library Documentation," [Online]. Available: https://docs.python.org/2/library/unittest.html. [Accessed 16 March 2017].

Official Python documentation on unit tests. Helpful in transitioning from JUnit to Python unit tests

[11] Atlassian Inc, "Software Testing: A Culture of Quality," [Online]. Available: https://www.atlassian.com/software-testing. [Accessed 07 May 2017].

Article comparing different approaches to testing; useful for building a model which explains which methodology is used to test different software components. Useful starting point

[12] Software Testing Help, "Definitive Guide to Develop a Good Test Strategy Document with These 7 Simple Steps," 17 Apr 2017. [Online]. Available: http://www.softwaretestinghelp.com/writing-test-strategy-document-template/. [Accessed 07 May 2017].

A high level overview of test plans; used as part of the research process to make the testing more methodical and reliable. Provided useful commentary on writing effective test plans

[13] *J. Bach, "Exploratory Testing Explained," 16 Apr 2003. [Online]. Available: http://www.satisfice.com/articles/et-article.pdf. [Accessed 07 May 2017].*

A very detailed analysis of exploratory testing. It presents a very compelling argument for its correct use in an appropriate setting. Highly useful in working out the low level details of the testing plan

[14] Faculty of IT Brno Czech Republic, "Test Plan Outline (IEEE 829 Format)," [Online]. Available: http://www.fit.vutbr.cz/study/courses/ITS/public/ieee829.html. [Accessed 07 May 2017].

A breakdown of IEEE 829 testing. Used as a template for the final project test plan

[15] Eyelean, "Scrum vs. Kanban vs. Scrumban," [Online]. Available: http://www.eylean.com/Publications/DownloadPublication/4e93cecc-4cb2-4e3f-849d-810d7aea33a5?name=Whitepaper---Scrum-vs-Kanban-vs-Scrumban. [Accessed 6 Nov 2016].

A detailed report comparing and contrasting Scrum , Kanban and Scrumban. Provided a lot of useful information used in making the decision as to which development methodology to follow throughout the development process

[16] Trello, "rello keeps track of everything, from the big picture to the minute details.," [Online]. Available: https://trello.com/. [Accessed 24 Apr 2017].

The issue tracking service used for this project. It is free to use (in a basic version, deemed sufficient for this project) and runs in the browser without installation, making it very convenient

[17] Freja Daily, "A Beginners Guide To Kaizen : The Art Of Improving Continously," [Online]. Available: http://frejadaily.com/2014/12/how-to-improve-continously/. [Accessed 20 Apr 2017].

An article outlining good Kaizen principles. Referred to in attempt to streamline the development process and make it leaner

[18] The ALSA Project, "Advanced Linux Sound Architecture Sequencer," 02 April 1999. [Online]. Available: http://www.alsa-project.org/~frank/alsa-sequencer/. [Accessed 24 Apr 2017].

The official website of the ALSA project including the download. Used heavily throughout the project

[19] Cloud9.io IDE, "Cloud9 IDE - Your Code Anywhere, Anytime," [Online]. Available: https://c9.io. [Accessed 28 Oct 2016].

An online development environment, tested early on for its suitability. The look and feel of it was very appealing but it fell short due to its poor suitability for interfacing with physical devices connected to the computer

[20] JetBrains, "PyCharm - Python IDE for Professional Developers," [Online]. Available: https://www.jetbrains.com/pycharm/. [Accessed 24 Apr 2017].

The IDE used for the project under an academic license. Provided many useful tools otherwise not found in the default Python IDE

[21] G. Lionetti, "Git vs Mercurial: Why Git?," Atlassian, 13 March 2012. [Online]. Available: https://www.atlassian.com/blog/git/git-vs-mercurial-why-git. [Accessed 15 Apr 2017].

An article comparing Git with Mercurial; used as part of the research process to decide which VCS platform to use. The article makes a compelling argument towards Git which ended up being used on the project

[22] Atlassian Inc, "Bitbucket - Code, Manage, Collaborate," [Online]. Available: https://bitbucket.org/. [Accessed 24 Apr 2017].

A commercial Git server solution which runs either in the cloud or on a local installation. Website used to research Git solutions

[23] GitHub, "GitHub - The World's Leading Software Platform," [Online]. Available: https://github.com/. [Accessed 24 April 2017].

An online Git repository. A free repository was available under an academic license, making this a useful choice

[24] JetBrains, "Version Control with PyCharm," 23 Dec 2016. [Online]. Available: https://www.jetbrains.com/help/pycharm/2016.3/version-control-with-pycharm.html. [Accessed 23 April 2017].

PyCharm IDE documentation, outlining the built in feature letting the program easily integrate with a Git repository

[25] draw.io, "Draw.io - An online diagraming website," [Online]. Available: https://www.draw.io/. [Accessed 24 Apr 2017].

An online tool used to create diagrams. It was very intuitive to use, making it a pain-free alternative to MS Word's built in diagram tool

[26] Atlassian Inc, "JIRA Software - The #1 software development tool used by agile teams," [Online]. Available: https://www.atlassian.com/software/jira. [Accessed 20 Apr 2017].

An issue tracking tool, running either in the cloud or on a local server. The service was analysed as part of choosing an issue tracking/project management tool. The functionality seemed superior to the alternatives but it cost money thus making it less favourable

[27] vishnubob, "Python MIDI," 08 Jul 2015. [Online]. Available: https://github.com/vishnubob/python-midi. [Accessed 24 Apr 2017].

The Python MIDI library repository. It was used initially during release 1 of the project, but was later discarded in order to use the ALSA sequencer, thus making it deprecated

[28] "alsaseq 0.4.1 - ALSA sequencer bindings for Python," 13 Jan 2014. [Online]. Available: https://pypi.python.org/pypi/alsaseq/0.4.1. [Accessed 24 Apr 2017].

The Python Alsaseq library. Used to interface Python with ALSA's sequencer. It forms an invaluable part of the project, making it a centrepiece of the final design and implementation

[29] C. Dobrian, "Timing in MIDI files," 19 May 2014. [Online]. Available: http://sites.uci.edu/camp2014/2014/05/19/timing-in-midi-files/. [Accessed 24 Apr 2017].

An article explaining how ticks work in MIDI; connecting separate events together. Used as part of the research to come up with a timing solution.

[30] Dublin Institute of Technology, "MIDI Note Numbers for Different Octaves," [Online]. Available: http://www.electronics.dit.ie/staff/tscarff/Music_technology/midi/midi_note_numbers_for_octaves.htm. [Accessed 24 Apr 2017].

A website containing a table of all 127 MIDI pitch values, along with their respective octaves and which note they translate to. Referred to frequently throughout the project.

[31] Pololu, "USB 16-Servo Controller User's Guide (Preliminary)," 2004. [Online]. Available: https://www.pololu.com/file/0J35/usc01a_guide.pdf. [Accessed 24 Apr 2017].

The official document for the servo board used on the project. It was extremely important to the nature of this project, making it referred to on a regular basis.

[32] Amazon , "WALTONS 1521 Brass Irish Whistle," [Online]. Available: https://www.amazon.co.uk/WALTONS-1521-Brass-Irish-Whistle/dp/B000VPFOFK. [Accessed 24 Apr 2017].

The exact model of tin whistle purchased for this project. This is where the whistle was ordered from

[33] C. Timbers, "Raspberry Pi Lights: how to sync Christmas lights to midi audio," 3 December 2013. [Online]. Available: https://chivalrytimberz.wordpress.com/2012/12/03/pi-lights/. [Accessed 01 Apr 2017].

An article highlighting the use of ALSA to time Christmas lights in time with MIDI music. A similar approach was used in Mr Price's robotic orchestra, thus prompting further research into this article and ALSA in general

[34] Guru99, "Exploratory Testing Tutorial: Process, Techniques & Examples," [Online]. Available: http://www.guru99.com/exploratory-testing.html. [Accessed 25 Apr 2017].

A detailed article, explaining the nature of exploratory testing along with its advantages and disadvantages. Used throughout the project to refer back when making test charters

[35] K. Reitz, "Hitchhiker's Guide to Python: Code Style," 2016. [Online]. Available: http://python-guide-pt-br.readthedocs.io/en/latest/writing/style/. [Accessed 07 May 2017].

A comprehensive guide to good coding practices in Python; used to back up the meaning/intended use of the word 'Pythonic' and the corresponding values

[36] mput<mput@users.sf.net>, "TiMidity++," 04 Oct 2004. [Online]. Available: http://timidity.sourceforge.net/. [Accessed 24 Apr 2017].

The website of the TiMidity++ project; a synthesiser used as the forwarding point for unplayable notes.

[37] Qualitest, "How to Write an Exploratory Test Charter," [Online]. Available: https://www.qualitestgroup.com/resources/knowledge-center/how-to-guide/write-exploratory-test-charter/. [Accessed 25 Apr 2017].

A template for test charters which was studied and adapted for this project. Some of the elements were cut down, while others were added to it.

[38] Zephyr, "QA METRICS: THE VALUE OF TESTING METRICS WITHIN SOFTWARE DEVELOPMENT," [Online]. Available: https://www.getzephyr.com/resources/whitepapers/qa-metrics-value-testing-metrics-within-software-development. [Accessed 07 May 2017].

A useful article explaining gathering metrics in software development. The formula for calculating test coverage was directly taken from this article.

[39] Google, "Google Python Style Guide," [Online]. Available: https://google.github.io/styleguide/pyguide.html. [Accessed 07 May 2017].

A very detailed guide to programming styles in Python. Used particularly for the section on comments and documentation.