Luke Halley

Internet Of Things

Project Semester 4 Report

20071820

24/05/17

Introduction

This report contains the following sections

1. **Overview Of The Entire Project**
2. **Technologies and Methods**
3. **My Role In The Project**
4. **Implementation**
5. **Problems & Solutions**
6. **Conclusion**

For my Project Semester 4 Report a group of students which included fourth year electronic engineering students and the second year Internet of Things were assigned a project.

After a few weeks of brainstorming the idea of a “living lamp” was decided on. Each person involved was put into groups, each group had different tasks (these will be further explained further on in the report)

As a group, we came together to decide each part of the project which we could then assign to groups which would include members who felt the task would suit there skillset.

Overview Of The Entire Project

The concept of the project was to turn an ordinary lamp bought from a hardware store with originally the only the function of shining light and moving between three axis, into a computer controlled, moving "being" of sorts.

This idea explained above instantly reminded me of Pixar's lamp used during they're intro before every Pixar movie (besides the first Toy Story where it came after) and it's one of the most recognized brands in film.

The concept our group had of the finished project had was for the lamp to be placed on a table in a public area around the WIT campus.

The appearance of the lamp would seem regular to any person passing by but using the technologies the lamp would act in an unsuspecting way!

The lamp would be implemented with various different technologies each developed by the groups mentioned above.

Technologies and Methods

To create the lamp multiple different technologies and methods were used;

Vision Systems:

Vision systems are a primary consideration for any manufacturer who is looking to improve quality or automate production. Vision systems can be thought of as computers with eyes that can identify, inspect and communicate critical information to eliminate costly errors, improve productivity and enhance customer satisfaction through the consistent delivery of quality products.

Primarily used for online inspection, vision systems can perform complex or mundane repetitive tasks at high speed with high accuracy and high consistency. Errors or deviations in the manufacturing process are immediately detected and relayed, allowing control modifications to be made on the fly to reduce scrap and minimize expensive downtime.

Vision systems are also deployed for non-inspection tasks, such as guiding robots to pick parts, place components, dispense liquids or weld seams.

Vision systems come in all shapes and sizes to suit any application need, but they all have the same core elements. Every vision system has one or more sensors that capture pictures for analysis and all include inspection software and a processing element that executes a user-defined program, or recipe, defining the inspection.

Additionally, all vision systems will provide some way of communicating results to complementary equipment for control or operator monitoring. That said, it is important to know that there are significant and important differences between vision systems that make one more suitable over another for any given application.

It is equally important to know and appreciate the importance of choosing the optimal lighting and optics for the job. Failure to do so may result in unexpected false rejects, or even worse, false positives.

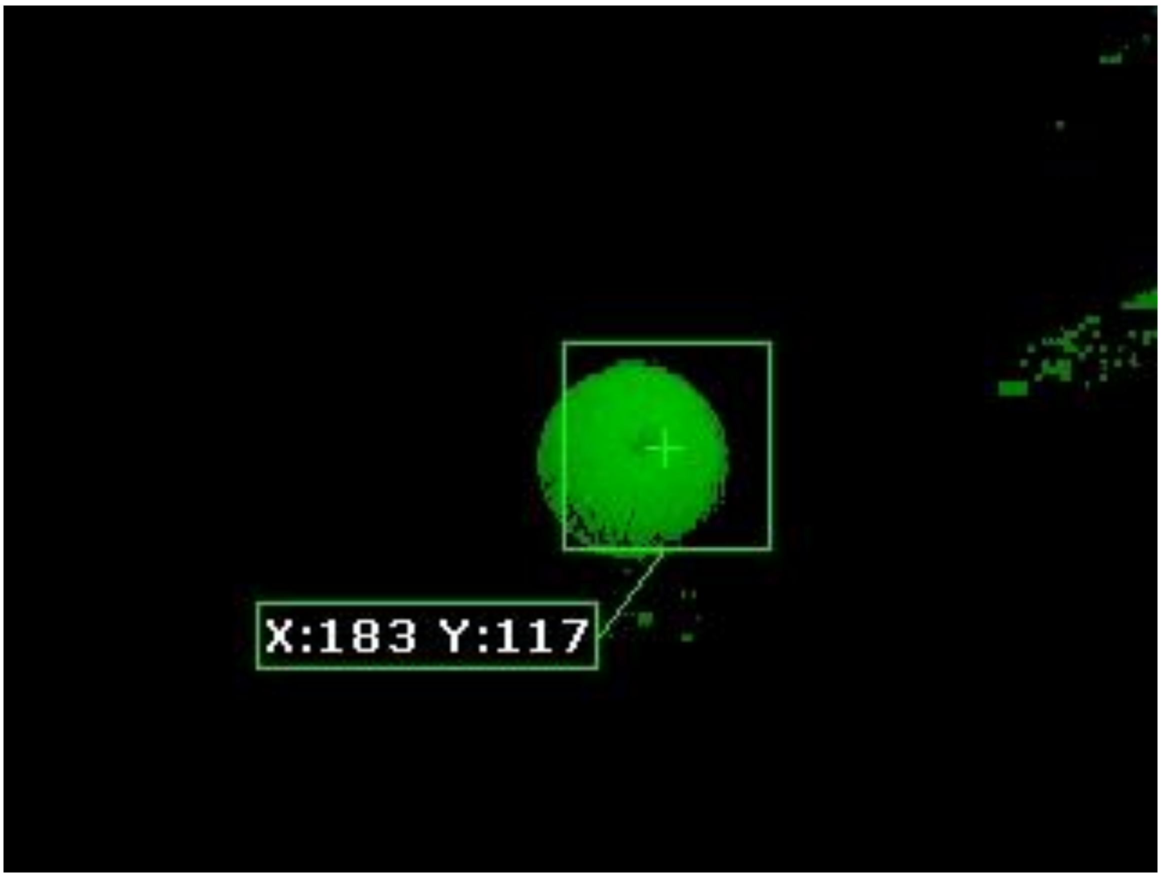
During this project the vision system RoboRealm was used, RoboRealm is an application for use in machine vision, computer vision, image analysis, and robotic vision systems.

Using an easy point and click interface, RoboRealm simplifies vision programming and allows you to easily experiment with many modules to achieve the desired result! We've compiled many image processing functions into a Windows (Win10, Win8, Win7, WinXP) based application that you can use with a webcam, TV Tuner, IP Camera, GenICam, Firewire, etc. Use RoboRealm to see the environment, process acquired images, analyse what needs to be done and send the needed signals to your hardware controllers.

Features:

- Easy to Use GUI Interface
- Hundreds of Image Processing Modules
- Camera Agnostic
- Realtime Parameter Changes
- Fully Supported Server API
- Multiple Image Sources (IEEE 1394 firewire, webcam, movie files, web images, etc.)
- Multiple Output Interfaces (File, Web, FTP, Email, etc.)
- Plugin Framework for Custom Modules
- Active Online Community with help from the experts

RoboRealm was used in order to detect people who were in proximity of the lamp.

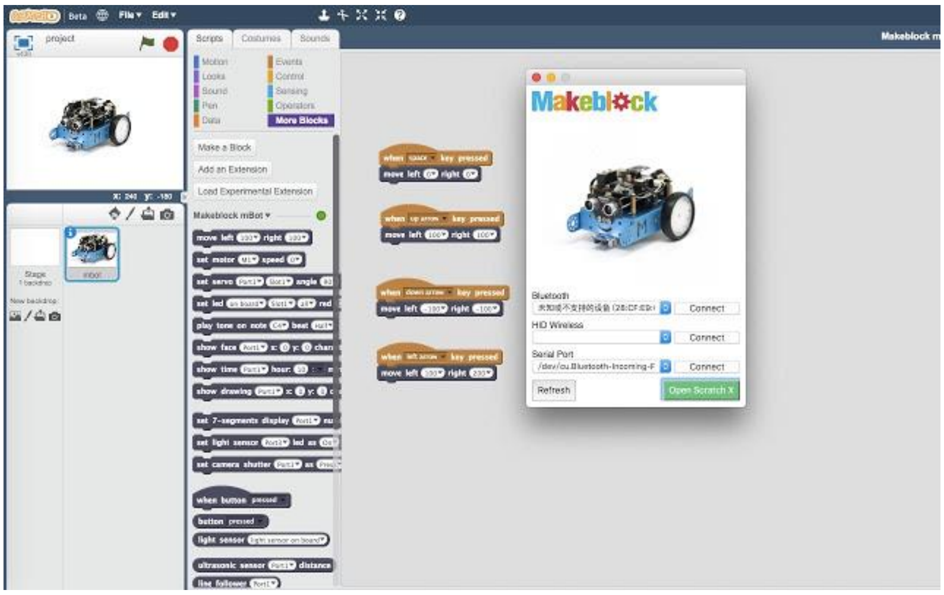


Scratch:

Scratch is a free visual programming language developed by the MIT (Massachusetts Institute of Technology) Media Lab. Scratch is used by students, scholars, teachers, and parents to easily create animations, games, etc. It provides a stepping stone to the more advanced world of computer programming. It can also be used for a range of educational and entertainment constructionist purposes from math and science projects, including simulations and visualizations of experiments, recording lectures with animated presentations, to social sciences animated stories, and interactive art and music. Viewing the existing projects available on the Scratch website, or modifying and testing any modification without saving requires no online registration.

A sub section of the Scratch programming was used called **ScratchX**. ScratchX is a platform that enables people to test experimental functionality built by developers for the visual programming language Scratch.

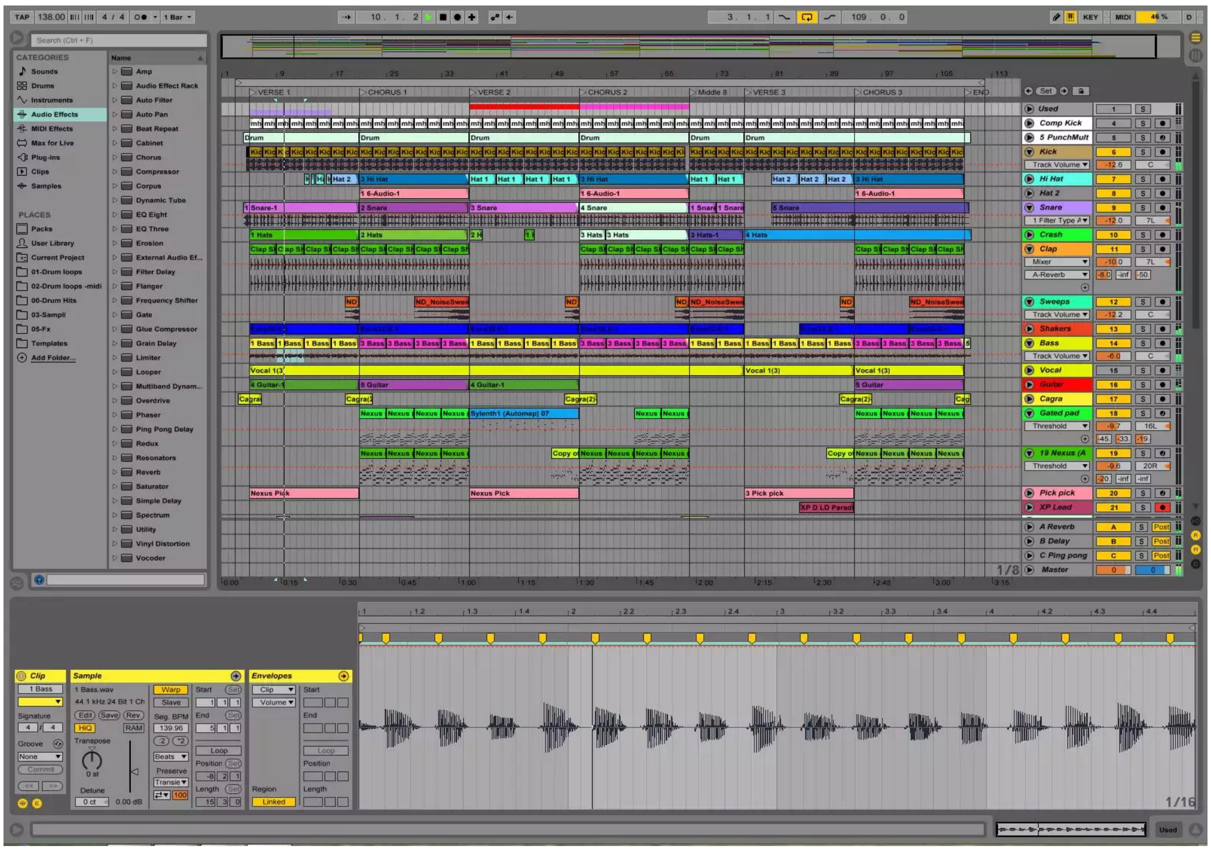
The idea of using Scratch was to create a custom interface which people who did not have any experience with any type of programming language to interact with the three lamps. Actions like moving their axis forward, backwards, sideways etc. were implemented.



Ableton Live:

Ableton Live is a software music sequencer and digital audio workstation for macOS and Windows. The latest major release of Live, Version 9, was released on March 5, 2013.

In contrast to many other software sequencers, Live is designed to be an instrument for live performances as well as a tool for composing, recording, arranging, mixing and mastering. It is also used by DJs, as it offers a suite of controls for beatmatching, crossfading, and other effects used by turntablists, and was one of the first music applications to automatically beatmatch songs.



Software Synthesizer (Serum):

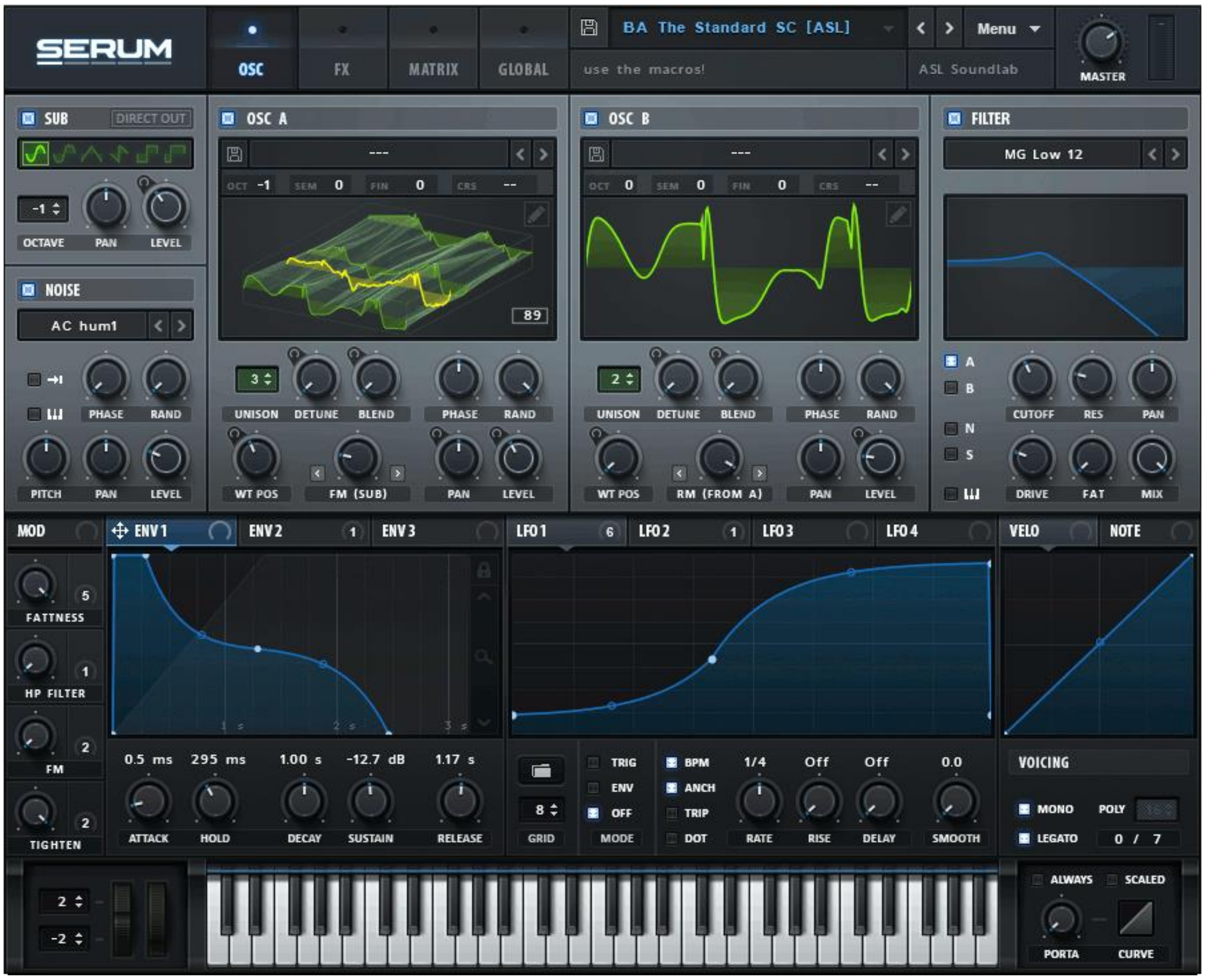
A software synthesizer, also known as a softsynth, is a computer program, or plug-in that generates digital audio, usually for music. Computer software that can create sounds or music is not new, but advances in processing speed are allowing softsynths to accomplish the same tasks that previously required dedicated hardware. Softsynths are usually cheaper and more portable than dedicated hardware, and easier to interface with other music software such as music sequencers.

Many popular hardware synthesizers are no longer manufactured, but have been emulated in software. The emulation can even extend to having graphics that model the exact placements of the original hardware controls. Some simulators can even import the original sound patches with accuracy that is nearly indistinguishable from the original synthesizer. Popular synthesizers such as the Minimoog, Yamaha DX7, Korg M1, Prophet-5, Oberheim OB-X, Roland Jupiter 8, ARP 2600 and dozens of other classics have been recreated in software.

Some softsynths are heavily sample based, and frequently have more capability than hardware units, since computers have fewer restrictions on memory than dedicated hardware synthesizers. Some of these sample based synthesizers come with sample libraries many gigabytes in size. Some are specifically designed to mimic real world instruments such as pianos. Many sample libraries are available in a common format like WAV or SoundFont, and can be used with almost any sampler based softsynth.

The major downside of using softsynths can often be more latency (delay between playing the note and hearing the corresponding sound). Decreasing latency requires increasing the demand on the computer's processor. When the soft synthesizer is running as a plug-in for a host sequencer, both the soft synth and the sequencer are competing for processor time. Multi-processor computers can handle this better than single-processor computers. As the processor becomes overloaded, sonic artifacts such as "clicks" and "pops" can be heard during performance or playback. When the processor becomes completely overloaded, the host sequencer or computer can lock up or crash. Increasing buffer size helps, but also increases latency. However modern professional audio interfaces can frequently operate with extremely low latency, so in recent years this has become much less of a problem than in the early days of computer music.

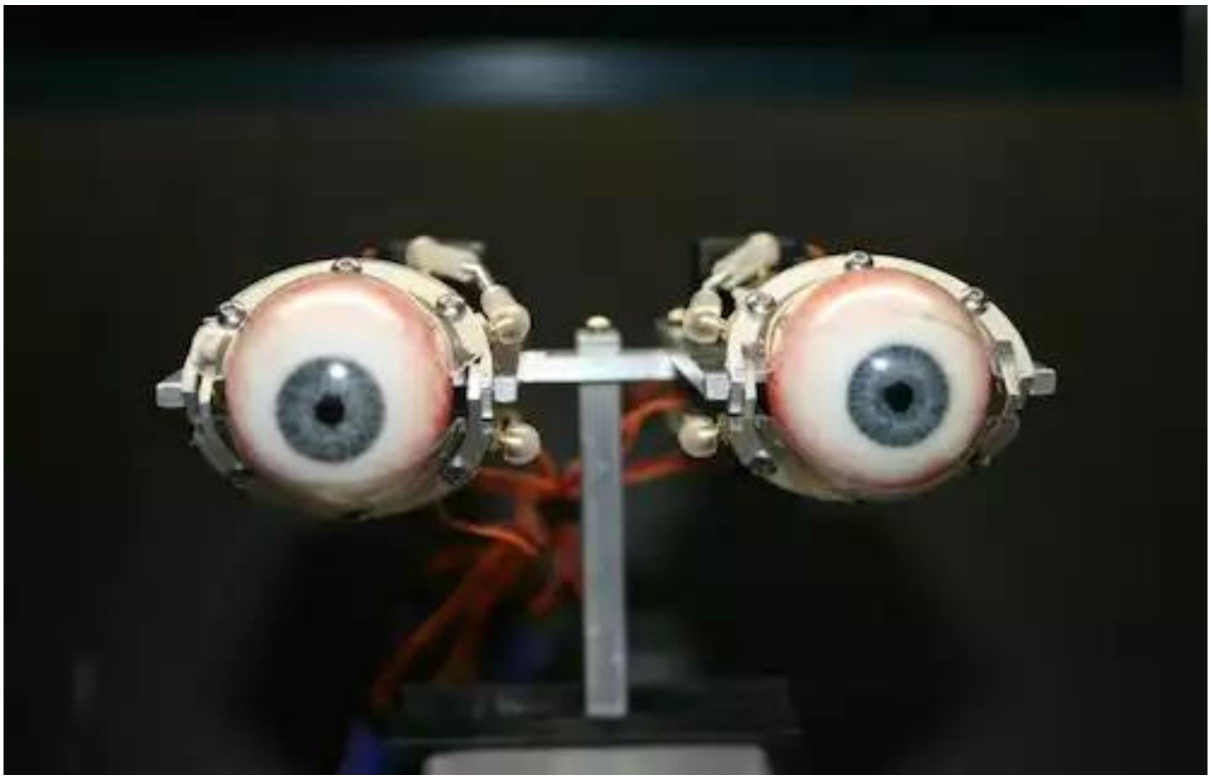
It is also possible to generate sound files off-line, meaning sound generation does not have to be in real time, or live. For example, the input could be a MIDI file and the output could be a WAV file or an MP3 file. Playing a WAV or MP3 file simply means playing a pre-calculated wave-form(s). The advantage of off-line synthesis is that the software can spend as much time as it needs to generate the resulting sounds, potentially increasing sound quality. It could take 30 seconds of computing time to generate 1 second of real-time sound, for example. The disadvantage is that changes to the music specifications cannot be heard immediately.



Animatronics:

Animatronics refers to the use of robotic devices to emulate a human or an animal, or bring lifelike characteristics to an otherwise inanimate object. A robot designed to be a convincing imitation of a human is more specifically labelled as an android. Modern animatronics have found widespread applications in movie special effects and theme parks and have, since their inception, been primarily used as a spectacle of amusement.

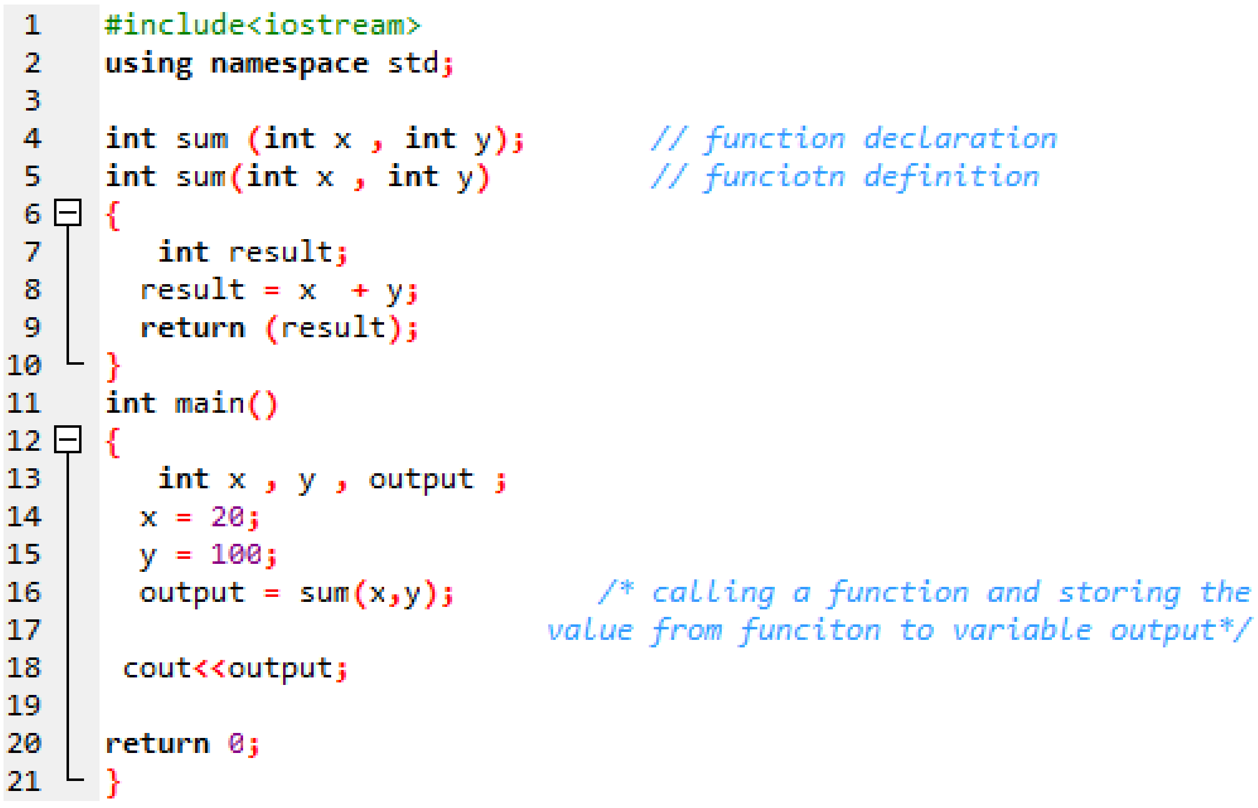
Animatronics is a multi-disciplinary field which integrates anatomy, robots, mechatronics, and puppetry resulting in lifelike animation. Animatronic figures are often powered by pneumatics, hydraulics, and/or by electrical means, and can be implemented using both computer control and human control, including teleoperation. Motion actuators are often used to imitate muscle movements and create realistic motions in limbs. Figures are covered with body shells and flexible skins made of hard and soft plastic materials and finished with details like colours, hair and feathers and other components to make the figure more realistic.



C Programming Language:

C is a general-purpose, imperative computer programming language, supporting structured programming, lexical variable scope and recursion, while a static type system prevents many unintended operations. By design, C provides constructs that map efficiently to typical machine instructions, and therefore it has found lasting use in applications that had formerly been coded in assembly language, including operating systems, as well as various application software for computers ranging from supercomputers to embedded systems.

C was originally developed by Dennis Ritchie between 1969 and 1973 at Bell Labs, and used to re-implement the Unix operating system. It has since become one of the most widely used programming languages of all time, with C compilers from various vendors available for the majority of existing computer architectures and operating systems. C has been standardized by the American National Standards Institute (ANSI) since 1989 (see ANSI C) and subsequently by the International Organization for Standardization (ISO).

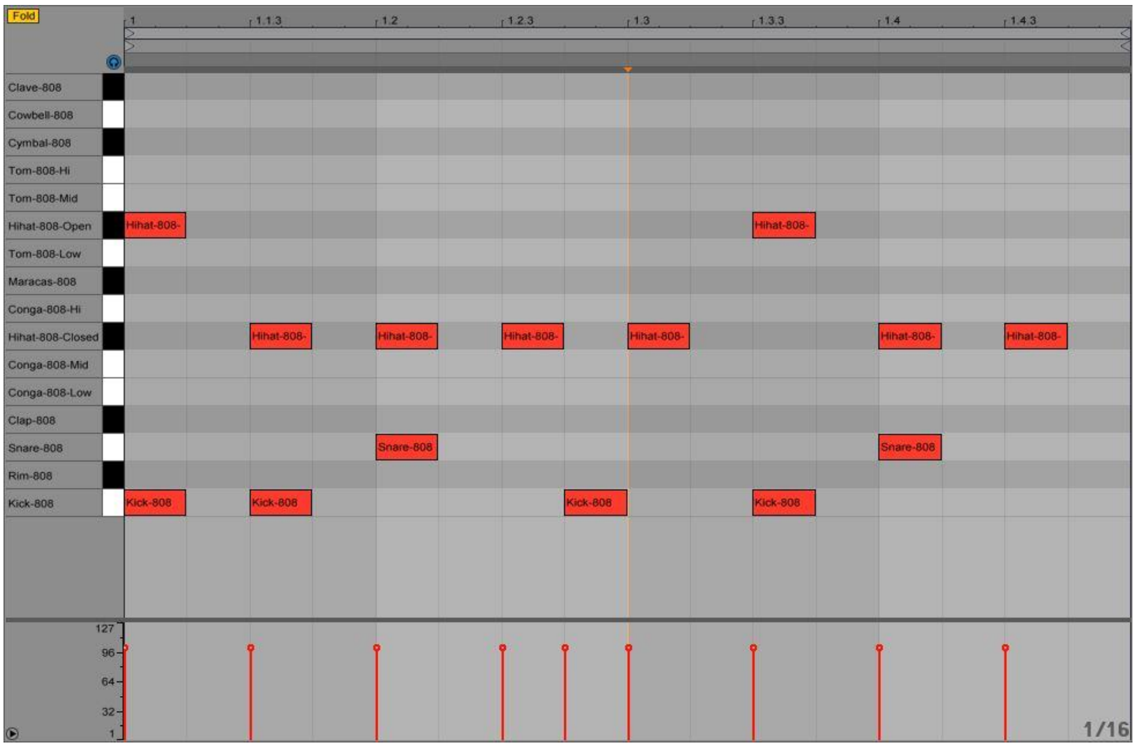


```
#include<iostream>
using namespace std;

int sum (int x , int y);        // function declaration
int sum(int x , int y)          // funciotn definition
{
   int result;
  result = x  + y;
  return (result);
}
int main()
{
   int x , y , output ;
  x = 20;
  y = 100;
  output = sum(x,y);        /* calling a function and storing the
                       value from funciton to variable output*/

 cout<<output;

return 0;
}
```

Midi:

MIDI is a technical standard that describes a protocol, digital interface and connectors and allows a wide variety of electronic musical instruments, computers and other related devices to connect and communicate with one another. A single MIDI link can carry up to sixteen channels of information, each of which can be routed to a separate device.

MIDI carries event messages that specify notation, pitch and velocity, control signals for parameters such as volume, vibrato, audio panning, cues, and clock signals that set and synchronize tempo between multiple devices. These messages are sent via a MIDI cable to other devices where they control sound generation and other features. A simple example of a MIDI setup is the use of a MIDI controller such as an electronic musical keyboard to trigger sounds created by a sound module. This MIDI data can also be recorded into a hardware or software device called a sequencer, which can be used to edit the data and to play it back at a later time.:4 Advantages of MIDI include compactness (an entire song can be coded in a few hundred lines, i.e. in a few kilobytes), ease of modification and manipulation and a wide choice of electronic instruments and synthesizer or digitally-sampled sounds.



My Role in The Project:

During this project, I was placed into a group alongside Gianni Kapele. The reason we were put into a group together is due to both of us having a mutual interest in music production and engineering.

Sounds are attached to simulations with minimal regard to synchronization or appropriateness to the motion events causing them. Does human movement have constraints similar to musical instruments which might suggest something akin to idiomatic expression? Is there a distinct character to the movement of the hands? What is finger music? What is running music? What is the sound of one hand clapping? These questions may be answered by allowing the physicality of movement to impact on musical material and processes. These relationships may be established by viewing the body and space as musical instruments, free from the associations of acoustic instruments, but with similar limitations that can lend character to sound through idiomatic movements.

Traditional studies in orchestration, as well as studies in sound synthesis, begin by examining the physical properties of instruments and their methods for producing sound. Physical constraints produce unique timbral characteristics, and suggest musical material that will be idiomatic or appropriate for a particular instrument's playing technique. These reflect the weight, force, pressure, speed, and range used to produce sound. In turn, the sound reflects, in some way, the effort or energy used to create it. The fact that brass tones add upper partials as they grow louder is a classic example.

When movement is linked with sound the human mind perceives it as a more realistic animation in real life. Our groups job was to synthesize sound which would match the motion of the lamp moving.

Implementation:

The following steps were involved in the implementation of mapping audio to the lamp:

<u>Learning to code MIDI:</u> This involved doing a large amount of research online. MIDI messages are sent as a time sequence of one or more bytes (8 bits). The first byte is a STATUS byte, often followed by DATA byteswitch additional parameters. A STATUS byte has bit 7 set to 1 and a DATA byte has bit 7 set to 0.

The main messages used were the NOTE ON and NOTE OFF messages. The NOTE ON message is sent when the performer hits a key of the music keyboard. It contains parameters to specify the pitch of the note as well as the velocity (intensity of the note when it is hit). When a synthesizer receives this message, it starts playing that note with the correct pitch and force level. When the NOTE OFF message is received, the corresponding note is switched off by the synthesizer.

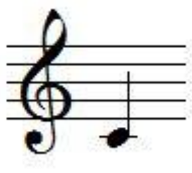
The **NOTE ON** message is structured as follows:

- Status byte: 1001 CCCC
- Data byte 1: 0PPP PPPP
- Data byte 2: 0VVV VVVV

where:

"CCCC" is the MIDI channel (from 0 to 15)
"PPP PPPP" is the pitch value (from 0 to 127)
"VVV VVVV" is the velocity value (from 0 to 127)

The pitch value determines the frequency of the note to be played. It goes from 0 to 127, with the middle C note being represented by the value of 60:

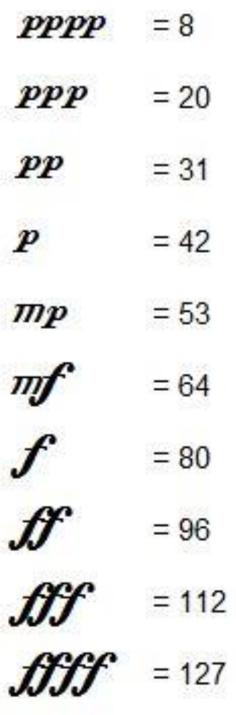


The value is represented in half steps, so that C# will be 61, D will be 62, ...

To transpose a note one octave higher, add 12 to its pitch value. By using MIDI, transposition is very simple as it is done simply by adding or subtracting a fixed value.

Be cautious however about the range of MIDI notes that goes from 0 to 127. By adding for instance 4 octaves (+48) to a note of value 96, the total is 144, which is outside the range and may be truncated to 16 (144 - 128) so that a very low note will result.

The velocity value normally goes from 1 to 127, covering the range from a practically inaudible note up to the maximum note level. It basically corresponds to the scale of nuances found in music notation, as follows (it is more indicative than exact values):



pppp	= 8
ppp	= 20
pp	= 31
p	= 42
mp	= 53
mf	= 64
f	= 80
ff	= 96
fff	= 112
ffff	= 127

In basic synthesizers, the velocity value is used only to determine the force with which the note is played, the only effect being a note that is louder or softer in volume.

In more sophisticated synthesizer, this value will also affect the sound quality. Indeed, on a real piano, hitting a note harder will not only affect its loudness but also the quality of the sound itself, the timber. This is practically the case with any real instrument.

There is a special case if the velocity is set to zero. The **NOTE ON** message then has the same meaning as a **NOTE OFF** message, switching the note off.

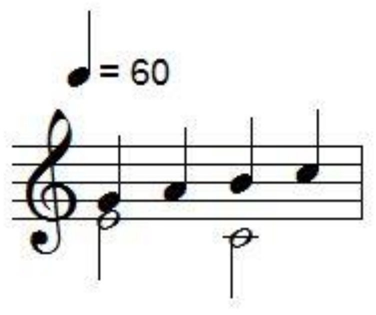
The **NOTE OFF** message is structured as follows:

- Status byte: 1000 CCCC
- Data byte 1: 0PPP PPPP
- Data byte 2: 0VVV VVVV

where CCCC and PPPPPPP have the same meaning as above. The VVVVVVV is the release velocity, which is very rarely used. By default, set it to zero.

When you send a **NOTE ON** message to a synthesizer, this note starts playing. Meanwhile, you can send other **NOTE ON** messages, with different note pitches, so as to hear a chord. However, you need to keep track of the notes that are playing, so that you can send a corresponding **NOTE OFF** for each note, otherwise there will be stuck notes playing forever.

Let's take an example. What are the MIDI messages needed to play the following measure?



As the time dimension must be present to hear the music, here is the time sequence of the MIDI messages that you need to send to a synthesizer to have it play the above music on channel 1 (remember, coded as 0), with a velocity of 64 (*mezzo forte*), in hexadecimal (0x means hexadecimal notation):

- t=0: **0x90** - **0x40** - **0x40** (Start of E3 note, pitch = 64)
- t=0: **0x90** - **0x43** - **0x40** (Start of G3 note, pitch= 67)

- t=1: **0x80** - **0x43** - **0x00** (End of G3 note, pitch=67)
- t=1: **0x90** - **0x45** - **0x40** (Start of A3 note, pitch=69)
- t=2: **0x80** - **0x45** - **0x00** (End of A3 note, pitch=69)
- t=2: **0x80** - **0x40** - **0x00** (End of E3 note, pitch=64)
- t=2: **0x90** - **0x3C** - **0x40** (Start of C3 note, pitch = 60)
- t=2: **0x90** - **0x47** - **0x40** (Start of B3 note, pitch= 71)
- t=3: **0x80** - **0x47** - **0x00** (End of B3 note, pitch= 71)
- t=3: **0x90** - **0x48** - **0x40** (Start of C4 note, pitch= 72)
- t=4: **0x80** - **0x48** - **0x00** (End of C4 note, pitch= 72)
- t=4: **0x80** - **0x3C** - **0x40** (End of C3 note, pitch = 60)

"t" represents the time in seconds. The score plays at 60 beats per minute, so each quarter note is 1 second.

Turning baseboard into MIDI controller:

Equipment used:

- Baseboard
- 6 short cables
- 5 long cables
- A prototyping breadboard
- 3 x 10k Linear Potentiometers (the potentiometer should normally be labelled B10K)
- Hairless Midi Serial Bridge (Software for sending/receiving midi data)
- LoopBe1 (Virtual Midi Device Software)

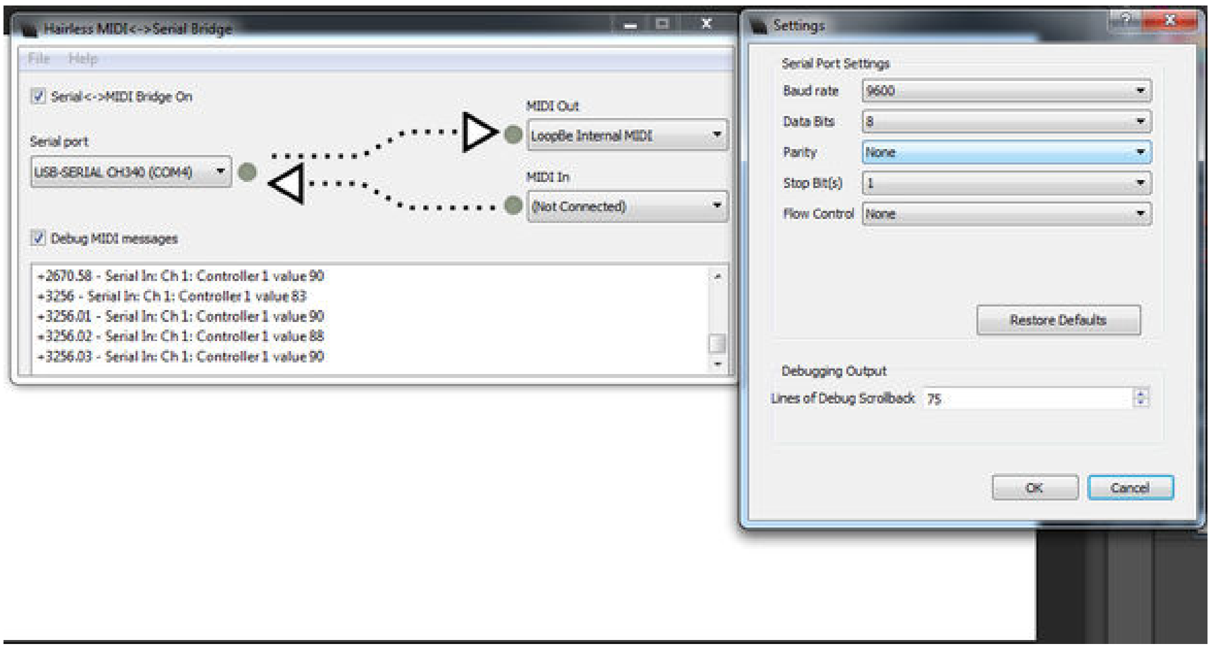
The next step was installing LoopBe1 and Hairless Midi Serial Bridge.

Using this initial code to get it up and running

```
int val = 0; //Our initial pot values. We need one for the first value and a second to test if there has b
een a change made. This needs to be done for all 3 pots.
int lastVal = 0;

int val2 = 0;

int lastVal2 = 0;

int val3 = 0;

int lastVal3 = 0;

void setup()

{

  Serial.begin(9600);      // Set the speed of the midi port to the same as we will be using in the Hairl
ess Midi software

}

void loop()

{

  val = analogRead(0)/8;   // Divide by 8 to get range of 0-127 for midi

  if (val != lastVal) // If the value does not = the last value the following command is made. This is be
cause the pot has been turned. Otherwise the pot remains the same and no midi message is output.

  {
```

```
  MIDImessage(176,1,val);}        // 176 = CC command (channel 1 control change), 1 = Which Control
, val = value read from Potentionmeter 1 NOTE THIS SAYS VAL not VA1 (lowercase of course)

  lastVal = val;

  val2 = analogRead(1)/8;   // Divide by 8 to get range of 0-127 for midi

  if (val2 != lastVal2)

  {

  MIDImessage(176,2,val2);}        // 176 = CC command, 2 = Which Control, val = value read from Pot
entionmeter 2

  lastVal2 = val2;


  val3 = analogRead(2)/8;   // Divide by 8 to get range of 0-127 for midi

  if (val3 != lastVal3)

  {

  MIDImessage(176,3,val3);}        // 176 = CC command, 3 = Which Control, val = value read from Pot
entionmeter 3

  lastVal3 = val3;

delay(10); //here we add a short delay to help prevent slight fluctuations, knocks on the pots etc. Ad
ding this helped to prevent my pots from jumping up or down a value when slightly touched or knoc
ked.

}

void MIDImessage(byte command, byte data1, byte data2) //pass values out through standard Midi
Command

{

  Serial.write(command);

  Serial.write(data1);

  Serial.write(data2);
```

```
}
```



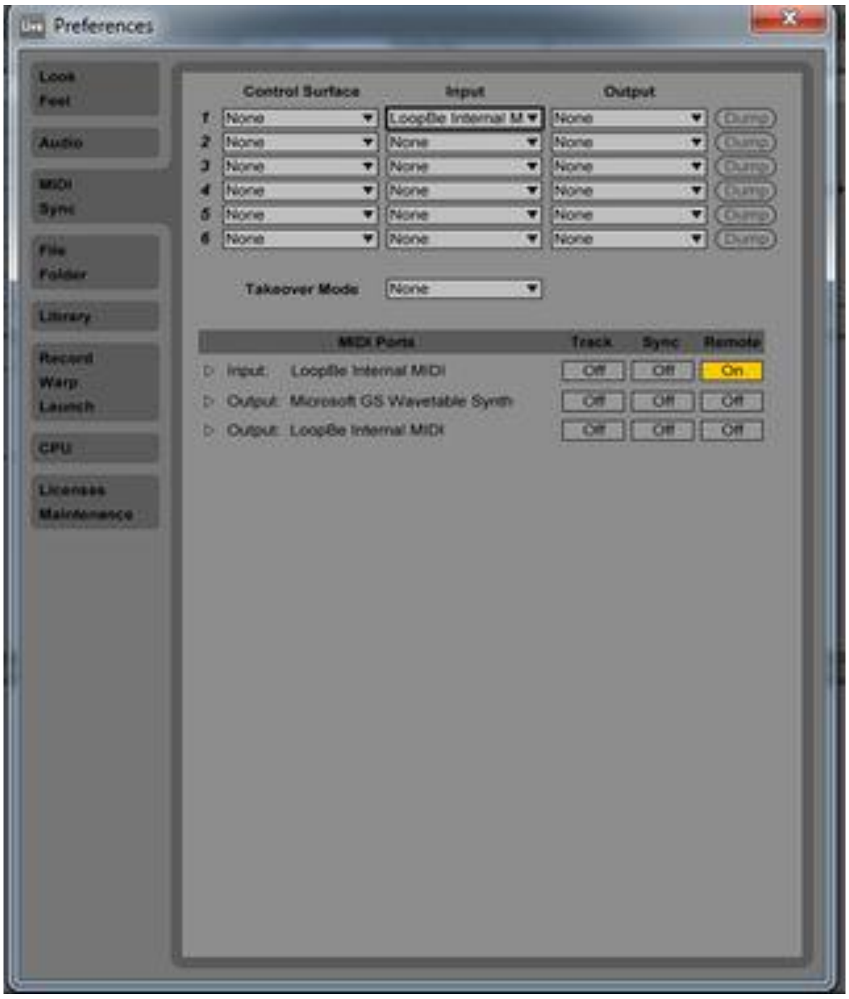
☑ Serial<->MIDI Bridge On
Serial port: USB-SERIAL CH340 (COM4)
MIDI Out: LoopBe Internal MIDI
MIDI In: (Not Connected)
☑ Debug MIDI messages

```
+2670.58 - Serial In: Ch 1: Controller 1 value 90
+3256 - Serial In: Ch 1: Controller 1 value 83
+3256.01 - Serial In: Ch 1: Controller 1 value 90
+3256.02 - Serial In: Ch 1: Controller 1 value 88
+3256.03 - Serial In: Ch 1: Controller 1 value 90
```

Serial Port Settings

Baud rate	9600
Data Bits	8
Parity	None
Stop Bit(s)	1
Flow Control	None

Debugging Output
Lines of Debug Scrollback: 75



	Control Surface	Input	Output	
1	None	LoopBe Internal M	None	Dump
2	None	None	None	Dump
3	None	None	None	Dump
4	None	None	None	Dump
5	None	None	None	Dump
6	None	None	None	Dump

Takeover Mode: None

MIDI Ports	Track	Sync	Remote
▷ Input: LoopBe Internal MIDI	Off	Off	On
▷ Output: Microsoft GS Wavetable Synth	Off	Off	Off
▷ Output: LoopBe Internal MIDI	Off	Off	Off

CAN Bus:

A Controller Area Network (CAN bus) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. It is a message-based protocol, designed originally for multiplex electrical wiring within automobiles, but is also used in many other contexts.

The CAN bus was used to send signals from all the boards used to the PC which was running Ableton.

Mapping in Ableton:

See the following for a step by step guide mapping in Ableton:
https://www.ableton.com/en/manual/midi-and-key-remote-control/

The midi messages that were sent were then mapped to the pitch bend parameter of Serum. This allowed the synth to follow the movement of the lamp as it rose and fell.

Three different sounds were mapped for each axis of the lamp which created further depth to the project.

The concept of the one shots were also discussed which would have included sounds like a motor starting or ending at the start and end of various movements. However due to time constrains this wasn't implemented.

<u>Sound Design:</u>

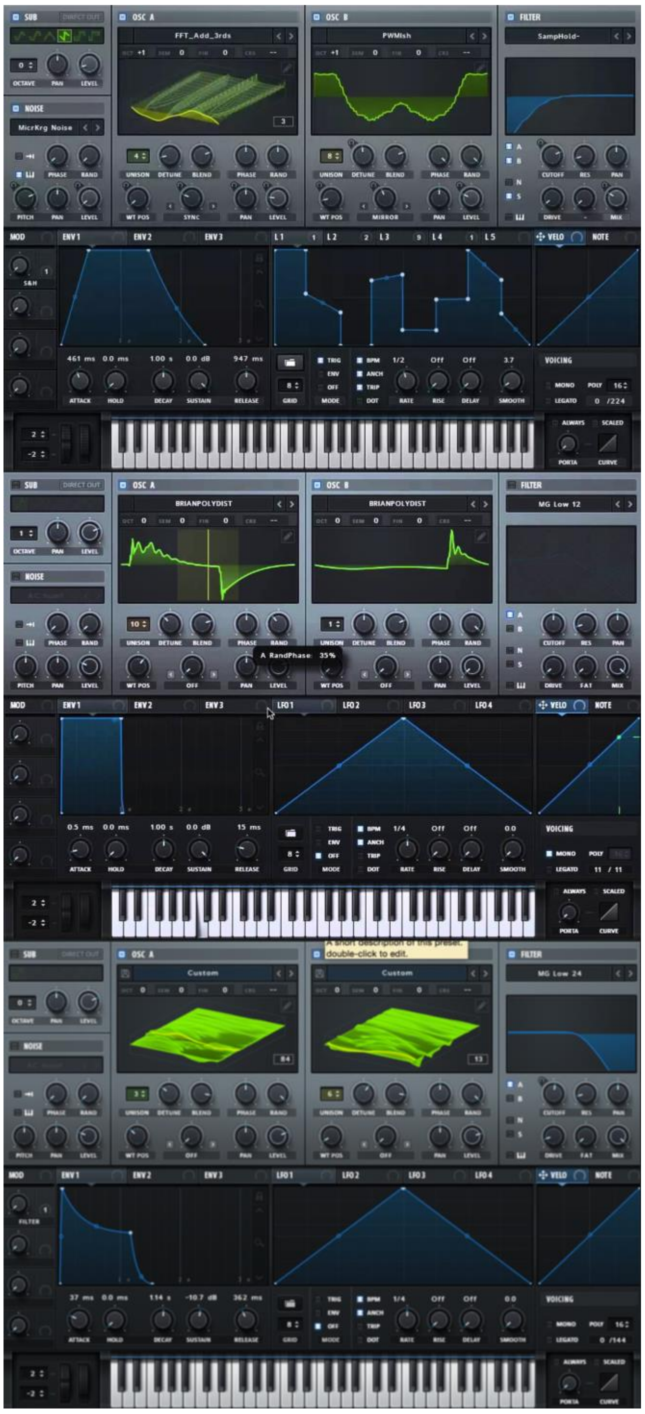
The final part of the project was to use Serum and its wavetables to create realistic movement sounds for the lamp. Various methods were used such as envelope automation, detuning, compression and reverb when creating the sounds.

A metallic and high-pitched sound was used for the top axis of the lamp.

A dull and warm sound was used for the middle axis.

A heavy and robotic bottom axis.

The three presets can be seen to the right:



Problems and Solutions:

Originally, we were going to use MIDI messages to play the sounds which led to us trying to use WAV samples for the sounds and use a VST which would change the pitch dynamically. This VST was called Soundshifter by Waves. The problem with this was Ableton could not resume MIDI message after stopping and would have to be returned to its starting point for it to play again, we then tried WAVs but the VST we were using made the samples lose a lot of their quality's and lost sync with the movement of the lamp.

Finally, we came up with the solution to not use the actual play button in Ableton but to use the Mute/Unmute button so the sound would constantly play but could not be heard when the lamp wasn't moving.

Conclusion:

The following project was quite interesting to me as I have been producing music for around two and a half years but would have never had thought that my knowledge for Ableton would come into my College education.

Although a lot of aspects of the initial project idea were covered some interesting features are still left to be completed. More time would have been beneficial to the project.

The technology used in the project was quite up to date and modern so another party could easily pick up where our group has left and add these features.