UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

GOOIZER A TANGIBLE USER INTERFACE

STUDENT NAME: TUSHAR MITTAL
STUDENT ID: 16123921
SUPERVISOR: MIKAEL FERNSTORM
COURSE: COMPUTER GAMES
ACADEMIC YEAR: 2019-2020

Table of Contents

1 Project Summary

The aim of the project is to develop a tangible user interface which would use computer vision to detect changes in shape and size of brightly coloured putty to record and produce sounds. The project involves research into image, video and audio processing.

The project will be developed in prototypes which would be tested in order to improve on user feedback.

This project will also help in finding if people are interested in interacting with an installation to produce sounds and tangible user interfaces in general. In this digital world where everything can be done easily on the phones will the participants for the study be interested in interacting with Gooizer. This project will also help me in finding out general development issues of a tangible user interface as user feedback is crucial in developing a human friendly interface.

2 Introduction and Objectives

2.1 General Introduction

“Tangible” refers to something that can be touched, and “user interface” refers to a device through which a person can interact and control a machine or a device. Therefore, the Tangible user interface (TUI) [1] is an interface in which the operator interacts with the machine by moving physical objects in the real world. Originally known as a Graspable user interface, the general idea is that by integrating physical elements into a user interface, it promotes greater satisfaction and ease of use. The main difference between the Graphical user interface (GUI) and the Tangible interface is that the GUI exists only in the digital world, whereas the TUI connects the digital and the physical world.

A user interface can be classified as tangible if it possesses the following characteristics [2]:

- Space-multiplex for input and output
- Concurrent access and manipulation of interface components
- Strong specific devices
- Spatially aware computational devices
- Spatial reconfigurability of devices

TUI has shown to improve learning performance through multimodal feedback, this is because users can feel the interface, which improves engagement and satisfaction. Results from various studies suggest that tangible environments support playful learning among children.

TUI is also subject to more tear and wear because the physical objects are moved for interaction and since it’s physical it can also degrade with time.

2.2 Objectives of Proposed Work

This project aimed to create a Tangible user interface that would allow the users to record, assign, and process sounds based on the colour and shape of blobs of putty. The changes in the shape and colour would be detected by a computer vision system programmed in Processing, which would then communicate with Pure Data, which would process and play the sounds. The resulting system was an installation.

The main objectives were:

- Create a tangible user interface that utilises computer vision and physical human interaction to play sounds.
- Make the interface as human-friendly as possible and learn more about user testing and human-computer interaction.
- Gain knowledge of computer vision and image processing algorithms.
- Learn the languages Processing and Pure Data.

2.3 Motivating Factors

Always being interested in how computer vision works and how different colours are processed from a video, this project seemed like a perfect way for me to discover new things. On top of this, having never worked on a project which would involve a lot of human interaction and the quality of which would depend on reviews from user testing, this project seemed like a perfect challenge for me to increase my knowledge and learn new things. While choosing this project, I knew this would be a very challenging project, but always being driven by challenging things, I was interested in this.

I think this project will help me learn and maybe come up with ideas that I can implement in developing some new kinds of games. I am interested in creating a game that will be played using a tangible user interface.

Another important motivation for choosing this project was that during my internship, I thought if there were ways using which one can help mute people talk with their hand gestures. Like, giving them gloves of a particular colour and identify which actions they are making and producing the desired sounds from those actions. The project is not exactly like the idea I had, but it does have a few similarities in using computer vision to detect changes in shape and producing the desired sounds.

My desire for challenge and to learn new things was the core reason for choosing this project. Being new to Processing and Pure Data, after doing some research into the two, I thought I could use processing for many things. What better way to learn a language than doing a project because when working on

a project, one must do a lot of research into the topic and develop programs that help in gaining more knowledge and experience at the same time.

3 Research

3.1 Similar Projects

The first task of my project was to find and learn about some similar products and how users interact with them. I found out that there aren't many sound processing TUIs that are available to the general public and that many are still in the development phase, but all of them ask the user to interact differently.

3.1.1 Reactable

One of the related projects is Reactable [3] developed by four graduate students from the Universitat Pompeu Fabra in Barcelona, Spain. Reactable could be considered similar and yet different from Gooizer. The Reactable is more advanced and can support more sounds and beats. It is essentially a music synthesizer the difference between the reactable and a typical synthesizer is that the participants manipulate sound with blocks on a round tabletop. By rotating or moving the blocks on the table, a person can create a variety of sounds, beats, and notes, creating an electronic soundscape.

FIGURE 1. TABLE TOP OF REACTABLE
http://reactable.com/wp-content/uploads/2014/12/reactable-hotel-museum.jpg

FIGURE 2. SHOWS PEOPLE INTERACTING WITH REACTABLE
https://upload.wikimedia.org/wikipedia/commons/thumb/e/e3/Reactable_Multitouch.jpg/1200px-Reactable_Multitouch.jpg

The table itself acts as a display, and when a tangible is placed on it, various symbols appear on the table such as waveforms, circles, sweeping lines, etc. These symbols show what each tangible is doing.

The table is translucent, and under the table is a video camera aimed at the underside of the table and inputting video to a personal computer. The table also has a video projector under it projecting at its surface from below, turning it into a display. Computer vision is used to detect the changes in the tangibles, and depending on which tangible is being used, the appropriate data is then sent to the computer, which creates the sound using an audio engine developed using Pure Data and SuperCollider.

The reactable is an excellent example of using computer vision and Pure Data to produce a TUI.

3.1.2 Scrapple

Scrapple [4][5] is a tangible user interface supporting audio and visual processing. It is an installation in which everyday use objects are placed on the table and interpreted as sound-producing marks. Scrapple examines the table surface as music notation and plays the music in real-time by looking at the objects lying there.

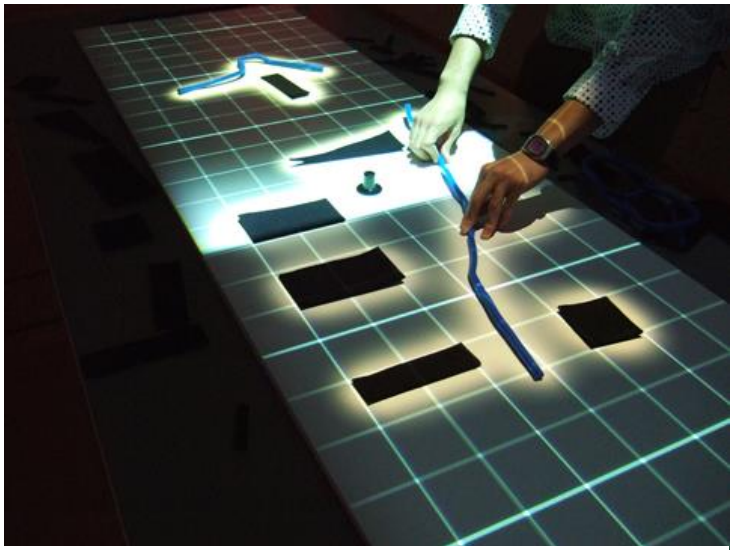


FIGURE 3. A PERSON INTERACTING WITH SCRAPPLE
http://www.multimedialab.be/blog/wp-content/uploads/2007/02/292870176_263647e784_b_482.jpg

Video projections on the Scrapple table transform it into an Augmented experience. The objects placed by users are elaborated through bright and beautiful graphics. The 3-meter long table and having a 4-second audio loop between sound production helps the participants to experiment with the interface freely.

It was created with support from the artist residency program of the Ars Electronica Futurelab, and since been acquired into the permanent collections of the San Francisco Exploratorium (2013) and the Universum Center, Bremen (2015).

3.2 Design

There was a bit of research involved in what would be the best installation of the Gooizer, which involved how to put the lights and how the user will interact with the GUI. The installation design didn't turn out as the one proposed due to unforeseen circumstances and unavailability of resources.

In this section, I have listed both the proposed installation design and the final installation that was achieved.

3.2.1 Proposed Installation

For the installation of the Gooizer, I planned to have a tabletop with a black paper, which is divided into three different sections, each section being scanned for different colours. The table would be lit with two lights on either side of the table to make the colours are clearly visible to the camera, and the lights will also help in low light situations. The plan was to have a camera on the top to scan the table. There would also be a laptop so that the users can interact with the GUI stop and start the program as they wish, etc. The GUI will also help me to set up Gooizer by calibrating it for the current environment lighting. The design sketches show a basic representation of what the installation should have looked like.

The design of the installation of the Gooizer was not final. This design was subject to change according to what I think would work best and would be ergonomically adequate for users.

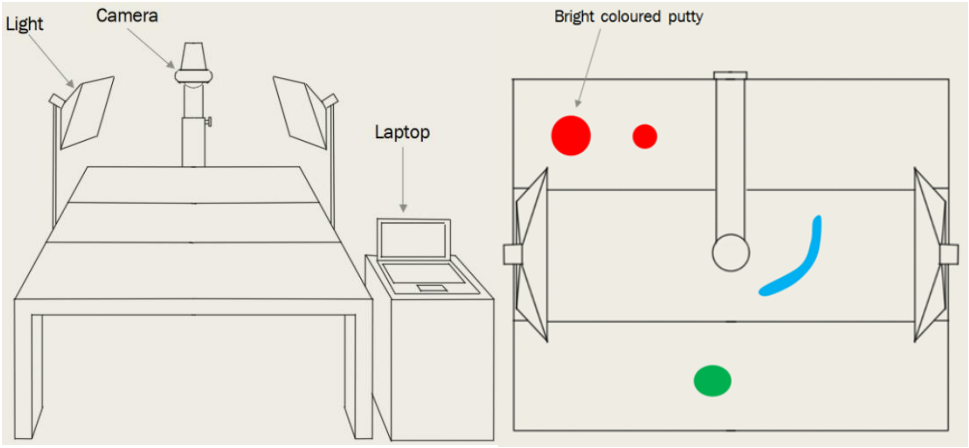


FIGURE 4. PROPOSED DESIGN OF GOOIZER

3.2.2 Final Installation

As stated above, the final installation was changed and couldn't be the same as the proposed plan. The installation was done with whatever resources were available and was built at home. The pictures show the final installation that was achieved.

Since the lamps and a table were not available, the interacting surface was built on the floor, and the room lighting was used to light up the surface.

FIGURE 5. ACTUAL INSTALLATION OF GOOIZER

FIGURE 6. ACTUAL INSTALLATION OF GOOIZER

The proposed idea of black paper was changed with white because the black one caught the stains of the colour of the putty, which then effected the scanning as non-existing shapes were being detected. Also, the section lines couldn't be drawn on the paper because a webcam has a bit of fisheye effect, which causes the center to be close, and as further to the edges you go, the lines would curve. This fisheye effect affected what the division lines were for the camera and what the user saw. Therefore, the lines had to be shown on the GUI, so when interacting with the surface, the user can look at the screen and see where he/she is putting the putty.

3.3 Processing

Since it was decided to use Processing [6] for computer vision, I had to do much research into processing as I had no prior knowledge of it. The good this is that Processing comes with a vast number of tutorials and documentation helping new developers a lot. Processing is a software sketchbook and a language for learning how to code within the context of visual arts. It is open-source and uses Java as its core language with additional simplifications such as extra classes and operations. Processing has a vast number of libraries for different functions like computer vision, audio processing, GUI, digital drawings, etc. Some of the features of processing are:

1. Free to download and open source.
2. Interactive programs with 2D, 3D, PDF, or SVG output.
3. OpenGL integration for accelerated 2D and 3D.

4. For GNU/Linux, Mac OS X, Windows, Android, and ARM.
5. Over 100 libraries extend the core software.

3.4 Pure Data

Pure Data [7] is an open source visual programming environment for multimedia. It enables musicians, visual artists, performers, researchers, and developers to create software graphically without writing lines of code. It can be used to process and generate sound, video, 2D/3D graphics, and interface sensors, input devices, and MIDI. It can also work over local and remote networks to integrate technology. It is suitable for learning basic multimedia processing and visual programming methods as well as for realizing complex systems for large-scale projects.

Functions are represented in Pure Data by visual boxes called **objects** placed within a patching window called a **canvas**. Data flow between objects are achieved through visual connections. Each object performs a specific task, which can vary in complexity from very low-level mathematical operations to complicated audio or video functions such as reverberation, FFT transformations, or video decoding.

4 Prototype 1

This section contains the steps taken to develop the first prototype, and the steps include the research behind them and the implementation and explanation of how they work.

When I started the project, my first goal was to gain knowledge of Processing because computer vision is what collects the data from the camera and then transfers that data to Pure Data for audio processing and sound generation. Therefore, I decided to concentrate on Processing first.

4.1 Image Processing

4.1.1 Research

My first goal with Processing was to learn how to load an image and display it to the screen. This was an easy task, but I found that the size of the window in

the setup method of the processing had to be the same as the resolution of the image otherwise, it stretches or crops the image to fit the window.

For the second step, I wanted to take an image, load it in processing and use image processing techniques to detect and print only one colour from that image onto the screen. For this, I had to discover a way to transverse through each pixel of the image and differentiate between different colours and select and display only the colour I was targeting.

After watching some videos and going through Processing's tutorial [8], I found that Processing stores each pixel as a number in a one-dimensional array from 0 to height*width of the image. So, now the task was to find out how I can traverse through the picture as a two-dimensional array of pixel values and find the one-dimensional index value of each pixel. I found out that the one-dimensional index of any pixel at X, Y point in the window, can be calculated using a simple formula: **X + (Y*WIDTH).** This converts the two-dimensional index to one-dimensional index of the stored pixel.

FIGURE 7. SHOWS HOW THE PIXELS ARE DISPLAYED ON SCREEN AND HOW THEY ARE STORED IN PROCESSING.
https://processing.org/tutorials/pixels/imgs/pixelarray.jpg

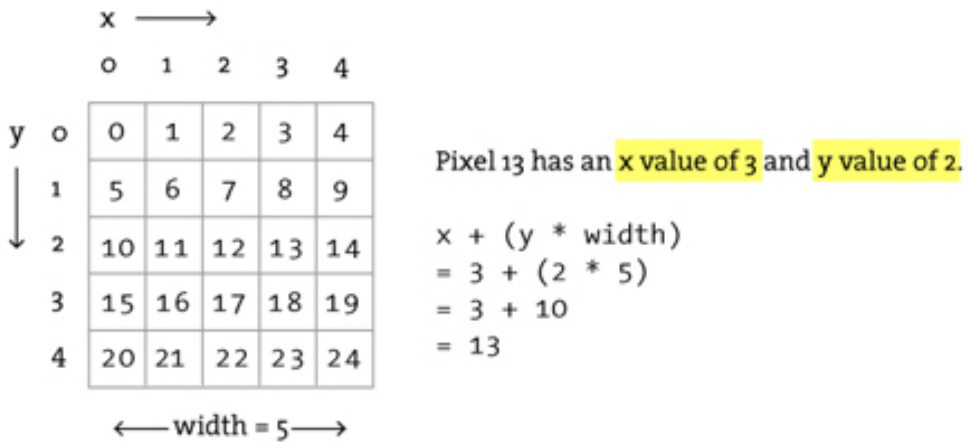


FIGURE 8. SHOWS CONVERSION FROM TWO-DIMENSIONAL INDEX TO ONE DIMENSIONAL INDEX FOR THE STORED PIXEL
https://processing.org/tutorials/pixels/imgs/pixelarray2d.jpg

4.1.2 Implementation

Processing has a distance function to find the distance between two different colours. I loop through all the pixels of the image and use this function to check the distance between the colour I am looking for and the colour of the current pixel. After calculating the distance, a threshold value is used to check how similar the two colours are and if they should be considered as same. As an example, if I am looking for RGB value (255, 0, 0) and the pixel currently selected from the image has RGB value (245, 10, 15), then the distance between the two colours would be 35. Since the current pixel is a lot like the colour I am looking for; I can increase the threshold value to 35. This would consider the colour of the current pixel the same as the colour I am looking for.

To try this, I took an image comprising of various colours and tried to print only red colour and replace the other colours with black. This was one of the milestones for me as this became the basis of processing the colours in each section.

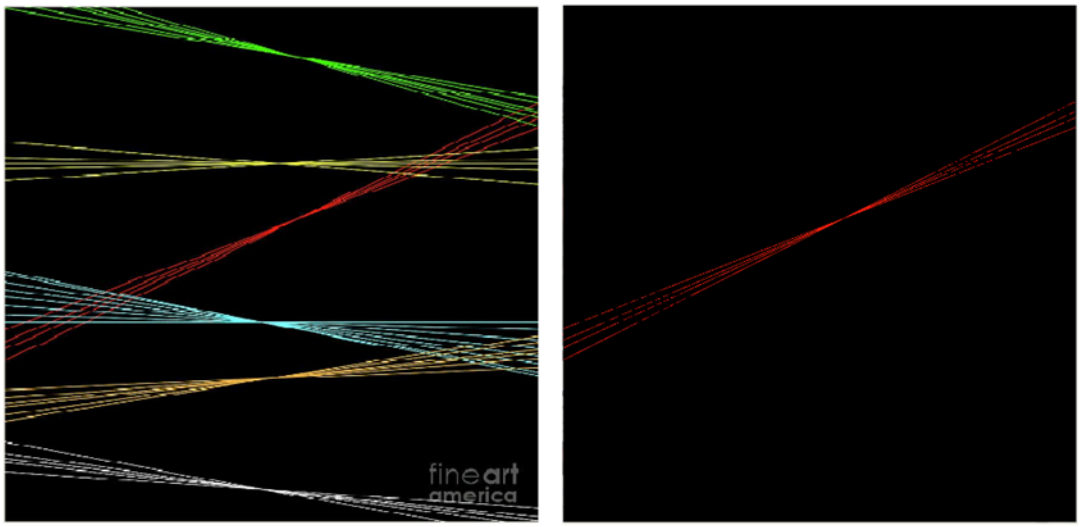


FIGURE 9. SHOWS THE IMAGE BEFORE AND AFTER PROCESSING AND DISPLAYING ONLY THE RED COLOUR
https://images.fineartamerica.com/images/artworkimages/mediumlarge/1/black-background-with-lazer-lines-bigalbaloo-stock.jpg

4.2 Video Processing

The first obvious task was to capture the video from the webcam and display it to the screen. This was an easy task. For this, I used the "**video"** library in processing.

4.2.1 Research

Like the image processing done before, the goal of this was to capture live video and process it to detect and show only one colour. The algorithm to do this is the same as the image, but there is a slight difference that a video is made up of a collection of images captured at a certain FPS. Therefore, the program had to process all those images as they were coming and then display them.

I noticed that the higher the resolution, the more lag there was in video processing. This was because the program had to go through each pixel and check if it should be selected or not.

4.2.2 Implementation

Since the tabletop of the installation will be divided into three different sections and each section would be representing a distinct sound and a different coloured putty, I divided the captured frames from the camera into three different sections of the same ratio and each section processing a different

colour. For this step, I had to come up with a solution to divide the areas and process different colour among each one of them.

Although doing this was not very challenging, I discovered some problems with it. Since in a live video, the light keeps changing and colours look different in different lights, it was hard to hardcode specific three colours that I want to look for in the video. Sometimes the colour would be there but have a different shade, and my processing code won't detect it. This problem required me to brainstorm some ideas and think about what a good solution to this would be.

After having a meeting with my supervisor, it was decided that having a slider for the threshold value would be useful as it can be adjusted to detect how similar of the colour would be considered as the same colour and also my supervisor gave me the advice to allow picking of three colours at the start of the program from the camera. For, since lighting would change the way the putty colour looks, therefore, it is a great idea to select the colours for each section in the current lighting and then start the processing. The implementation of the same was done. Clicking once in each region would set the colour for that section, and after all three have been selected, the threshold can be controlled using the slider. Once colours are calibrated, the window will only display the processed video.

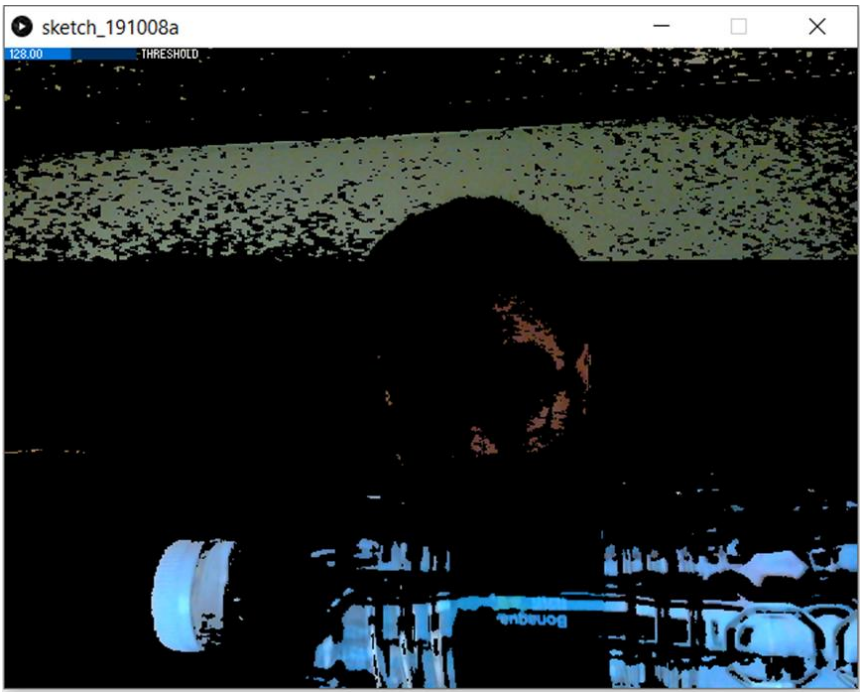


FIGURE 10. SHOWS THE THREE SECTIONS LOOKING FOR THREE DIFFERENT COLOURS AND A THRESHOLD SLIDER.

4.3 GUI Creation

Onto the next step was the creation of GUI through which the user will control the Gooizer.

4.3.1 Research

For this, I first tried to use the library **controlP5** [9]. This library was a lot more complicated than I expected and didn't have enough functionality available that was required for the project. After consulting with my supervisor, he advised me to use the **G4P** (GUI for Processing) [10].

G4P library provides an extensive collection of 2D GUI controls that were perfect for the project, and it can be used both by novice and experienced programmers as it comes with a builder tool. The GUI builder tool provides a visual environment for the rapid creation and editing of user interfaces using GUI controls.

The images (Fig.11 and Fig.12) show how the tool works. Adding a button in the GUI builder tool autogenerates the code for it in a new file named "gui".

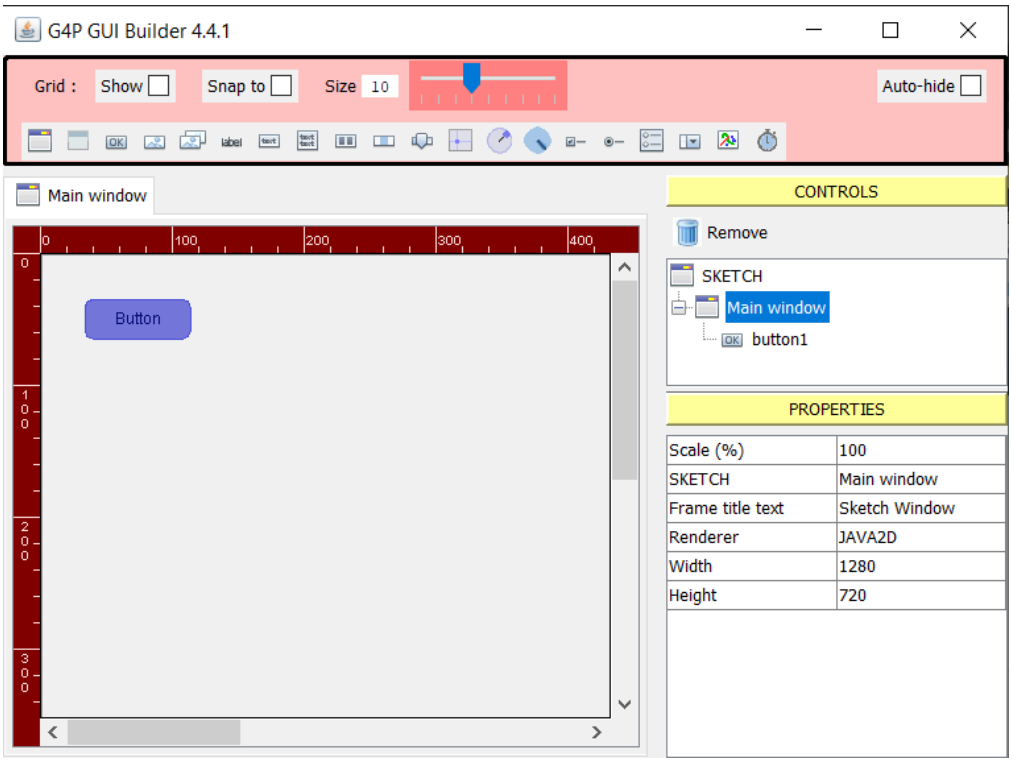


FIGURE 11. G4P GUI BUILDER TOOL

```
public void button1_click1(GButton source, GEvent event) { //_CODE_:button1:382733:
  println("button1 - GButton >> GEvent." + event + " @ " + millis());
} //_CODE_:button1:382733:



// Create all the GUI controls.
// autogenerated do not edit
public void createGUI(){
  G4P.messagesEnabled(false);
  G4P.setGlobalColorScheme(GCScheme.BLUE_SCHEME);
  G4P.setMouseOverEnabled(false);
  surface.setTitle("Sketch Window");
  button1 = new GButton(this, 33, 34, 80, 30);
  button1.setText("Button");
  button1.addEventHandler(this, "button1_click1");
}

// Variable declarations
// autogenerated do not edit
GButton button1;
```

FIGURE 12. AUTOMATIC CODE GENERATED BY THE TOOL

4.3.2 Implementation

I created a new window called controls, from where the user will control the Gooizer. I implemented the buttons that I need for the first prototype and were necessary for the user (Fig.13). The functionality of each button is as follows:

- **Slider:** It is used to change the threshold value during calibration. The threshold value has been allowed a range from 0 to 255, although one would probably never need 255.
- **Calibrate colour buttons:** Each button represents one section. Firstly, the user needs to click on the desired button and then the colour in the respective section. This would calibrate the colour for that button.
- **Finalize Calibration:** After all, three colours have been calibrated, clicking the finalize calibration will save those colours and start showing the processed video.
- **Scan:** To be pressed after calibration had been finalized. Used to scan the table. Needs to be pressed every time want to do a new scan.
- **Stop:** Used to stop the sound.

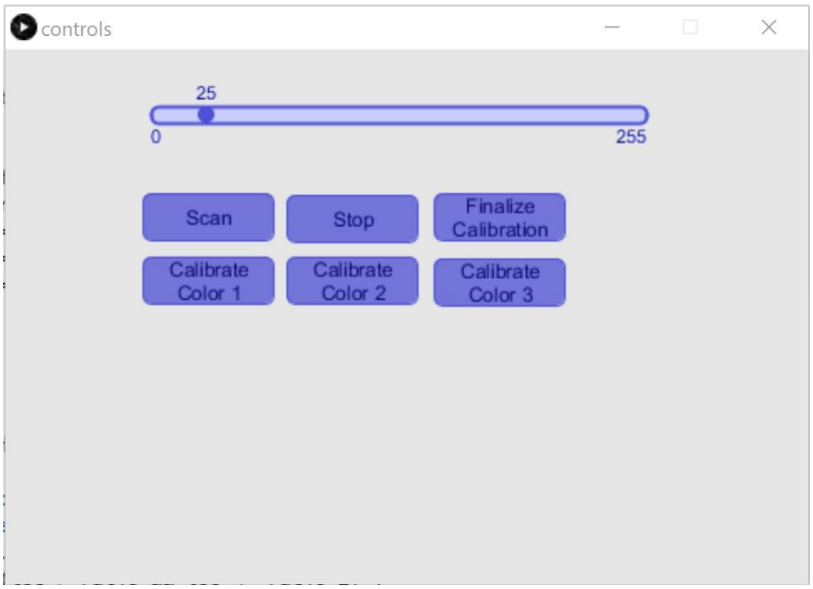


FIGURE 13. CONTROL WINDOW FOR GOOIZER

4.4 Amplitude Change

After I had my base of computer vision done, I went on to learn Pure Data, a lot of research, trial, and error was required in order to achieve what was desired.

Amplitude change was decided as one of the functions that Gooizer will support. It will change the amplitude depending on the position of the putty in each section.

4.4.1 Research

Amplitude modulation is a process by which the wave signal is transmitted by modulating the amplitude of the signal. Since I have no musical background, the words were quite confusing to me, and I didn't exactly know what I should search for, but after extensive searching, I found that what I needed to create was an Amplifier [11]. An amplifier changes the gain of the signal.

In the example (Fig.14), I will show what I found from my research. To make an amplifier, the sound file is first opened and read using the "**readsf~**" object. The **bang** is used to continuously play the sound once it stops.

The audio signal being read from the file can be changed using the slider. The limit of the slider is from 0-1, where zero means no sound, and one means full volume. The value is multiplied with the signal coming from the sound file, and this in turn increases or decreases the volume.

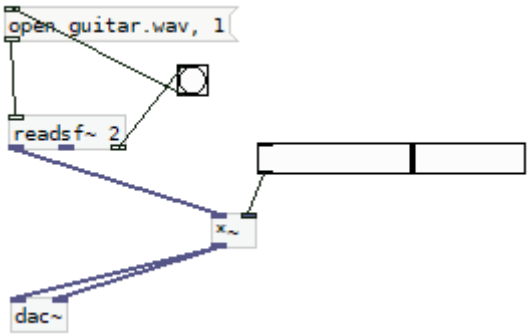


FIGURE 14. EXAMPLE OF AMPLIFIER

4.4.2 Implementation

For the implementation, I used the way I found from the research and did the same for three different sound files; each belongs to a separate section. And in place of the slider, the change in volume would be done by reading the position of the putty from arrays.

4.5 Communication between Pure Data and Processing

4.5.1 Research

For this, I had to research ways to communicate and send data between the two, and I came across Open sound control protocol (OSC) [12] [13]. It is a protocol for communication among computers, sound synthesizers, and other multimedia devices that is optimized for modern networking technology. Both Processing and Pure Data support communication using OSC.

OSC library in Processing is called **oscP5**. It can both listen and send messages. Using Processing, I just needed to send data to pure data. So, the way sending messages works is by sending packets to a specified Net Address. The Net Address takes two parameters, an IP address and a port number. The packet is sent with an address pattern and data. Data can be whatever you wish to send. The data can then be routed to desired objects in pure data depending on their address pattern.

4.5.2 Implementation

In Processing, I created two methods (Fig,15) that take care of sending the data to a specified Net Address with IP 127.0.0.1 and port 12001. One method takes

care of sending the coordinates, and the other sends messages such as “play” and “stop”.

```
oscP5 = new OscP5(this,12001);
myRemoteLocation = new NetAddress("127.0.0.1", 12001);

void sendMessage(String message) {
  OscMessage myOscMessage = new OscMessage(message);
  oscP5.send(myOscMessage, myRemoteLocation);
}

void sendMessage(String colorType, float x, float y){

  OscMessage myOscMessage;
  myOscMessage = new OscMessage(colorType);

  log("sending " + (int)x + " " + y);
  myOscMessage.add(y);
  myOscMessage.add((int)x);
  oscP5.send(myOscMessage, myRemoteLocation);
}
```

FIGURE 15. METHODS THAT SEND THE DATA TO SPECIFIC IP AND PORT

Pure Data listens for messages at port 12001 (Fig.16) waiting to receive packets with address pattern “/play”, “/stop”, “colour1”, “colour2”, “colour3” from processing. The patterns named as colours are sent with the x and y values, which are then routed to their respective objects and stored in arrays. The arrays have the same length as the width of the video because it needs to store the coordinates of the putty.

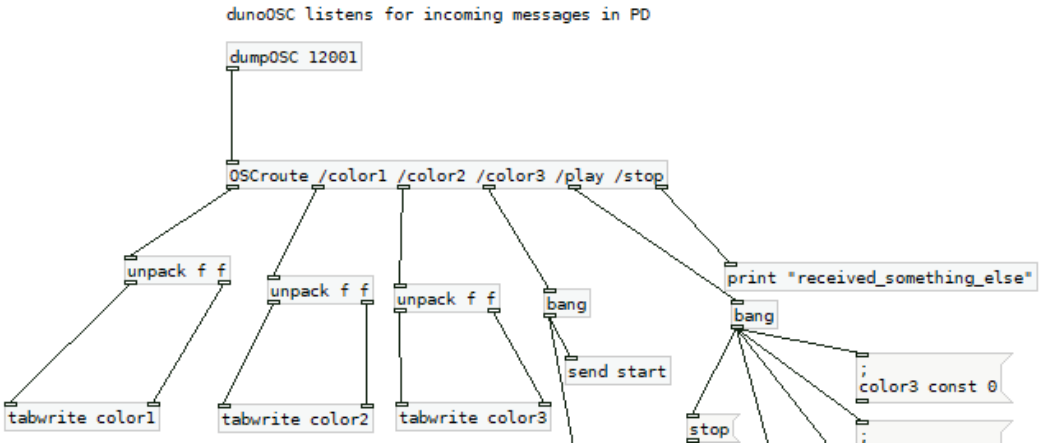


FIGURE 16. PURE DATA LISTENING AT PORT 12001 AND ROUTING MESSAGES AS PER THEIR ADDRESS PATTERN

If “/play” is received, the program starts reading the arrays and playing the sound, whereas getting the “/stop” stops the reading and sends 0 to the amplifiers to set the volume to 0.

4.6 Detecting and sending putty coordinates

As described earlier, the way Gooizer works is that Processing sends the data to Pure Data, and from that data, Pure Data produces sound.

The data here are the coordinates of the positions of the putty. I needed to find a way to obtain these coordinates and store and represent them in Pure Data.

4.6.1 Research

When detecting and processing colours in Processing, we know that many pixels are found in the same colours. These pixels are what represents shapes. But sending the coordinates of all the located pixels containing the calibrated colours would slow down the program and is not possible as the pixels on the same x coordinates would overwrite the values sent before it. So, after a discussion with my supervisor, it was decided the best way to do this would be to detect blobs and send the start and the end point of each blob and calculate and send the points connecting the two.

A **blob** is a region of an image in which some properties are constant or approximately constant; all the points in a blob can be considered in some sense to be like each other. For example, several connected pixels with the same colour can be called a blob.

The first thing required for this was to detect blobs for which I needed to find a blob detection library. Blob detection refers to modules that are aimed at finding blobs. It is usually done after colour detection and noise reduction to find the required object from the image finally.

I found a library named "**BlobDetection"** [14] on Processing's website. This library was useful if one just wanted to see the blobs and know how many exist, but for my needs, I required something that can give me more properties of the blob such as edges, centroid, etc. So, I had to find some other alternative, and came across a library called **"Blobscanner"** [15], which fulfills my requirements. It is a free software and registered under the general public license. The Blobscanner is far more feature packed then BlobDetection.

How it works is Blobscanner finds all the pixels in the image with a specified brightness value, assigning to those connected between them (connected

component) an equal ID and counting for each group of pixels with the same ID a new blob.

This library is perfect for my work as it allows me to get the information required that is the edges and the centroid of each blob.

4.6.2 Implementation

4.6.2.1 Finding and calculating coordinates

The implementation of code was rather complicated, and a lot of trial and error went into finding an algorithm that would work. I will use code to explain better how it works.

FIGURE 17. GOOIZER SCREEN AFTER FINALIZE CALIBRATION HAS BEEN PRESSED

```
/**
 * Detects the blobs from the processed image and for each blob sends to puredata the points on
 * the line connecting the most left, centroid and most right point of the blob.
 */
void processImageAndFindBlobs() {

  bs.imageFindBlobs(processedImage);
  bs.loadBlobsFeatures();
  bs.findCentroids();

  log("Number of blobs found " + bs.getBlobsNumber());

  float ratio = (video.height-1)/3;
  PVector[] edge;

  //stores the most left pixel of the blob
  PVector min;
  //stores the most right pixel of the blob
  PVector max;

  //process one blob at a time and send it's coordinates
  for(int i = 0; i < bs.getBlobsNumber(); i++) {
    edge = bs.getEdgePoints(i);
    min = new PVector(width,height);
    max = new PVector(0,0);

    for(int k = 0; k < edge.length; k++) {
      if(edge[k].x < min.x ) {
         min.x = edge[k].x;
         min.y = edge[k].y;
      } else if(edge[k].x > max.x) {
         max.x = edge[k].x;
         max.y = edge[k].y;
      }
    }

    //calulates the position of the blob using the centroid
    float centroidY = bs.getCentroidY(i);

    //find in which section the blob is located and send appropriate messages
    if(centroidY <= ratio) {
      log("Current Blob: " + i + " is in color1");

      min.y = ratio - min.y;
      max.y = ratio - max.y;
      float centreY = (ratio - centroidY) / ratio;
      float centreX = bs.getCentroidX(i);
      sendBlobCoordinates("/color1", min, max, centreX, centreY, ratio);
    } else if(centroidY > ratio && centroidY <= ratio*2) {
      log("Current Blob: " + i + " is in color2");

      min.y = ratio*2 - min.y;
      max.y = ratio*2 - max.y;
      float centreY = (ratio*2 - centroidY)/ratio;
      float centreX = bs.getCentroidX(i);

      sendBlobCoordinates("/color2", min, max, centreX, centreY, ratio);
    } else {
      log("Current Blob: " + i + " is in color3");

      min.y = ratio*3 - min.y;
      max.y = ratio*3 - max.y;
      float centreY = (ratio*3 - centroidY)/ratio;
      float centreX = bs.getCentroidX(i);

      sendBlobCoordinates("/color3", min, max, centreX, centreY, ratio);
    }

    //draw the contours of the blob
    bs.drawBlobContour(i,color(255,0,0),2);
  }
  //sends the play message after all blobs have been processed
  sendMessage("/play");
}
```

FIGURE 18. CODE FOR DETECTING BLOBS AND CALCULATING COORDINATES

After all the colours have been calibrated, and the user has finalized calibration, the Gooizer window will only display the calibrated colours. Now when the user presses the scan button again, this time after the image has been processed, the **processImageAndFindBlobs()** is called. **processImageAndFindBlobs()** is the method that takes care of finding the blobs and the coordinates.

"bs" is the Blobscanner object. First the function uses the **imageFindBlobs()** function of the Blobscanner library. Then **loadFeatures()**, this method is necessary to load all the edges and vertices in the Blobscanner object before retrieving it later. After that **findCentroids()** is called, it finds the centroids of all the blobs.

After that, the **ratio** is calculated; it is the height of each section. After, three **PVector** variables are created. The "**edge**" to store all the edges of a blob, "**min**" to hold the most left point of the blob, and "**max**" to keep the most right.

The for loop performs the algorithm for all the blobs that Blobscannner has found. **getEdgePoints()** is used to get all the edge points of the current blob and stores them in the "edge" array. The "min" is set to the width and height of the screen, and "max" is given the origin coordinates.

The for loop after that loops through all the edge points of the blob and the first if statement checks if it's less than the x coordinate of "**min**". if it is, then store the values of the edge in "**min**". Whereas if not, then if the edge x value is greater than the "**max**" value then stores it in the "**max**".

CentroidY stores the Y coordinate of the centroid of the blob. After that, the if statements check where does the blob exists. Is it in the first section, second or third? It uses the **ratio** and the **CentroidY** value of the blob to check that.

While implementing, I encountered a problem for which I had to come up with a solution. The origin (0,0) in a Processing window is on the top left, and all other coordinates are calculated in relation to that (Fig.19). Whereas, the origin for arrays in Pure Data is on the bottom left (Fig.20). Now, this is a little hard to explain here, but I will try my best.

What happens is, if I sent the coordinates like as per Processing's coordinate system, then the coordinates in pure data looked opposite (Fig.20) of what had

to be done. The higher coordinates became low, and the low coordinates became high. This resulted in the change of volume being the opposite of what the user desired.

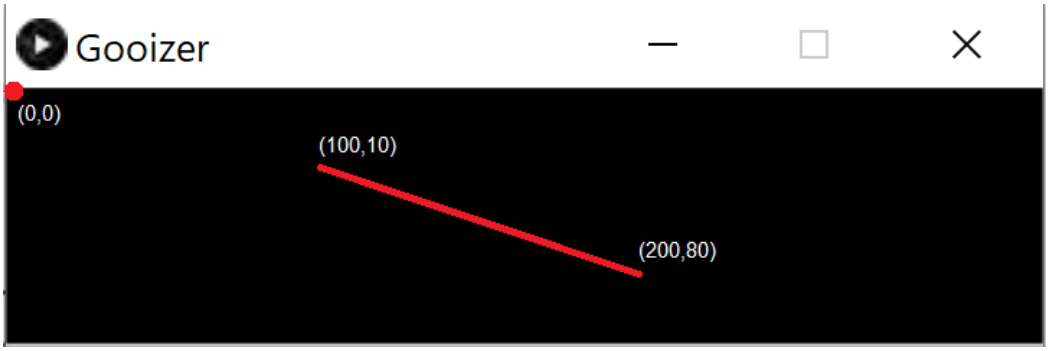


FIGURE 19. COORDINATE SYSTEM OF PROCESSING. SHOWING THE COORDINATES OF A LINE THAT NEEDS TO SEND TO PURE DATA.

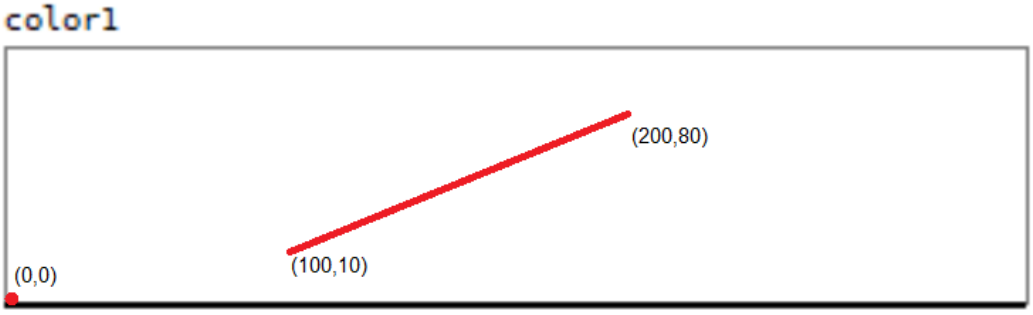


FIGURE 20. COORDINATE SYSTEM OF ARRAY IN PURE DATA. SHOWING HOW THE COORDINATES RECEIVED FROM PROCESSING WILL LOOK LIKE IN PURE DATA.

To fix the opposite coordinate system problem, the Y coordinate for all min, max, and centroid was subtracted from the ratio value for that section, which is the highest Y value of that section. For example, If the most left coordinate of the blob is (100,10) and the ratio of the first section is 100, then the updated coordinates will be (100,90) where Y = 100-10. And the same will be done for the centroid and the right most coordinate. Now the way the blobs coordinates are represented in Pure Data is the same (Fig.21) as the way it looks in Processing.

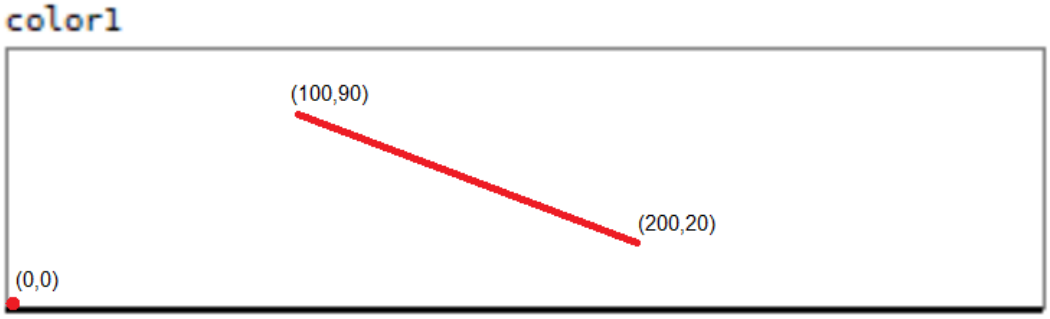


FIGURE 21. AFTER CALCULATION THE LINE IS REPRESENTED THE SAME WAY AS IT LOOKS IN PROCESSING.

Now since the value of Y is limited from 0-1, because that's the range needed to change the amplitude. The Y coordinates are divided by the ratio to get the value between 0 and 1, and then **sendBlobCoordinates()** is called, which takes care of sending the coordinates to Pure Data. Once coordinates for all the blobs are sent, a "/play" message is sent to tell Pure Data to start playing sound.

4.6.2.2 Sending Coordinates

sendBlobCoordiantes() takes care of sending the coordinates. Since only the most left, centroid and most right values were available. It was essential to find the coordinates between them. The goal was to find a line connecting these three points and obtain all the points on that line and send them to Pure Data. But I was unable to find a way to do it, so what was done is that the Y value of min was sent with all the X coordinates between min and centroid. And then, the Y value of centroid was sent with all the X coordinates between centroid and max. This resulted in a bizarre shape (Fig.22) that didn't look like the shape of the putty but still was perfect for testing purposes.

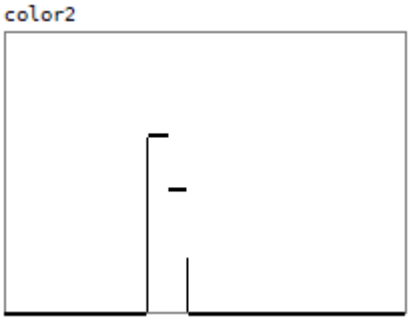


FIGURE 22. BLOBS REPRESENTATION IN PURE DATA

I planned to fix this later before the final product was done.

4.7 Reading data to Produce sound

Once the data has been sent to Pure Data, it needs to be read to play the sound.

4.7.1 Implementation

When Pure Data receives the play message from Processing, it starts a counter which increments by one after every 100ms. Once the counter reaches 320, it starts again from 0. The counter is used to read data from the arrays, which contains 320 indexes. Therefore, it takes 32 seconds to read the data of an array. The values read from the arrays are sent to their specific amplifiers. These amplifiers then increase or decrease the amplitude by multiplying the numbers received with the signal from the sound file.

This goes on until Pure Data receives a stop message. When the stop message is received, Pure Data stops playing the sounds and resets the arrays to 0.

4.8 Testing

Before doing user testing for the prototype, I conducted a series of intensive testing to make sure it was ready for user testing. The testing included independent functioning of the Pure Data and Processing and then the system as a whole. From these tests, I found a few minor bugs that needed to be fixed, and also some small improvements were made. These were:

Section lines on the Gooizer window: Since lines couldn't be drawn on the interacting surface, there was no way to know in which section one was placing the putty. Therefore, lines were added on the screen to show these sections (Fig.23). The user can now see the view from the camera when placing the putty.

Performance improvements: While testing, I noticed that the program was slow and lagging in some situations. Removal of unnecessary variables and optimizing the loops improved the performance.

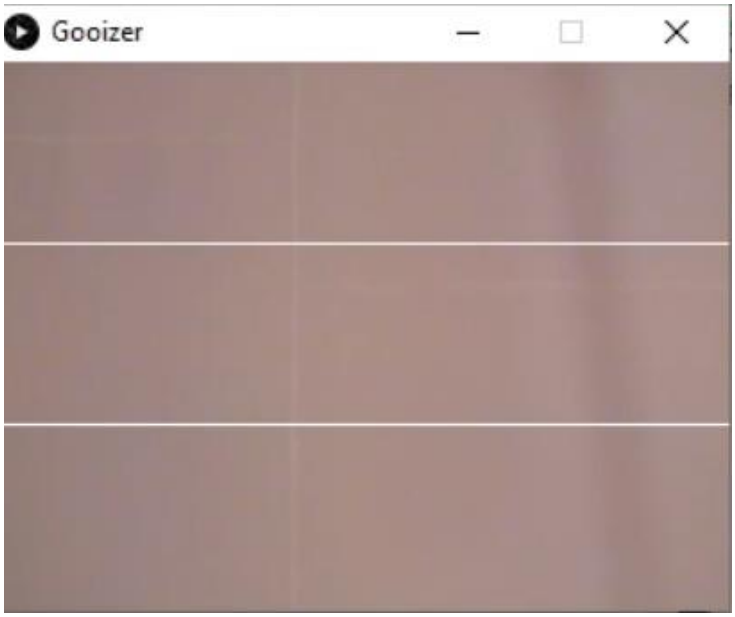


FIGURE 23. GOOIZER WITH SECTION LINES ALLOWING USER TO KNOW WHERE THEY ARE PLACING THE PUTTY.

4.8.1 User Testing

I was only able to test with one user and got some comments from my supervisor on the prototype.

The user was given several tasks to do:

1. Calibrate colours.
2. Put putty in shape to increase the amplitude.
3. Put putty in shape to decrease the amplitude.
4. Scan to produce sound.
5. Feedback

There were a few apparent improvements that Gooizer needed were found from the test and comments.

Possible improvements discovered from testing

- Shadows caused a problem if the threshold was too high, required more lighting.
- Add functionality to change other properties such as pitch and filter and let the user control it using a matrix.
- Add the functionality of automatic scanning after set intervals.
- Show a timer or slider so the user knows for how long he or she can change before rescanning occurs.

- Make replaying manually easy by allowing the user to scan and stop using keyboard keys,
- Add an extra window that always displays what the camera is viewing.

5 Prototype 2

This contains the changes made to develop prototype 2 from the user testing and personal additions.

5.1 Extra window displaying camera view

5.1.1 Problem

Gooizer had only two windows the control and the Gooizer window. Users can see and interact with the Gooizer by using the Gooizer window. This window allowed the user to see in which section they were placing the putty, but once the colours are calibrated, the Gooizer window only showed the processed image. There was no way for the user to know the position of where they were placing the putty once calibrated.

5.1.2 Solution and Implementation

To solve this problem, I created a window named “Live Feed” that will continuously display the view from the camera with the partition lines. This window will only start displaying once the finalize calibration button is pressed. Before that, the user can see from the Gooizer window as until then the user needs to only interact with it and would not confuse the user.

After adding this, the Gooizer had a total of 3 screens. These are the live feed window, control window, and Gooizer window.

5.2 Add functionality of changing the pitch

It was identified from the results of user testing that it was necessary to have additional functionalities and options to make Gooizer more exciting and that just amplitude control was not enough. So, I researched how to change the pitch of a sound file.

5.2.1 Research

The research into this took quite some time as the examples I usually came across were not using a sound file, but rather oscillators, and I had no music background, so I didn't know the logic behind pitch change.

After a few tutorials and examples, I discovered that pitch could be changed by increasing or decreasing the speed at which a sound is played. So, the pitch change I was targeting was done by increasing or decreasing the frequency at which the file played as pitch and frequency are directly proportional to each other.

I found a solution to what was required. The way it works is, first, the sound file is read into an array. Then the "**soundfiler**" object is used to get the number of samples from the file. A "**phasor**" wave is used to read the array and told how fast to read the data from the sound file.

Now considering that the soundwave had been sampled at 44100 Hz, then dividing it by the number of samples of the sound file will give me the rate at which it should be read by "phasor" to produce the default frequency of the file. This is then multiplied by the number of samples, so it's mapped onto the number of samples in the array and then read out to the "**dac~**" object to produce the sound.

For example (Fig.24), the sound file guitar.wav contains 109520. Dividing 44100 Hz with 109520 will give 0.402, which tells the phasor that it needs to munch 40.2% of the data from the array every second. The sound file will finish playing when the total is 100%. So, in this case, 2.5 seconds approximately, which is the duration of the sound file. Therefore, this will produce the sound at its standard frequency.

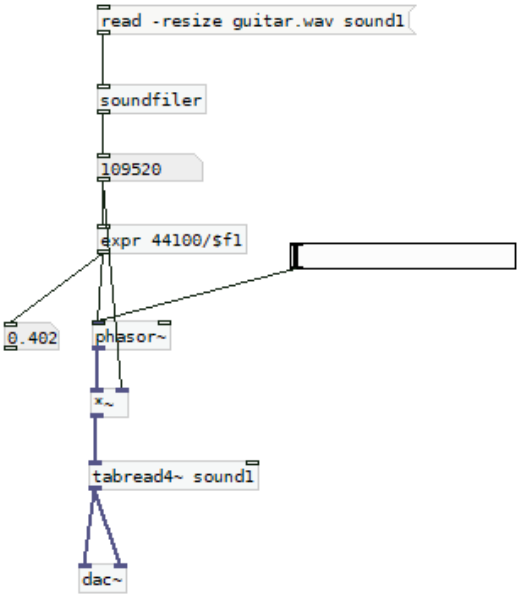


FIGURE 24. EXAMPLE SHOWCASING HOW PITCH CAN BE CONTROLLED

The rate at which the file is read can be changed by adding a slider with values from 0 to 1. This will tell the phasor to read the data at different rates; for example, if I send the value of 0.2, then the sound file will be read in 5 seconds, which will increase the duration and reduce the frequency and the pitch of the sound file.

5.2.2 Implementation

The implementation is divided into two parts.

5.2.2.1 Pure Data

For the implementation, I used the way I found from the research and did the same for three different sound files, each belonging to a separate section. And in place of the slider, the change in volume would be done by the numbers being read from the arrays.

Since the numbers being read from the arrays were in the range from 0 to 1, I decided to change the gain in the actual frequency of a sound file by -0.5 and 0.5. If the value read from the array is less than 0.5, then the frequency is decreased, whereas if it is greater than the frequency is increased, and 0.5 is considered as no change in frequency. The lower the value, the slower the frequency, and the higher the number, the faster the frequency.

For example,

Let, the rate at which the phasor should read to produce the actual frequency of sound file be 0.25.

Let X be the value received from the array, which is equal to maybe 0.4.

Now the updated rate at which the phasor will read the sound file will be

0.25 + (0.4 – 0.5) = .0.15

Which will result in reducing the frequency. Similarly, if X = 0.8 then

0.25 + (0.8 – 0.5) = 0.55

This will result in increasing the frequency.

5.2.2.2 Processing

Now the user can control both pitch and amplitude. Two buttons were added on the controls window that will allow the user to switch between the two whenever he/she wished. These buttons will send the message to Pure Data to tell what to change, and control and Pure Data will route the numbers accordingly to either the amplitude modulators or frequency modulators.

Once this was implemented, there was a need to tell the user somehow where the midpoint of the pitch was. For this, I added a line in the middle of each section. These lines will only show when the user selects pitch and will disappear if the user selects amplitude as they are not required for it. Anything below this line would decrease the pitch, and above would increase.

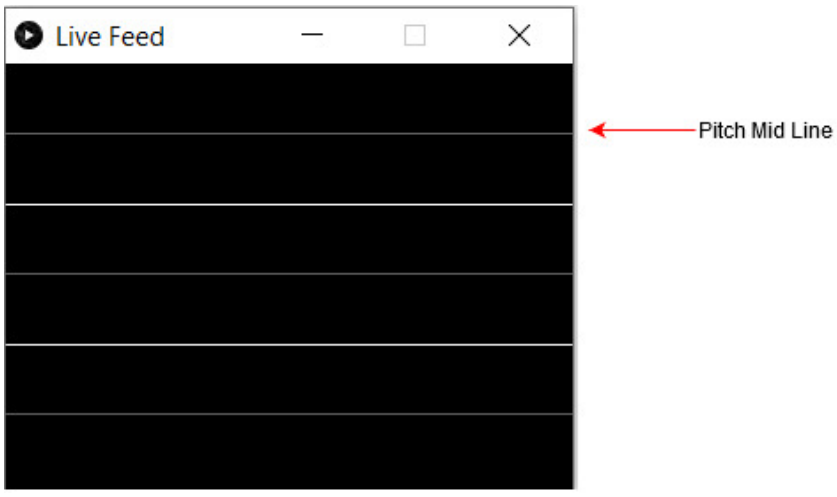


FIGURE 25. EXAMPLE SHOWCASING HOW PITCH CAN BE CONTROLLED

5.3 Automatic Scanning, Timer and Manual control keys

The scanning before was done manually only, but as suggested by one of the testers, it was evident that it would be for better if there was an automatic function.

5.3.1 Implementation

As mentioned before, the duration of a scan is 32 seconds. How the automatic function works is that it rescans after every 32 seconds. When the user clicks on the automatic function, it takes the first scan, processes the image, and sends the coordinates and “play” message to the Pure Data. After that it waits for 32 seconds before rescanning again.

A timer field has also been added to the control GUI. The 32-second timer starts as soon as “/play” message is sent. A rescan is done when the timer reaches 0. The timer was needed so the user knows for how long they can interact with the table before a rescan will happen.

Another functionality was added, which would allow the user to switch between automatic and manual as per their choice. Clicking on either first sends a ‘/stop’ message which empties the arrays and stops the sound and then prepares for rescanning. In automatic, the rescan happens itself whereas in the manual, the user must click on scan to do a new scan every time otherwise Pure Data will keep playing the last scan in a loop unless it receives a stop.

In Manual mode, scanning and stopping can also be done using the keys.
The key ‘S’ or ‘s’ would do a new scan, whereas ‘E’ or ‘e’ will stop playing the sound.

5.4 Updated GUI

The control window GUI had to be updated because a few more controls were added to the GUI. The GUI had to function in such a way so the user knows what to do next and not bombard the user with controls that cannot be used at a given time, and some previous steps needed to be done before using them. For this, I used some principles I learned in my HCI module and displayed only the buttons in order of their use, and unnecessary buttons were hidden when the user didn’t have any use for them. Fig.26 shows the flow of the GUI.

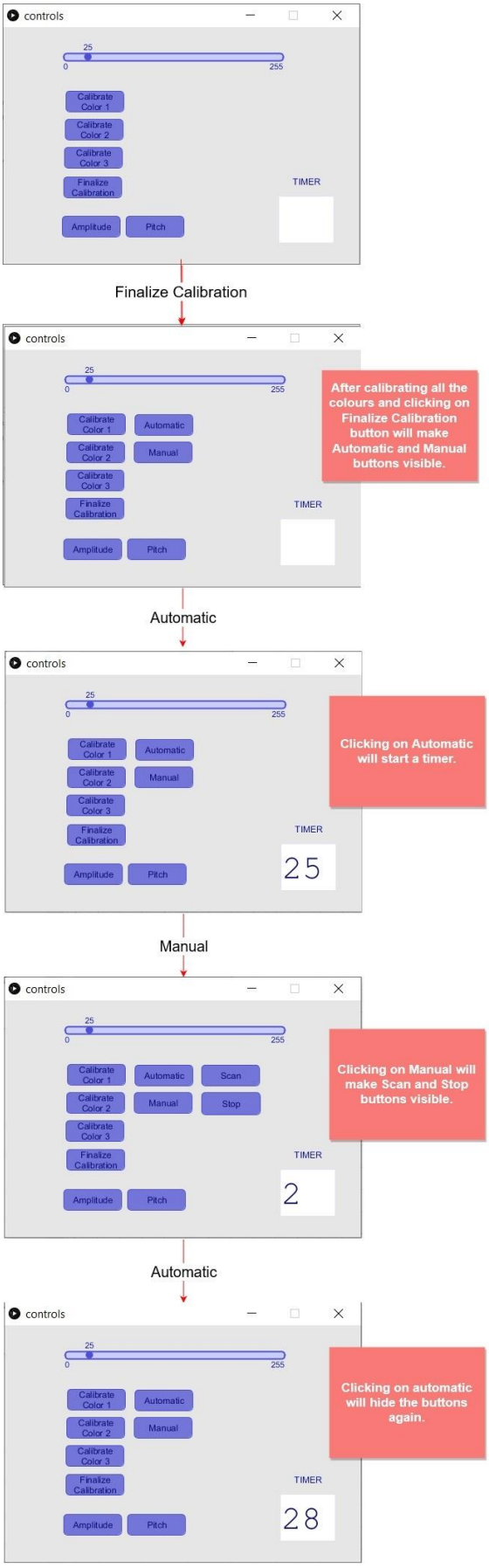


FIGURE 26. USERFLOW OF GUI

5.5 User Testing

Due to circumstances the second prototype was tested by just one user as well.

The user was given several tasks to do:

1. Calibrate colours.
2. Select amplitude.
3. Put putty in shape to increase the amplitude.
4. Put putty in shape to decrease the amplitude.
5. Scan Automatically.
6. Scan Manually.
7. Select pitch
8. Put putty in shape to increase the pitch.
9. Put putty in shape to decrease the pitch.
10. Scan Automatically.
11. Scan Manually.
12. The user was then allowed to interact with Gooizer freely.
13. Feedback

The test helped find two minor problems which would improve the user experience.

Possible improvements discovered from testing

- The slider doesn't have the "Threshold" name on it. So, the user didn't know from where to control the threshold.
- Add functionality so the user knows what part of the table is being played in real time.

6 Final Product

This section contains the major changes made to Prototype two to build the Final Product.

6.1 Scanning Line

6.1.1 Problem

As pointed by the user, there was no real-time information that would allow user to know what was being read and played. Sometimes, the user got confused about which sound was playing because all three sounds were playing at the same time.

6.1.2 Implementation

To fix the above-stated problem, I first tried to use a slider in Pure Data, which would show the increment from 0-320. Looking at which, the user would be able to know which part of the table is being read, but this violated my principle of having only one control area for the user rather than having them look at multiple programs. For an inexperienced user, Pure Data can be very daunting to look at. Therefore, I had to come up with a new solution.

The solution then I came up with was to have a scanning line on the live feed window that will allow the user to see what part is being read. The live feed window was chosen rather than Gooizer because after every scan, the Gooizer window draw function is stopped, so it doesn't scan anymore, and the draw function is necessary for the updating of the screen.

One the play message is sent, the line is drawn, and its x value is incremented till it reaches 319. The timer and the line both reset at the same time in automatic mode. Although I was unable to test the final product, but I am sure through this the user experience would improve a lot.

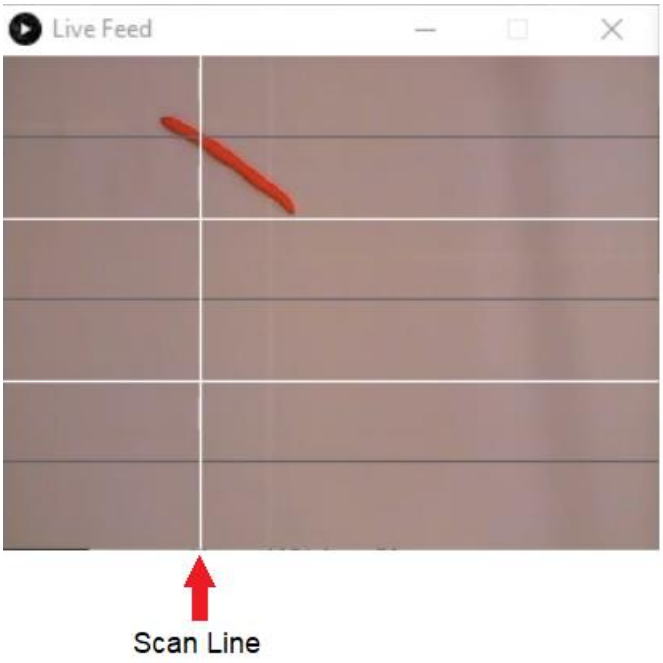


FIGURE 27. SCAN LINE ON THE LIVE FEED WINDOW

6.2 Fix Coordinates

As stated above that the points that were being sent didn't precisely represent the putty's shape. This needed to be fixed to produce better sound effects.

6.2.1 Research

For this, I had to find a way to get the points on the line connecting the min, centroid, and max of the blob. For this, I researched interpolation. Interpolation is the process of estimating unknown values that fall between known values. I found I can do it in two ways. One is that I can create a mathematical function that would do the interpolation, or I could just find a library or a method that can do it for me.

I would like to create a method for interpolation because it would help me gain more knowledge, but due to time constraints, I went with the second option.

I found that processing has a function named **lerp()**. This method takes three parameters **"start"**, **"stop"** and **"amt".** The function calculates a number between two numbers at a specific increment. The **amt** parameter is the amount to interpolate between the two values where 0.0 equal to the first point, 0.1 is very near the first point, 0.5 is half-way in between, etc. This function is convenient for creating motion along a straight path and for drawing dotted lines.

For example, if my start value is 2 and my stop value is 4, setting amt to 0.5 will give me 3. Similarly, now if I give amt = 0.2 I will get 2.4.

6.2.2 Implementation

For this I needed to update my **sendcoordinates()** method. What the method now does is it sends the **"min"** point and then calls the **sendMissingPoints()** (Fig.28) and similarly after sending the **"centroid", sendMissingPoints()** is called again . This method takes care of interpolating the points between given two points. **sendMissingPoints()** takes 4 arguments the x and y value of the first point and similarly the x and y value of the second point. So, after sending min, the method is called with values of min and centroid as arguments.

```
/**
 * Calculates and sends the points on line connecting the given two points.
 *
 * @param message The message to send with the coordinates.
 * @param x1
 * @param y1
 * @param x2
 * @param y2
 */
void sendMissingPoints(String message, float x1, float y1, float x2, float y2) {
  float rate = 1/(x2-x1);
  float fixedRate = rate;
  println("Interpolating");
  for(int i = (int)x1+1; i < (int)x2; i++){
    sendMessage(message, i, lerp(y1, y2, rate));
    rate+= fixedRate;
  }
}
```

FIGURE 28. CODE FOR FINDING THE POINTS

How the method works is it first calculates the rate at which it should increment the points the points. For example, if r x1 value is 50 and x2 value is 75 then the rate would be 1 / (x2 -x1) which is equal to 0.04. This means that we need to increment 50 at a rate of 0.04 to reach 75 in 25 steps.

After that I use a for loop, to loop through all the points from x1 to x2 and interpolate their y values using the rate calculated before. This is because x and y values will increase at the same rate to reach the final position.

This method is used to interpolate points between both min and centroid, and centroid and max giving a much better representation of the blob in pure data and also better sound effects.

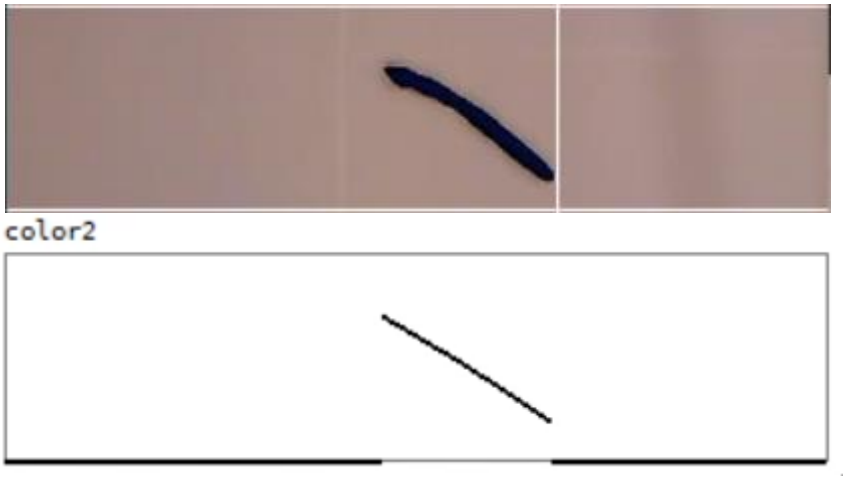


FIGURE 29. THE DATA IN ARRAY OF PURE DATA RESEMBLES MORE LIKE PUTTY SHAPE NOW

7 Conclusions and Further work

This chapter draws conclusions and makes recommendations for further work.

7.1 Conclusions

A tangible user interface was created with user experience at its core. The TUI scans and produces sounds depending on the interaction with the user. The Gooizer not only allows playing sounds but also allows the control of pitch and amplitude. This project achieved all the objectives except recording the sound.

The project lacks in the fact that there was not enough user testing done. The goal was to increase testing from prototype two but that couldn't happen because of the college closure.

This project has provided me with invaluable knowledge and experience in the many facets of software development as well as affording me the opportuning to learn about user experience and testing. I also gained knowledge of computer vision and audio processing, which would be a great addition to my arsenal of skills.

7.2 Recommendations for further work

While this project achieves what it targeted, but it can be extended to support changes to more sound properties and although, the user can still select the sounds they want to assign to each section using Pure Data, it is not very easy. Thus, the project can also be extended to giving the user an easy to interact interface that would allow them to choose and assign the sounds.

A person who has more knowledge about music and audio processing can add quite a few more features to this project which I can't possible think of.

8 References

[1] Shaer, Orit & Hornecker, Eva. (2009). Tangible User Interfaces: Past, Present, and Future Directions. Foundations and Trends in Human-Computer Interaction. 3. 1-137. 10.1561/1100000026.

[2] Nelson-Miller, 2016. What is tangible user interface. [Online] Available at: http://www.nelson-miller.com/what-is-a-tangible-user-interface/ [Accessed 30-12-2019].

[3] Reactable [Online], Available at: https://reactable.com [Accessed 01-01-2020]

[4] Levin, Golan. (2006). The Table is The Score: An Augmented-Reality Interface for Real-Time, Tangible, Spectrographic Performance. International Computer Music Conference, ICMC 2006.

[5] Flong, 2005. Scrapple Installation. [Online], Available at: http://www.flong.com/projects/scrapple/ [Accessed 01-01-2020]

[6] Welcome to Processing 3, Processing, Available at: https://processing.org/ [Accessed 24-04-2020]

[7] Pure Data, Available at: https://puredata.info/ [Accessed 24-04-2020]

[8] Images and Pixels, Processing.com, Available at: https://processing.org/tutorials/pixels/ [Accessed 23-10-2019]

[9] Andreas Schhlegel (2015), controlP5, Available at: http://www.sojamo.de/libraries/controlP5/ [Accessed 04-02-2020]

[10] G4P (GUI for processing), Quarks place, Available at: http://www.lagers.org.uk/g4p/ [Accessed 10-02-2020]

[11] The Amplifier, FLOSS Manuals, Available at: http://write.flossmanuals.net/pure-data/amplifier/ [Accessed 14-02-2020]

[12] Fraietta, Angelo. “Open Sound Control: Constraints and Limitations.” NIME (2008).

[13] Introduction to OSC, opensoundcontrol.org, Available at: http://opensoundcontrol.org/introduction-osc [Accessed 18-02-2020]

[14] Intro, Available at: http://www.v3ga.net/processing/BlobDetection/ [Accessed 22-02-2020]

[15] Antonio Molinaro (2016), Blobscanner, Available at https://sites.google.com/site/blobscanner/home [Accessed 24-02-2020]

9 Appendices

```
/**
 * This thread is created if the scanning is set to automatic.
 * Displays a countdown between scans and re-scans after the timer reaches 0.
 */
void timer(){
  int time = millis();
  int count = PLAY_TIME;
  while(runAutomaticTimer){
    if(millis() - time == 1000){
       count--;
       time = millis();
       timerField.setText("" + count);
       if(count == 0){
         runAutomaticTimer = false;
         sendMessage("/stop");
         redraw();
       }
    }
  }
}
```

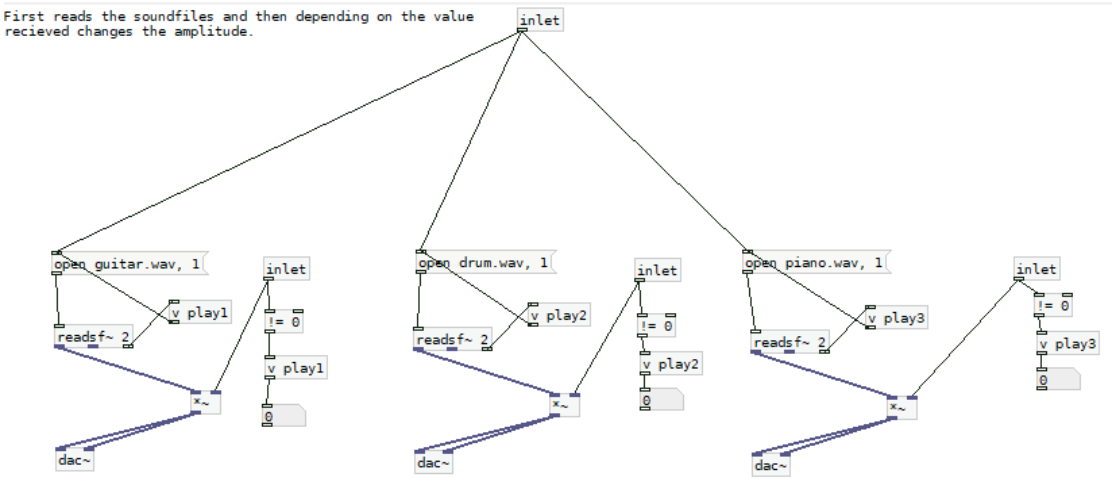
APPENDIX 1. CODE FOR THE TIME FUNCTION

```
/**
 * Send coordinates of the blobs.

 * @param message The message to send with the coordinates.
 * @param min Most left coordinate of the blob.
 * @param max Most right coordiante of  the blob.
 * @param centreX X coordinate of the centroid.
 * @param centreY Y coordinate of the centroid.
 * @ratio the ratio of sections.
 */
void sendBlobCoordinates(String message, PVector min, PVector max, float centreX, float centreY, float ratio){
  log(min.x + " " + min.y + " " + max.x + " " + max.y);
  sendMessage(message, min.x, min.y/ratio);
  sendMissingPoints(message, min.x, min.y/ratio, centreX, centreY);
  sendMessage(message, centreX, centreY);
  sendMissingPoints(message, centreX, centreY, max.x, max.y/ratio);
  sendMessage(message, max.x, max.y/ratio);
}
```

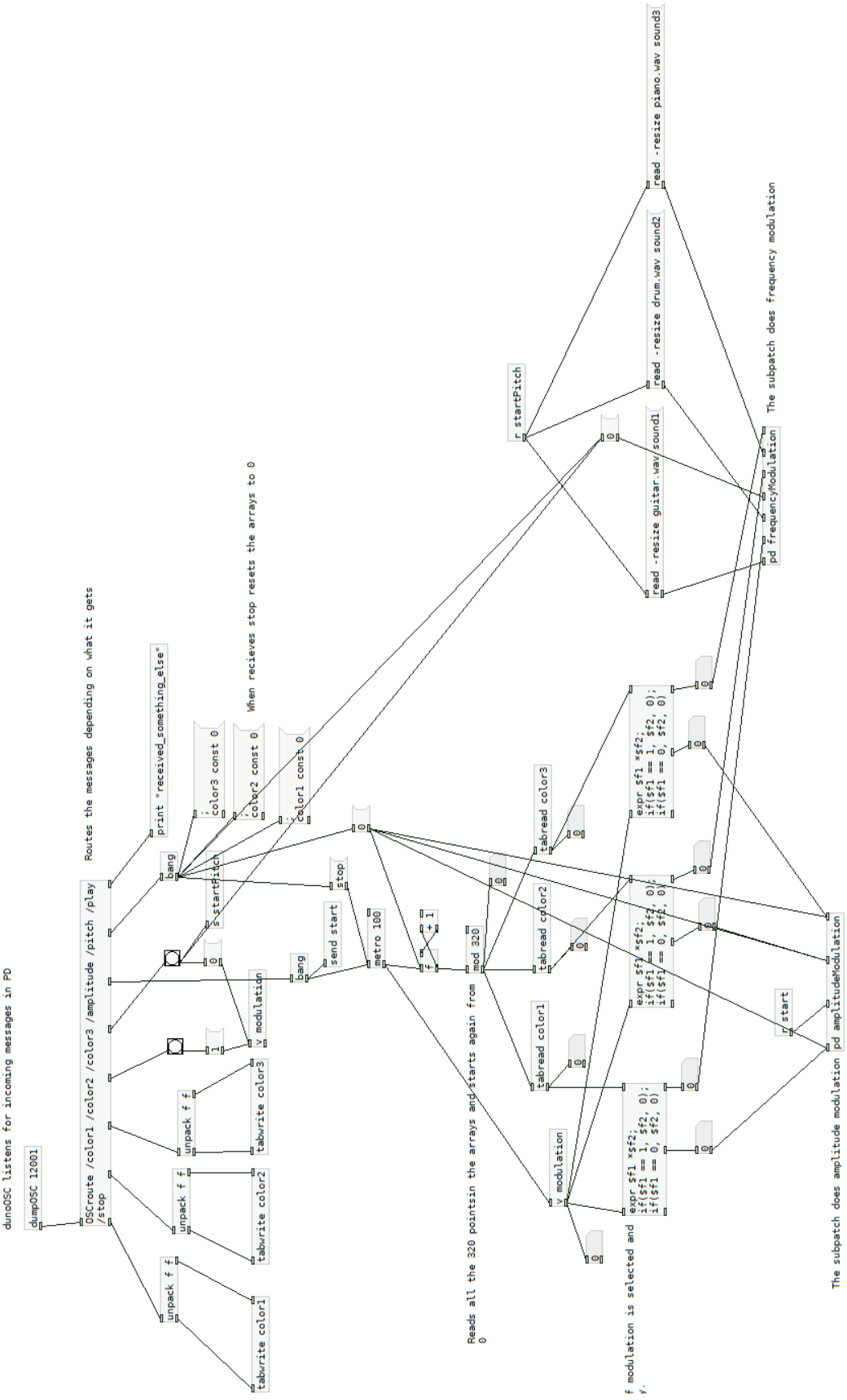
APPENDIX 2. CODE FOR SENDING THE BLOBS COORDINATES

```
synchronized public void live_feed_draw(PApplet appc, GWinData data) { //_CODE_:liveFeedWindow:222651:
  //Draws line on screen depending on which x value is being read at that time.
  if(calibrationComplete){
    appc.image(video,0,0);
    drawPartitionLines(appc);
    if(playing){
        if(indexNumber == (PLAY_TIME*10)-1){
          indexNumber = 0;
        }else{
          appc.stroke(255);
          appc.line(indexNumber, 0, indexNumber, video.height-1);
          if(appc.frameCount%12 ==0){
            indexNumber++;
          }
        }
    }
  }
} //_CODE_:liveFeedWindow:222651:
```

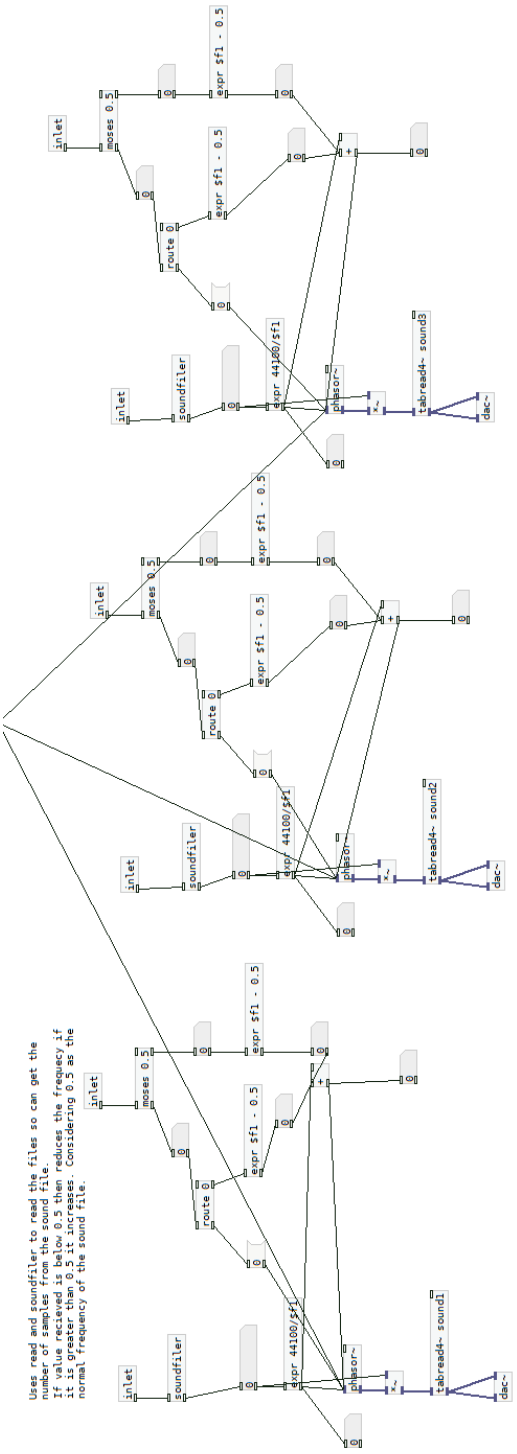
APPENDIX 3. CODE FOR DRAWING THE SCANNING LINE



APPENDIX 4. AMPLITUDE MODULATION



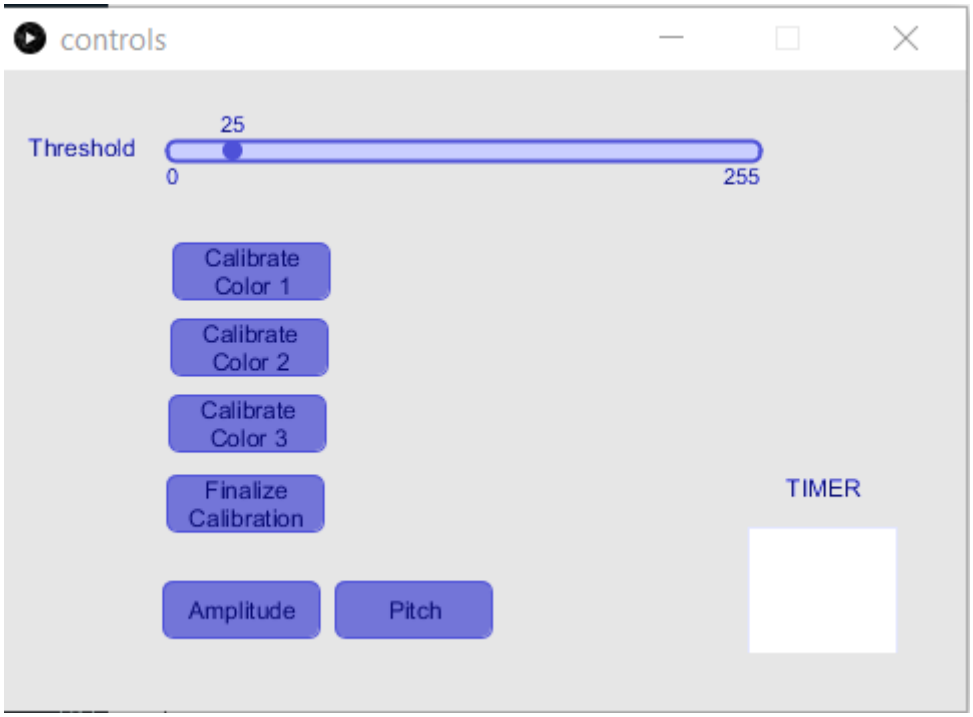
APPENDIX 5. PURE DATA PATCH



APPENDIX 6. FREQUENCY MODULATION TO CHANGE PITCH

```
/**
 * Processes the image captured and detectes the calibrated colors and outputs
 * the processed image.
 */
public void draw() {
  video.loadPixels();
  if(calibrated == 3) {

      image(video,0,0);
      loadPixels();
      processedImage.loadPixels();

      //maximum is the index of the last pixel
      int maximum = (video.width-1) + (video.height-1)*(video.width-1);

      //ratio is used to divide and process different sections for different colors
      int ratio = maximum/3;

      //processes each pixel of the image
      for(int x = 0; x < video.width; x++) {
        for(int y = 0; y < video.height; y++) {

        //loc is the index of the current pixel being processed
        int loc = x + y*video.width;

        //color of the current pixel
        color currentColor = video.pixels[loc];
        float r1 = red(currentColor);
        float g1 = green(currentColor);
        float b1 = blue(currentColor);

        float r2;
        float g2;
        float b2;

        //processing different colors for different sections
        if(loc <= ratio) {
          r2 = red(first);
          g2 = green(first);
          b2 = blue(first);
        } else if(loc > ratio && loc <= ratio*2) {
          r2 = red(second);
          g2 = green(second);
          b2 = blue(second);
        } else {
          r2 = red(third);
          g2 = green(third);
          b2 = blue(third);
        }

        //finds distance between the color of the current pixel and the color selected for that section
        float d = dist(r1,g1,b1,r2,g2,b2);

        //checks if the color is within the threshold value, if not then replaces with black color
        if(d < thresholdVal) {
          if(calibrationComplete) {
            //if calibration has been completed shows the processed areas as white
            processedImage.pixels[loc] = white;
          } else {
            processedImage.pixels[loc] = currentColor;
          }
        } else {
          processedImage.pixels[loc] = black;
        }
      }
     //displays the processedImage to the screen
     processedImage.updatePixels();
     image(processedImage,0,0);
   }
   //if calibration has been completed then processImageAndFindBlobs is called to process the blobs
   if(calibrationComplete) {
       log("Processing image to find blogs");
       processImageAndFindBlobs();
   }
 } else {
   image(video,0,0);
 }
 drawPartitionLines();
}
```

APPENDIX 7. CODE FOR PROCESSING IMAGE AND DETECTING COLOURS



APPENDIX 8. FINAL CONTROL WINDOW