Visual Access for Blind People

A Blind Envisioned Learning Tool System (B.E.L.T.S)

ELEC95012 EE2 Electronics Design Project

Electrical & Electronic Engineering, Imperial College London

Group 7 – Dr. Thomas Clarke

Issa Bqain	CID: 01503843
Umut Ekinci	CID: 01486809
Pavan Singh Gill	CID: 01481616
Arman Fidanoglu	CID: 01512561
Alp Atakav	CID: 01523868
Xia Chen Hao	CID: 01249695
Omar Inuwa	CID: 01187586
Lukas Baliunas	CID: 01352301

Submission Date: 27th March 2020 (GMT)

Contents Page — Group 7 Engineering Design Project

Abstract

Simple visual-based tasks, such as plotting and understanding graphs often prove difficult for the visually impaired, limiting their educational prospects and contributing to high unemployment rates [1] among the blind. On top of that, blind-related technology currently available for sale also does not address this issue directly, leaving an empty niche on the market.

Our project seeks to solve this by introducing a novel mechanical touch interface that allows them to directly feel and understand any possible line graph (e.g. mathematical functions, statistics, charts), when connected to a personal device. The input is fed into the mechanical interface via the text-to-speech front-end software and translated into a function, which is then plotted and displayed on the interface through linear displacements of physical dots.

The prototype development spanned over two months, with the hardware and software team actively working together to develop the structure, mechanism and circuits, along with the user interface and control system for the device. The first month saw plenty of experimentation with various mechanisms ranging from sliding brackets to conveyor belts, with the latter being eventually chosen due to its compactness. Development of microcontroller and front-end code also took place concurrently. The prototype was then built, tested and calibrated. Results show that the prototype is successful, being able to display various types of curves (sine, Gaussian, parabola) with adjustable step sizes and window which can be easily felt by touch.

1 **Introduction:** Background on the Visual Access System

1.1 Problem Statement

Visually impaired people often suffer from limited education and job prospects due to their inability to interpret non-verbal information that can only be transmitted visually, such as text and images.

Since young, the lack of tools to properly learn visual based subjects such as STEM and Economics in schools have led to a high dropout rate (up to 50%) [2] in the United States. This often progresses into high unemployment rates during adulthood, with most jobs from accountancy to economics and engineering all requiring the regular use of graphics (e.g. schematics, graphs, charts). As reported by the Royal National Institute of the Blind, only 27% [3] of visually impaired people of working age in the UK are in employment, with 39& expressing great difficulty in making ends meet.

Even in their everyday lives, basic needs like viewing stock charts and statistics to independently understand and navigate the world around them are often near impossible feats. As a result, they are often unable to fulfil their true potential in society and find it near impossible to make a living, or even survive independently. It was also estimated by the World Health Organization that there are 285 million visually impaired people worldwide, with 90% living in developing countries [4]. The number is also set to increase as the population ages, indicating a huge demand for an affordable solution.

1.2 Rationale

In view of all these issues, the project therefore aims to develop a long-term affordable solution that would help blind individuals bridge the gap caused by their visual disabilities in order to improve their education and employment prospects while enriching their daily lives. This can be done in many ways.

While Braille has existed for centuries and modern technology such as text to speech and refreshable Braille displays have made textual information much more accessible to the blind, they do not directly tackle the issue of images, with the closest attempt being to verbally describe every detail of the graphic. (e.g. image recognition apps like TapTapSee [5]).

This often leaves out several abstract characteristics which are essential for the proper understanding of the information, presented, especially in the context of graphs and charts which often conceal essential information (arbitrarily close points, local maxima/minima, change in gradients, points of inflexion) within minute graphical details.

This project therefore focuses on helping blind individuals interpret images, with particular emphasis placed on the graphs in order for our contributions to make a larger impact on the blind community.

1.3 Competitors

Even though Braille devices and image recognition apps are commonplace on the market, there are currently no products focusing on graphic visualization for the blind exists on the market, suggesting very limited competition. The products listed (e.g. Refreshable Braille Display, wearable GPS [6] [7]) often serve a completely different purpose and act more as complements to the solution we plan to develop. In addition, the price range of these products far exceeds what most blind adults and with modest income can afford, (2000 pounds and above), and what most blind organizations can mass purchase, meaning that most of its target audience who live in poverty have not been benefitting from them.

All these indicate huge developmental and marketing potential for an affordable product that could directly address both issues.

1.4 System Overview

The initial chosen idea was as follows:

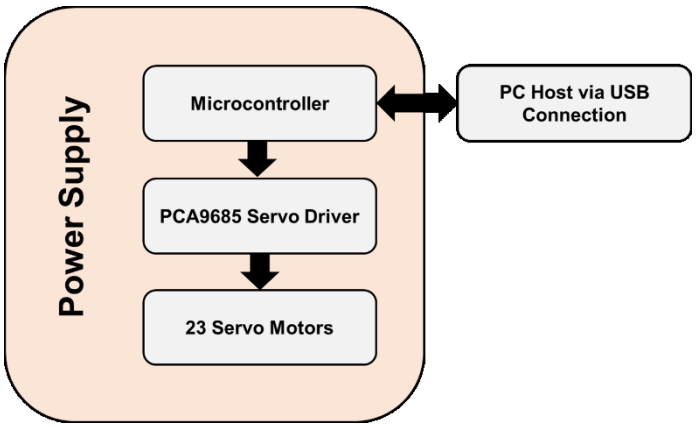


Figure 2 Proposed System Schematic

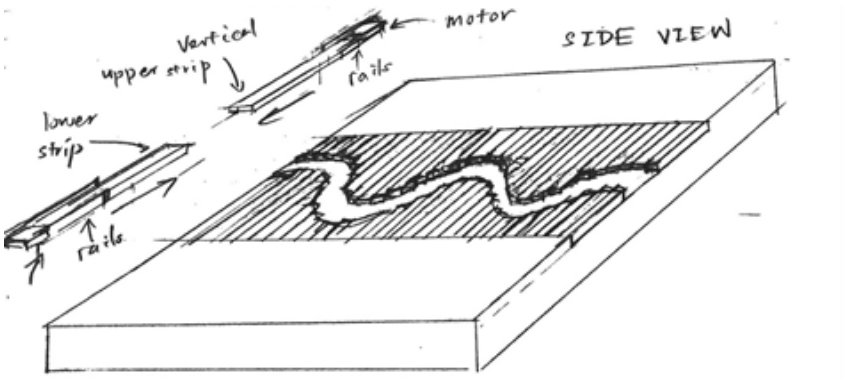


Figure 1 Concept Design from Preliminary Report (Outputting a Sine Wave)

How the System Works: When a user inputs a graph into the computer, it is translated into vertical displacement signals and mapped onto the display shown above (for a sine wave). It consists of two rows of vertical metallic strips that move up and down, with the gap between the two rows arranging to form a hollow cavity in the middle that can be sensed through touch.

Rows of linear actuators are driven by multiple motors mounted under the strips, and are controlled by drivers and the Arduino, which in turn receives commands from the user interface software. The development can therefore be split into hardware (motors) and software components, with the overall subsystem layout shown above.

2 Design Criteria: Design Specifications of the Project

The following product design specifications are prioritized when designing the device.

2.1 Performance

Resolution - how much information the display can represent without ambiguity

Accuracy – information should be represented with little error or distortion

Versatility- Display should be able to accommodate wide variety of graphs at different scales

2.2 Target Product Cost

The device needs to be cheap to reach most of the blind community

2.3 Ergonomics

The device should be designed specifically for blind people, considering all their needs and allowing them to use it with minimal assistance.

2.4 Size, Shape, Weight

For portability and regular use, the dimensions of the device should be around 250.6mm x 174.1mm x 7.5mm and weigh less than 1.18kg.

3 Hardware Concept Design and Critical Analysis

3.1 Performance (Hardware)

3.1.1 Mechanical Design:

The first major modification to the preliminary design shown above is the removal of the bottom half the display, and the modification from concave sensing to convex sensing. This saves the need for half of the motors, drastically reducing cost and space while improving ease of construction and maintenance with little compromise in resolution. The strips themselves will now be touched instead of the gaps between them, with the points on the graph embossed as protruding dots that can be easily felt by sliding the entire hand across the display.

3.1.2 Choice of Motors

Servo motors were chosen over DC motors, due to higher precision and built in H-bridge. More specifically, 360 servo motors (Feetech FS90R) are used, which allows the gears to rotate continuously at a steady rate. This in turn allows for unlimited displacement over time. At the same time, they are also relatively small (23.2 x 12.5 x 22mm) [8] compared to more sophisticated versions, which minimizes the space taken up by each linear mechanism and maximises resolution, while being relatively cheap at 5.04GBP [9].

Unlike most servo motors though, they are normally controlled by time delays instead of angle. For our application however it does not make a significant difference, and any slight misalignments can be corrected by calibrations.

The running current is stated as 120mA with a small load at our required voltage which is feasible within our power constraints. The Stall current is also given as 650mA [8] which will come in useful when using this as a sense to know when the device reaches the end of a strip. It is also lightweight at 10g which makes using a large amount lightweight which is pinnacle for a portable device.

3.1.3 Choice of Transport Mechanism

The task at hand was to develop a mechanism for the linear actuators that is effective, durable and easy to replicate. It should also be as small as possible for maximum resolution.

(1) Initial Rack and Pinon Transport Mechanism

Initially, the rack and pinion design were explored, and it was the closest design to the concept illustration. The first variation of the idea came from an open source model by YouTuber Potent Printable [10], shown in figure 3. The simple mechanism consisting of gear strips with gears placed vertically inside a motor bracket.

The proposed design was to have a set of the mechanism on the top and bottom separated by a small gap as shown. The strips themselves will then be mounted to and slide along with the gear tracks. The design however had noticeable flaws when used in the graphical application. The distance between each rack in series due to the protrusion of the motors created a sense of discontinuity in the graphs, and it is difficult to mount strips on the gear tracks themselves.

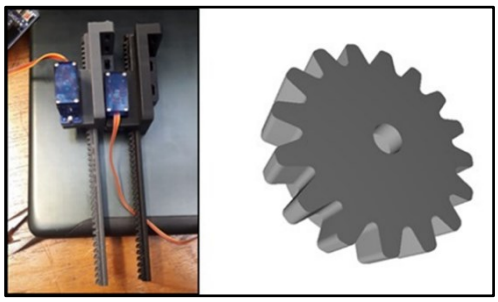


Figure 3 Initial Rack and Pinion Design with gear (Right)

(2) Improved Rack and Pinion

This led to the second variation of the mechanism. This time the servos would be placed under a new version of the rack, shown in Figure 4. which would allow for the racks to be closer together removing sense of discontinuity. The plot points would then be embossed on top of the bracket to enable.

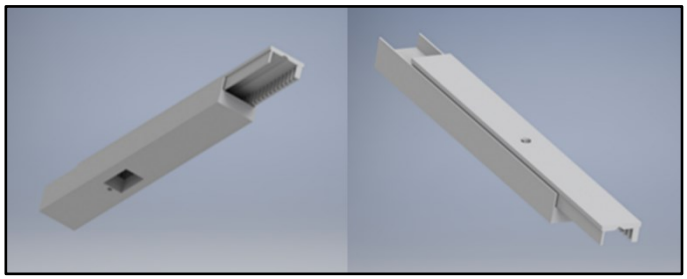


Figure 4 Rendered 3D files of the second variation of the Rack and Pinion mechanism

The slot cut out in figure 4 was designed to house the servo motor and the top piece was designed to have teeth at its inner side for the gear to bite onto it; moving it horizontally. The other rail without teeth holds it in place, preventing uncontrolled rotations and vibrations. The top piece design had a noticeable semicircle depression that ran through its length, which was designed to accommodate the height of the screw head, and also allowed for the screw head to further act as a guide for the rack to slide. Initial testing suggested that the mechanism works, but the design leads to inefficient use of space when replicated and scaled, since the bracket extends to twice the length of the display, which could easily exceed the size requirement.

On top of that, the resolution is still limited by the width of the servo motors, as the brackets needs to be wider than the motors themselves. This leads to our final design.

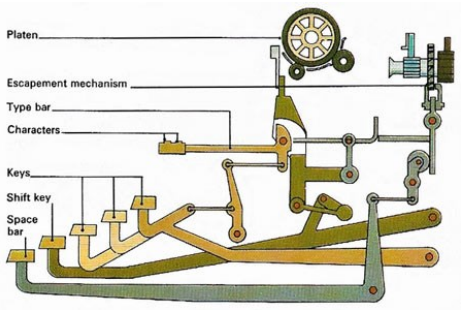


Figure 5 Picture of typewriter mechanism *[21]*

The final variation of the design was influenced by the old typewriters' designs shown in Figure 5. The idea was to use make use of strips that would connect to a rack in the first variation at different angles. This would improve the resolution at a system level due to its ability to eliminate gaps between each strip, with the width of the motor no longer limiting its resolution.

However, it similarly suffers from the same problem of inefficient use of space due to need for extensions. On top of that, due to wide variety of precise, oddly shaped mechanical components required for each motor, it is extremely difficult to build and maintain, and would likely lead to multiple misalignment errors.

(3) GT2 Timing Belts (Modular Strip Design)

The belts idea was explored as a better alternative to strip representations. It was inspired by conveyor belts arranged in columns as shown in Figure 6. Each belt would have a dot, which represented a coordinate point and together form a graph. This has the distinct advantage of being extremely compact and modular, with little additional space used beyond the display itself. The idea was changed to having the belts on its side with a clip attached to it to represent physical points instead of a strip as shown in the Figure 7.

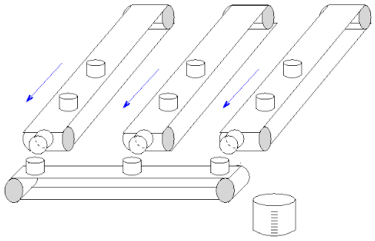


Figure 6 Inspiration from Conveyor Belts *[22]*

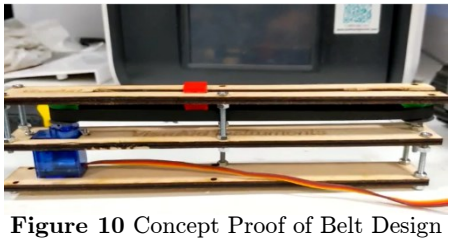


Figure 10 Concept Proof of Belt Design

Figure 8 3D Printer Belt (GT2) by Blu Siber [23]

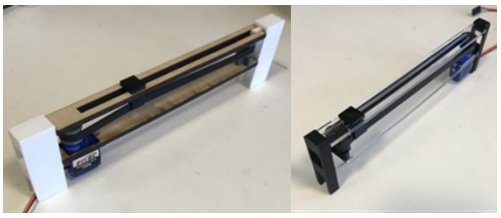


Figure 9 (Left) Initial Concept Design in wood. (Right) Refined Strip in Plastic

The inspiration for the design of the gears were taken from the existing 3D printers gears which control the movement of durable belts to change the coordinates of the nozzle, shown in Figure 8.

Figure 7 Custom Designed 'T' Design

The gear was design to mimic that of a 3D printer gear and it was also designed to be printed with no supports, hence it was split into two parts the gear and the lid with an inter-locking mechanism shown in figure 9.

Both the lid and the gear has different diameter openings which allows it to be dual purposed. The smaller diameter allows for it to be screwed into the servo and the larger opening allows it to freely rotate on the rod. The rod is design to have four screw points a T shaped base for support to withstand the tension exerted by the tighten belt.

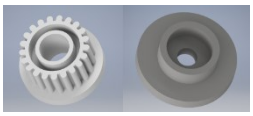


Figure 11 Custom Designed gear and Pulley in two pieces

The rod additionally has a fillet transition between the protruding surface and the T shaped base, to add some rigidity to the design as seen in figure 10.

The Prototype was further refined to have supports and was made into a modular design shown in Figure 11. Each modular piece would be placed slide by slide to form a chain of belt systems shown in Figure 12.

The row of modules were placed into a case made of wood to keep them in place as was used as the initial debug unit to develop software shown in figure 13.

Figure 13 Initial Debug Unit in Wood case

Figure 12 10 quantities of refined plastic design

(4) GT2 Timing Belts (Single Middle Plate Design)

After a successful test with the modular system the team decided to move away from a modular design, as it introduces too many variations on the surface and may cause confusion to blind users and instead adopted a three-tier layer, simpler design shown in the final design in 15 and an early product design in Figure 14. We also increased from 10 strips to 23 as this was required to have a higher resolution and was evident from the testing we did from the 10 modular strips we built (seen in Figure 13)

The middle plate of the device house all the circuits and mechanism that allowed the device to function shown in Figure 16.

The middle plate was also design with extended slides to prevent any caving in on the sides of the device.

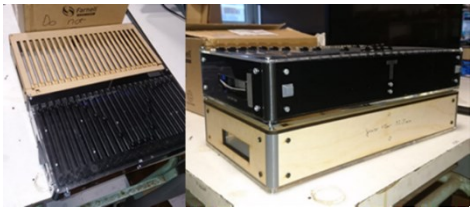


Figure 16 Final Refined Simpler Design (Black)

Figure 14 Early Wooden Prototype Design

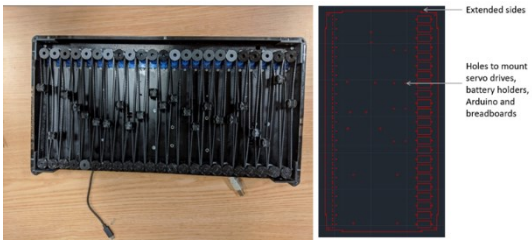


Figure 15 (Left) Middle plate exposed, (Right) AutoCAD File

3.1.4 General Internal Electronic Design:

The internal system needs to withstand a large number of components with different current and switching demands. The device also needed to meet the standard of being portable and not restricted to an AC adapter.

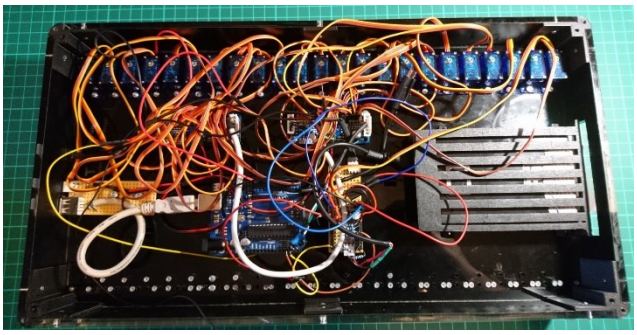


Figure 17 Back Panel removed to expose electronics

For this reason, a battery pack was utilised in order to supply the motors and control circuitry with power. In order to implement easy serviceability, the battery pack was made removable with a slide-in battery slot on the side of the device (see appendices for more images).

The battery then connected to a Power distribution board that separated out power lines for all the different components in the system. This included the Adafruit Motor controller and all the servos. The USB connection and power to the Arduino however were not shared with the battery supply as this caused a ground loop issue from the PC supply. The PC ground potential had to be included in the USB connection as the data lines are with reference to the ground of the PC and hence is required.

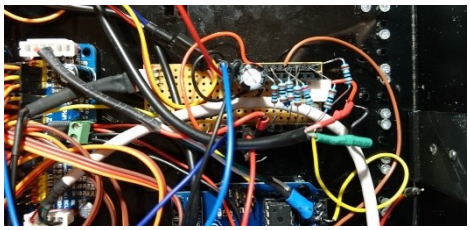


Figure 18 Internal Power Distribution Board (including sense resistors)

A USB and charging port input were made available to the exterior of the casing. This enabled a standard USB cable to be used to connect to a computer. This cable included all the Power (+5V and GND) and data lines needed from the PC. This port acted as the interface to communicate with the device.

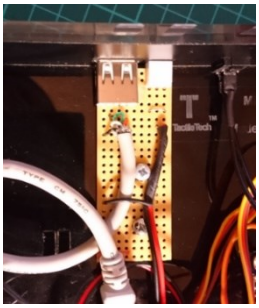


Figure 19 USB and Charging Ports

A disadvantage of the system however is that the current supplied to the system is limited. We are limited to what the Power Supply (in this case the battery) can supply which is approximately 2A. Therefore, running the motors all at once would consume up to 4A which would not be feasible at this stage and add unnecessary complications to the software. In the end, a decision was made to move only one servo at a time in order to meet this power constraint. If the project were to be done again, a better high-performance battery system would be installed to speed up the device updating the diagram displayed.

3.1.5 Sensing when the elements reached the end of the strip:

Since the motors gear and pulley were made out of plastic, it was not advisable to keep running the motors when they reached the end of the strip as this was where most of the stress would be inhibited by the mechanism and therefore increase possible wear and tear.

To rectify this problem, there needed to be a way to sense when the elements reach the end of the strips. At first, an idea of using contact switches was advised which would make sure all 23 elements had contact switches implemented at the end of each strip. This, however, was unnecessarily complicated as it would need a way of communicating a 23-bit value. Another solution of utilising a weighted adder and converting these logic levels to an analogue waveform was abandoned due to the sheer complexity required.

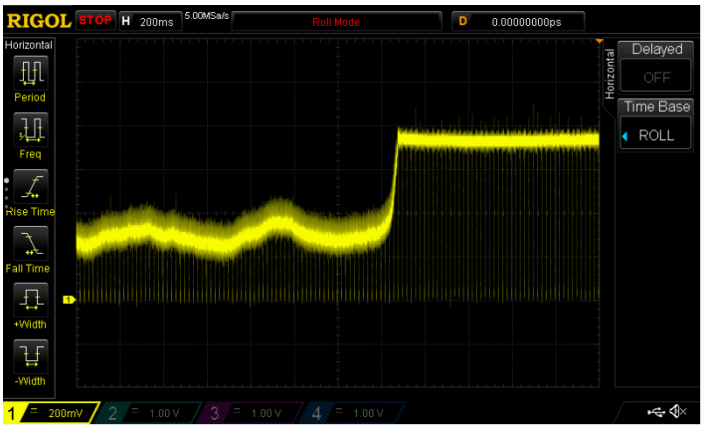


Figure 20 Waveform just after an element hits the end of the strip.

The solution we settled on was inserting a resistor in series with the positive 5V rail of all the servo motors. When a motor hits the end of the rail, the current required by it jumps suddenly to a much higher level. From this, we can use the sense resistor to pick a proportional voltage and when we reach a threshold voltage on this resistor, we know it must have reached the end.

The datasheet of the motors states that this stall current is somewhere in the range of 550mA to 650mA [8]. The sense resistors were placed internally in the power distribution board. A small resistance needs to be used such that it doesn't result in a large power loss or dip in voltage to the servo power supply, however needs to be high enough to produce distinguishable sense voltages, hence a resistance of approximately 1 Ω is used. You can see from figure 20 that the voltage across the sense resistor (in this case 1 Ω) is approximately 300mV when the servo is moving normally. When it reaches the end, the current spikes and, as shown, the voltage jumps from the nominal 300mV to approximately 800mV. We set a threshold of approximately 700mV (or using the lowest voltage of all the servos) and by reading the voltage across the sense resistor (using two pins into an analogue read pin), we can deduce when they have reached the end.

An advantage of using this method enables only 2 wires (potential from each end of the sense resistor) to be used rather than 23 wires (or 46 if we decide to sense both edges of the strip). This method enables us to sense both the upper and lower limits of the strip as due to the current-spike caused by the servos loading.

Drawbacks and Improvements: The method used is sometimes temperamental and requires calibrations and can become unreliable.

Figure 21 shows Figure 20 zoomed in. You can see small pulses in the waveform. This is due to the fact that the PWM signal sent to the servos cause the power to turn off and on due to the duty cycle. This in turn causes the current through the sense resistor to suddenly become 0 and then back to it's nominal (or maximum) voltage.

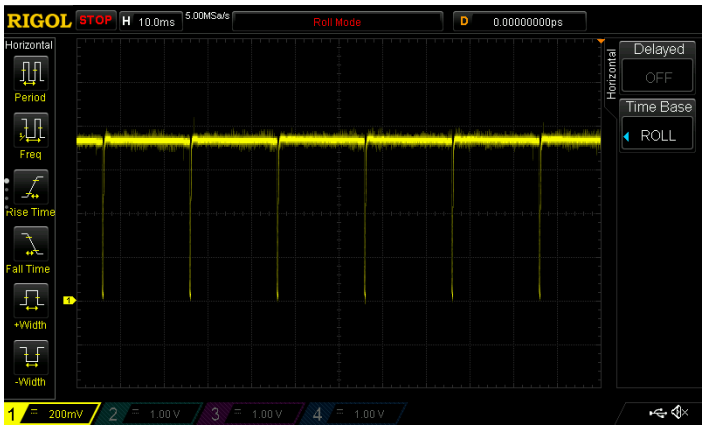


Figure 21 Zoomed in of the waveform just after an element hits the end of the strip.

Another issue we encountered is the fact that the threshold voltage changes from time to time. These issues are addressed in the software development (section 4.4.2) in more depth however we will go through some possible solutions here.

A hardware approach can be taken to solve these issues. One possible way is to introduce a comparator with a feedback mechanism (such as a thermistor) that considers the changes caused by the intolerances and, consequently, removes any of the variations present in the sense signal. While this would work, it would cause unnecessary complexity in developing new control circuitry for a small part of this project.

To remove the PWM variations, you could use a low-pass filter (similar to a debouncing circuit) to remove the sudden jumps to 0V. This would work however, the system could become unstable if a large number of strips were close to the edge, it would result in the consecutive current spikes becoming filtered out and eventually the system skips one or two strips.

As you can see, there is no perfect solution, along with the fact that the threshold constantly changes, a more optimum solution is required. From our prototype testing, we had concluded that a hardware solution for such an issue is not efficient and have settled on a software solution instead.

Possible software solutions and the chosen method is detailed in section 4.4.2.

3.2 Ergonomic Design

Careful consideration when into the design of the device to include braille elements for ease of interaction with the blind student, enabling them to interact with the device more independently and being more comfortable with the device. The location of the braille elements where based on drug labels for blind individuals, where the elements where place in a vertical manner on the sides. A reset button was also added to enable the user to reset the device to its starting position. This is connected to the reset pin of the microcontroller.

Figure 22 Engraved Braille Elements

The top of the device also has a braille element beside the reset button to allow the user to reset the device easily. (shown in Figure 24)

The Design of the clips had been made to be more rounded to create a more continuous feeling when the user runs their hands across the device to interpret the graphs, shown in figure 23

Figure 24 Braille Indentation for Reset Button

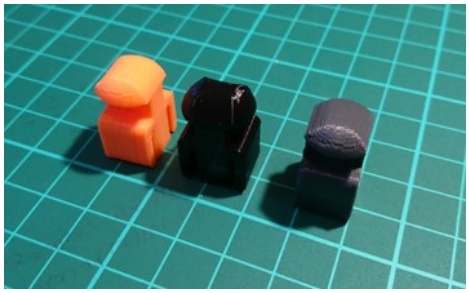


Figure 23 Smooth Interface for ergonomic interactions

The overall look and feel of the device were made as such to remove any unnecessary corners, this resulted in a smooth curvy design with flushed edges. In addition, screws were carefully selected to have rounded heads to prevent any cuts due to flat and sharp protrusions.

Figure 25 Final Device from the Side

3.3 Size, Shape and Weight (Portability)

3.3.1 Physical Dimensions and Weight:

The device has dimensions of 390mm x 210mm x 70mm (see figure 25). The length and width of the device fall within that of a laptop. The device weighs 1.8 kg. And the device can be carried in a haversack. Its shape comprises of rounded sides with flushed edges and a level top and bottom. The device can, as a result, be easy transported and used like a laptop computer in that regard.

3.3.2 Battery Portability (Power Supply):

Prior to choosing to use the device remotely, a 20W barrel connection adapter was going to be used, however, given this device could be used in classrooms and other remote locations without access to the mains, we decided to use a power source. In order to make the device as lightweight as possible, we avoided using bulky battery packs such as lead acid batteries. This narrowed the option down to using small Alkaline batteries or rechargeable batteries. To match our design specification, reduce environmental waste and increase the products life span and ease of use; rechargeable batteries was deemed to be a better option. Looking at the datasheets for the components, the power consumption of the system could be estimated.

Component	Voltage/V	Current/A
Driver circuit x2	0.5	0.2
Arduino	5	1
Servo motor x23	5	0.2

Since we are only using one servo at a time, this reduces the power consumption of the device at any one time. Hence, a 5V/3A power source would suffice.

There are a large variety of rechargeable batteries that could be used. For example, Lithium-ion 18650 batteries are commonly used to projects such as this as they are small and compact (slightly larger than AA batteries) while offering up to 3.7V and 2.15Ah. However, using these could lead to charge distribution and discharge issues. Therefore, using a battery pack at a set rating with circuit protection was chosen to overcome this issue.

4 **Software:** Development of Device – PC Interface

4.1 **Software Implementation:** User Application

4.1.1 Criteria

The purpose of the user application is to provide the people assisting the visual impaired the ability to select a graph they want to plot and upload it to the device. The interface was required to have the following functionality:

- Ability for user to enter a formula for a graph
- Ability for user to choose the mathematical distance between each point (step size)
- Preview image of the expected output of the device
- Upload the selection to the device

Subject to the functionality, the design criteria for the development of the user interface was chosen:

- Cross-platform (ability to run on different operating systems – Windows, Mac, Linux/Ubuntu)
- Intuitive to use
- Minimalist design
- Different mathematical formulae support
- Possibility of future improvement

Due to the design criteria mentioned and the skills of the team's software developers, Electron [11] was chosen as the framework for the app. It is an open-source framework based on Chromium [12] and NodeJS [13] for building cross-platform desktop apps using JavaScript, HTML and CSS.

React [14] together with Material UI [15], were used as the front-end frameworks for the application. This provided a simple component-based development process, and trusted and tested design components.

To make the development simpler and a large code base more manageable, Electron React Boilerplate [16], which is an open-source foundation for cross-platform application development, was used. This provided the project with structure, linting and pre-set running environments.

All the project files for the user application can be found in a public GitHub repository [17] using the following URL: https://github.com/saucena/blind-app

4.1.2 Design

The design of the user interface, shown in figures 26 and 27 includes the following:

- toolbar
- navigation bar with the project's name and the upload button
- text fields for the formula and step size inputs
- preview button
- area for the graph to be displayed

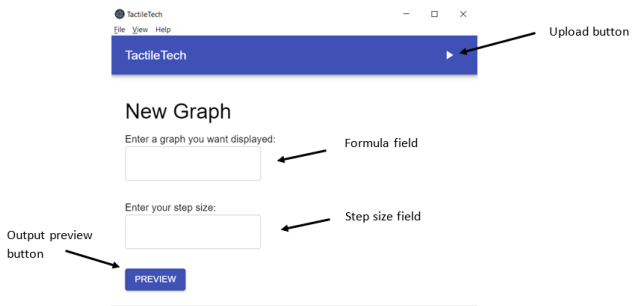


***Figure 26** User Application Design*

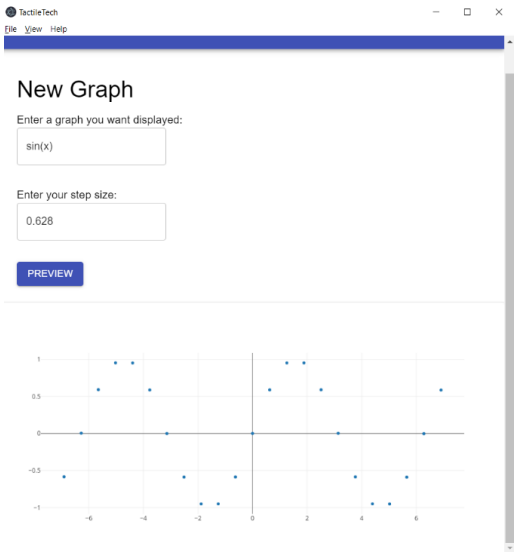


Figure 27 *User Application with Graph Preview*

4.2 Parser Implementation: User Application

4.2.1 Manually Written Expression

The first method of using formula inputs was to manually map each input to a specific function and only allow a finite number predefined by the developers. This was done using a switch statement, simplified with JavaScript's ability to declare functions as variables dynamically, see figure 28.

```
let fn;
switch (key) {
  case "x":
    fn = x => x;
    break;
  case "-x":
    fn = x => -x;
    break;
  case "x^2":
    fn = x => x*x;
    break;
  case "x^3":
    fn = x => x*x*x;
    break;
  case "sin(x)":
    fn = x => Math.sin(x);
    break;
  case "cos(x)":
    fn = x => Math.cos(x);
    break;
  default:
     fn = 'INVALID';
    break;
}
```

Figure 28 Using Switch Statement to manually code functions

The problem with this method is that the variety of graphs available is very limited and no mathematical transformations can be possible, such as multiplying the dependent variable, e.g. sin(2x), cos(3x).

4.2.2 Using a Parser

The solution to the problems mentioned in the previous chapter is to use a parser that converts the text to numerical values. Writing a proprietary parser is a very complicated task, thus an open source library *math-expression-evaluator* [18] was used to simplify this process.

This library converts expressions written in numbers and common mathematical operations but does not understand functions with variables. Thus, extra logic was added to convert a function with a dependent variable to an array of 23 values (the number of points in the device).

Firstly, the string is checked if it contains expressions with implied multiplication, i.e. does not contain the multiplication sign (e.g. 2x meaning 2*x). The input string is split into substrings using x as the separator and the substrings are checked if their last character is a number. If that is the case, a multiplication symbol is added. The code implementation of this is shown in Code Extract 2 on page 15.

A for-loop iterates 23 times (from index -11 to 11, representing the points on the device with the middle motor being the zero of the x-axis), each time multiplying the index by the step size and then using that number as the separator for joining the substrings. The resulting string is then passed to the parsing library object to give a numerical result, which is pushed into an array.

Since the Arduino code requires the values for the positions in the range of $0 - 100$, the resulting array is mapped in this range. This results in the fact that device will always draw the full shape of the graph in the 23-point range. Hence, formulas, such as "sin(x)" and "sin(x) + 2" will look identical, because both are identical between their minimum and maximum points. Thus, the y-axis is not of significance in the device, just the x-axis relative to the origin and the shape and behaviour of the graph.

The method of replacing x with a number each iteration currently limits the functions to those defined in math-expression-evaluator. Formulas that do not give real numbers or asymptotes are managed by not moving the motors and keeping them at the lowest position. The available mathematical operations can be seen in the appendix (Page 30). The code implementation of this is shown in Code Extract 3 on page 15.

When the user presses the Preview button, the app checks if the graph is valid. If yes, then it sends the 23 data points to Plotly [19], an API for drawing plots, which returns an image of the plot that is then displayed in the app.

```
If (step > 0) {
    let tmpExpression;

    for (
      let i = -Math.floor(moduleCount / 2);
      i <= Math.floor(moduleCount / 2);
      i++
    ) {
      axesOutput.x.push(i * Number(step));
      if (i * Number(step) < 0) {
        tmpExpression = expSplit.join(`(${i * Number(step)})`);
      } else {
        tmpExpression = expSplit.join(i * Number(step));
      }
      let value;
      try {
        value = mexp.eval(tmpExpression);
      } catch (error) {
        console.log(error);
        axesOutput.isValid = false;
      }
      axesOutput.y.push(value);
    }
  } else axesOutput.isValid = false;
```

Code Extract 1 Creating the Positions Array

4.3 Uploading a Sketch

After the user Previews the graph (not required, but ensures graph is valid), he/she presses the Upload button, which Uploads the positions of the motors to the Arduino.

The blueprint Arduino sketch is contained in the constants part of the project. Initially, the positions array, which holds the motor destined positions, is written to equal to the string '*INSERT HERE*'. This is a placeholder for the code to replace with actual values. When Upload is pressed, the program reads the blueprint as a string and replaces *INSERT HERE* with the calculated values as described in the previous part. Then a .ino file is written in the Arduino project folder with the replaced text inserted.

The sketch is compiled and uploaded using NodeJs child processes and rduino-cli [20]. Child processes allow to run commands in the command line, and rduino-cli is a command line for managing Arduino projects.

```
// ensure 2x => 2*x
  const expression = text.replace(/ /g, '');
  let expSplit = expression.split('x');
  expSplit = expSplit.map((x, i) => {
    const lastChar = x.charAt(x.length - 1);
    const num = Number.parseInt(lastChar, 10);
    if (!isNaN(num) && i < x.length - 2) {
      x += '*';
    }
    return x;
  });
```

Code Extract 2 Managing Implied Multiplication

```
const max = Math.max(
    ...axesOutput.y.filter(
      x => !isNaN(x) && x != null && x != 'Infinity' && x != '-Infinity'
    )
  );
  const min = Math.min(
    ...axesOutput.y.filter(
      x => !isNaN(x) && x != null && x != 'Infinity' && x != '-Infinity'
    )
  );
  axesOutput.y = axesOutput.y.map(x => {
    if (isNaN(x) || x == null || x == '-Infinity' || x == 'Infinity') {
      return min;
    }
    return x;
  });
```

Code Extract 3 Mapping non-real points to bottom of display

Testing with Different Inputs

Figure 29 (Left) Sinc(x), (Centre) 1/(x-0.1), (Right) (x-1)(x-5)

The software was tested with various commonly used mathematical functions as shown above. It is generally able to correctly display bounded (Gaussian), unbounded and shifted (x^2), periodic (sine wave) and discontinuous graphs (sinc(x) with discontinuous point at 0), and 1/(x-0.1) with vertical asymptotes.

Testing with different Step Sizes

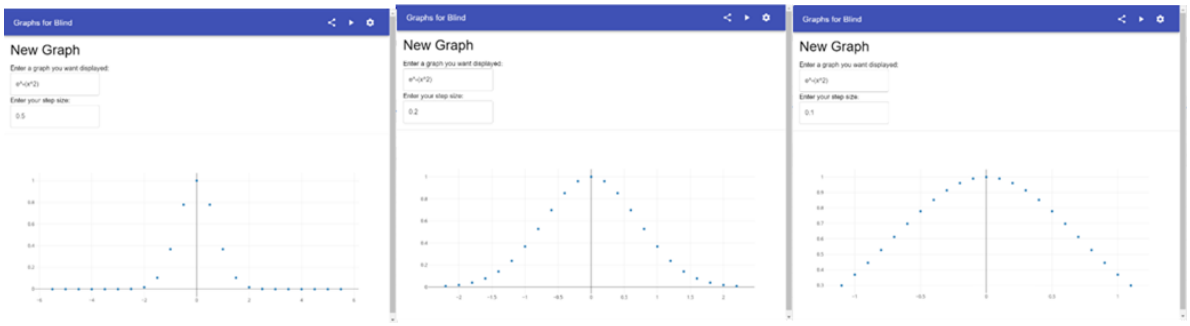


Figure 30 Gaussian plot at different h (left: h = 6, centre: h = 0.2, right: h = 0.1)

Increasing the step size also increases the window of the graph as the number of points remain constant at 23, which creates a "zooming out" effect, displaying larger horizontal sections of the graph as shown. More pictures of graphs at various step sizes are found in the appendix.

For periodic functions however, the step size needs to be adjusted with caution, with the correct output only displayed within certain ranges. For example, for the step size of 6, sin(x) is displayed as an inverted sine wave with period of 120, and for other step sizes it could lead to complete distortion of output shown on the left and centre:

Figure 31 Sine wave plot at different h - left: h = 6, centre: h = 3, right (correct): h = 0.273

To find the recommended step size, the display can be modelled as a discrete time sampled waveform. According to the Nyquist sampling theorem, the sampling period must be strictly less than twice that of the signal period for accurate representation, beyond which the graph can no longer be recovered, which explains the distorted output.

In this case, to show one complete period of the sine wave at maximum resolution, the ideal step size would be 2pi/23 = 0.273, with the graph displayed accurately as shown on the right above. Below that step size the sine wave is incomplete. Further increasing the step size would lead to more cycles displayed at lower resolution, with the recommended limit being estimated at 1.092 (see appendix).

4.4 Software Implementation of the Microcontroller (Arduino)

The Arduino script receives an array of 23 numbers from the application. These numbers are each between 0 and 100 and correspond to positions that the servos are required to move to (with 0 being the minimum and 100 the maximum values).

```
#define SERVO_FREQ 50 // The servo frequency, typically 50 Hz

#define MAXFORWARD 100 // This is the 'minimum' pulse length count
#define MAXBACK 500    // This is the 'maximum' pulse length count
#define STOP 4096      // This is the pulse length count that will stop the servo

#define MAXPOSITION 100 // The maximum position that can be entered, change to adjust precision
#define MAXDELAY 2250   // Maximum delay, tweak to fit strip size
```

Figure 32 Arduino Sketch Constants

The servos are connected to the Arduino via two Adafruit PWM drivers. Thus, their speed can be controlled by adjusting the duty cycle of the pulse width modulated signal. However, the libraries for the drivers are designed specifically for 180-degree servos, which have feedback circuits installed that allow them to be controlled by entering the angle that the user wishes them to move to.

However, the team's design uses 360-degree servos in order to achieve the full range of motion required by the strip lengths. Such servos do not have feedback circuits and cannot be controlled by angle. Thus, the library was reverse-engineered, and inputs were changed from angle values (0 to 180) to pulse width values (0 to 4096). Constants for pulse widths that moved the servos forwards and backwards at maximum speed and that stopped the servos were determined experimentally and used throughout the script. The constants are defined as shown in figure 32:

4.4.1 Moving the Servos

To accommodate for the continuous servos, the positions received from the app are mapped by the script onto durations that the servos need to be moving at maximum speed for. The script thus accurately moves the servos to sampled graph positions by turning them on at maximum speed, waiting for the required amount of time and then stopping them.

Initially, the team wished to move all servos at the same time. Thus, Arduino's "delay()" function could not be used to implement the delays for which each servo would keep moving. This is because delay() pauses the entire script and puts everything on hold. Since all servos needed to move for different lengths, it would not be possible to delay the servos simultaneously.

Therefore, another idea was conceived: the servos would be moved to quantised levels. All servos would be moved using delay() to the same level. Then, if any other servos needed to move any further, they would be moved together to the next level. This would continue until all servos were approximately at their correct positions. This method was dismissed because of the inaccuracy and needless complexity that it was found to bring to the device.

Then followed another method, which was to use Arduino timing: the code would measure the current time at every iteration of a loop and calculate the time that had passed since the servos had started moving. It would then check whether the time requested had passed for each servo and if it had would stop the corresponding servo. The code for this method is given in figure 33.

```
  for(int i = firstServo; i ⩽ lastServo; i++)
      pwm.setPWM(i, 0, MAXCLOCKWISE);
}

// Arduino keeps checking whether the specified wait times have been reached
// and if so stops the corresponding servo
void loop() {
  //Does operation for each connected servo
    for(int i = 0; i ⩽ lastServo-firstServo; i++) {
        unsigned long currentMillis = millis();
        interval = (long) waitTime[i];
        Serial.print("Interval: ");
        Serial.println(interval);
        if (currentMillis ⩾ interval) {
            Serial.print("Stopping servo ");
            Serial.println(i+1);
            pwm.setPWM(firstServo+i, 0, STOP);
        }
    }
}
```

Figure 33 Simultaneous Servo Movement Function

It is worth noting that the above code is only for one PWM servo driver, since this idea did not make it to the final design which used two drivers. When this method was used to draw graphs using an early prototype of the device and a bench power supply, it was found to be ineffective due to two reasons:

1. The time is calculated only once per iteration of the loop, which itself takes time. Due to this, the distances servos travelled were extremely inaccurate and the graphs looked nothing like they were supposed to, and
2. Even when the code was tested with only 10 servos, the device drew a very large amount of power, and it was found that most affordable and portable battery packs would not be able to handle the load produced by all 23 servos functioning simultaneously.

Due to the above reasons, the team decided to go with the simple, accurate Arduino delay() function, which meant that each servo would need to be moved individually.

This is implemented in the function shown in figures 34 and 35.

```
for (int i = lastServo2; i ⩾ firstServo2; i--)
    moveServo(i, positions[i + lastServo1 + 1], false);

for (int i = lastServo1; i ⩾ firstServo1; i--)
    moveServo(i, positions[i], true);
```

Figure 35 Individual Servo Movement Function

```
void moveServo(int n, int pos, bool first, bool forward) {
    int direct = forward ? MAXFORWARD : MAXBACK;

    if (first) pwm1.setPWM(n, 0, direct);
    else pwm2.setPWM(n, 0, direct);

    int waitTime = map(pos, 0, MAXPOSITION, 0, MAXDELAY);
    delay(waitTime);

    if (first) pwm1.setPWM(n, 0, STOP);
    else pwm2.setPWM(n, 0, STOP);
}
```

Figure 34 Individual Servo Movement Function

4.4.2 Resetting the Servos (to the initial position)

Before drawing a graph, the servos must return to their initial positions (the end of their strips, position 0). Since they cannot be controlled by angle, the trivial reset method is to move each of them backwards for MAXDELAY and force them all to reach the end. However, this method does not consider that not all servos are at maximum position when a graph is drawn. Servos that are close to the maximum position will not cause problems but those closer to the initial position will stall for the majority of that delay. As discussed earlier this will not cause an issue immediately however it will significantly reduce the lifetime of the servos and the device will begin to fail after short use.

As seen in the hardware, we implemented a sense resistor which showed experimental results of approximately 700mA when the servo stalled. It was later found via experimental use that 1Ω is the optimal value for the sense resistor, since it represents the best balance between measurable (much larger than electrical noise) and non-obstructive (small enough to not cause noticeable power loss).

Since all servos are different, the script contains threshold calibration functions that, assuming that the servos are already at the end of their strips, move each of them backwards, continuously measure their current spikes and save the smallest value in the Arduino's EEPROM.

The functions are shown in figure 36.

After the thresholds are determined, the rest is handled by the reset function. Each servo is moved backwards until the voltage drop threshold is reached, which is when it is stopped.

```
void findAllThresholds(bool forward) {
    int j = forward ? 0 : 23;
    int temp;
    for (int i = firstServo1; i ⩽ lastServo1; i++) {
        temp = findThreshold(i, true, forward);
        EEPROM.write(j, temp);
        j++;
    }
    for (int i = firstServo2; i ⩽ lastServo2; i++) {
        temp = findThreshold(i, false, forward);
        EEPROM.write(j, temp);
        j++;
    }
}

int findThreshold(int n, bool first, bool forward) {
    int minValue = 1000;
    int temp;
    int direct = forward ? MAXFORWARD : MAXBACK;

    if (first) pwm1.setPWM(n, 0, direct);
    else pwm2.setPWM(n, 0, direct);

    for (int i = 0; i < 100; i++) {
        delay(5);
        temp = analogRead(A2) - analogRead(A3);
        if (temp < minValue)
            minValue = temp;
    }

    if (first) pwm1.setPWM(n, 0, STOP);
    else pwm2.setPWM(n, 0, STOP);
    return temp;
}
```

Figure 36 Threshold Calibrating Functions

```
void resetServo(bool forward) {
    int j = forward ? 0 : 23;
    int direct = forward ? MAXFORWARD : MAXBACK;
    for (int i = firstServo1; i ⩽ lastServo1; i++) {
        Serial.println(j);
        pwm1.setPWM(i, 0, direct);
        Serial.print("Threshold for this motor: ");
        Serial.println(EEPROM.read(j));
        delay(10);
        Serial.println(analogRead(A2) - analogRead(A3));
        while ((analogRead(A2) - analogRead(A3)) < EEPROM.read(j)) {
            Serial.println(analogRead(A2) - analogRead(A3));
            delay(1);
        }
        pwm1.setPWM(i, 0, STOP);
        j++;
    }

    for (int i = firstServo2; i ⩽ lastServo2; i++) {
        Serial.println(j);
        pwm2.setPWM(i, 0, direct);
        Serial.print("Threshold for this motor: ");
        Serial.println(EEPROM.read(j));
        delay(10);
        Serial.println(analogRead(A2) - analogRead(A3));
        while ((analogRead(A2) - analogRead(A3)) < EEPROM.read(j)) {

            Serial.println(analogRead(A2) - analogRead(A3));
            delay(1);
        }
        pwm2.setPWM(i, 0, STOP);
        j++;
    }
}
```

Figure 37 Servo Reset Function

There are some problems with this method, two of which turned out to be significant enough that the code was altered to accommodate for them. Below are some issues (determined during testing):

1. The servos need significant current to be accelerated from rest, because the static friction coefficients of plastic and rubber (which are the materials that the gears, servo and timing belt are made from) are higher than the dynamic ones. (Thus, the servos need to do more work to quickly accelerate the belts from rest). This meant that the servos caused current spikes when being started and occasionally caused the thresholds to be reached, stopping the servos immediately.

To resolve this issue, a delay of 10 milliseconds was introduced before the code starts checking for the thresholds.

2. The current spikes were sometimes inconsistent, which occasionally resulted in the thresholds not being reached, causing a servo to keep moving indefinitely.

This issue was resolved by reducing the thresholds by an appropriate amount which was empirically determined.

The above issues were resolved successfully. Therefore, the team proceeded to use this approach when resetting the servos on the final version of the product.

Although this method was a success, there was still a minor issue that could not be resolved:

- Over time with repeated usage, the resistance of the resistor changed, which interfered with the current spikes, reducing the reliability of the system.

Other method such as setting a dynamic threshold were considered where the device will check for changing and reform threshold detection. It was also considered to introduce feedback in software considering temperature of the circuitry. These ideas was abandoned for the unneeded complexity. In the end, this issue was resolved by using a higher quality resistor with a lower spread (due to temperature).

4.4.3 Calibration

Every servo has a slightly different top speed. On top of this, every belt has a slightly different tension and all gears are slightly different. All these factors add up and cause severe imprecision when drawing graphs. For this reason, calibration is required when moving the servos.

Two ways of calibrating the servos were tested. The initial idea was to use current spikes once more. Assuming that all servos are at their maximum position, a function would move them back until the current spike threshold was reached, and measure the time taken for the servo to move the entire distance. Afterwards, the time taken for the slowest servo would be recorded and used to calibrate all other servos.

```
// Adds a servo's calibration factor to an array calibrationFactors.
void calibrateServo(int n, bool first, int threshold) {
    unsigned long startTime = millis();
    if (first) pwm1.setPWM(n, 0, MAXBACK);
    else pwm2.setPWM(n, 0, MAXBACK);

    while (analogRead(A2) - analogRead(A3) < threshold)
        delay(1);

    unsigned long endTime = millis();
    pwm2.setPWM(n, 0, STOP);
    unsigned long delta = endTime - startTime;

    calibrationFactors[n] = calibrationTime / delta;
}

// Calculates all calibration factors and adds them to the array.
void calibrateAll() {
    int i = 0;
    for (i + firstServo1; i ≤ lastServo1; i++)
        calibrateServo(i, true, EEPROM.read(23 + i));
    i = 0;
    for (i + firstServo2; i ≤ lastServo2; i++)
        calibrateServo(i + lastServo1, true, EEPROM.read(23 + i));
}
```

Figure 38 Servo Calibration Functions

The functions used to do this are shown in figure 38: calibrateServo calibrates a single servo n and saves its corresponding calibration factor in an array. Here, calibrationTime is the time taken by the slowest servo.

Additionally, calibrateAll runs the calibration code for each servo by reading the correct thresholds from memory and builds the calibration array.

In theory, this seemed perfect. However, due to the reliability issues associated with the current spike sensing method, the servos did not always stop when required and the calibration array was incomplete and/or inaccurate.

```
int calibrationFactors[23] = [13.750000000000002, 6.874999999999998, 9.375, 4.375000000000001, 5.625000000000002, 0.6249999999999978,
                              8.124999999999998, 10.625000000000002, 1.874999999999999,9.375, 12.5, 4.999999999999999, 1.874999999999999, 4.999999999999999,
                              3.125, 6.25, 3.125, 6.25, 6.25, 11.249999999999998, -0, 10.625000000000002, 14.374999999999998];

const unsigned long calibrationTime = 1900;
```

Figure 40 Servo Calibration Functions Array

```
// Adds a servo's calibration factor to an array calibrationFactors.
void calibrateServo(int n, bool first, int threshold) {
    unsigned long startTime = millis();
    if (first) pwm1.setPWM(n, 0, MAXBACK);
    else pwm2.setPWM(n, 0, MAXBACK);

    while (analogRead(A2) - analogRead(A3) < threshold)
        delay(1);

    unsigned long endTime = millis();
    pwm2.setPWM(n, 0, STOP);
    unsigned long delta = endTime - startTime;

    calibrationFactors[n] = calibrationTime / delta;
}

// Calculates all calibration factors and adds them to the array.
void calibrateAll() {
    int i = 0;
    for (i + firstServo1; i <= lastServo1; i++)
        calibrateServo(i, true, EEPROM.read(23 + i));
    i = 0;
    for (i + firstServo2; i <= lastServo2; i++)
        calibrateServo(i + lastServo1, true, EEPROM.read(23 + i));
}
```

Figure 39 Servo Move Function Calls in Setup() with calibration factors

Due to this, another calibration method was used. This time, all servos were moved to position 0, and then code was written to move them to 50% of the length of their strips. Afterwards, the offset of each servo from the exact 50% position was manually measured and the differences were recorded. Then, an array consisting of percentages of the offsets to the entire length of the strips was created. This array is shown in figure 40.

When this array was constructed, it was used to modify the delay of each servo. This was accomplished by subtracting each servo's calibration factor from its corresponding position when calling the moveServo function.

At the end of the manual calibration process, the servos were much better coordinated, and the graphs looked significantly more accurate and smoother. Figure 41 contains an example of a graph drawn with manual calibration; a gaussian graph is clearly visible.

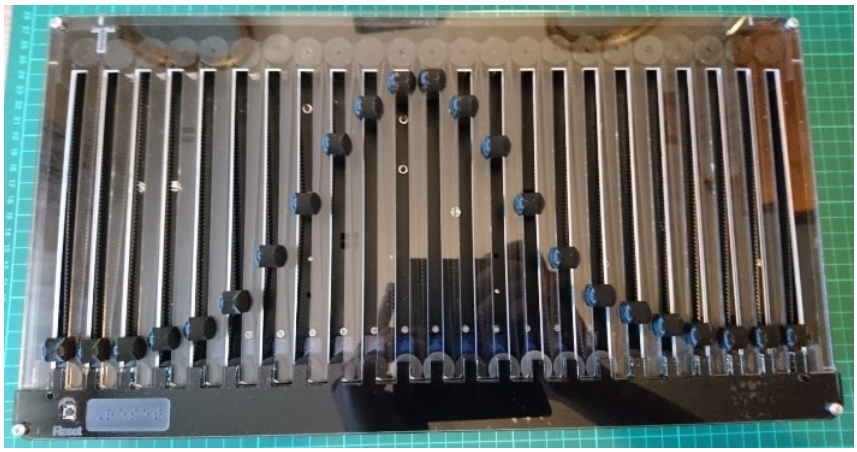


Figure 41 Example Gaussian Wave plotted with Calibration

5 Project Management

5.1 Organisation of Work

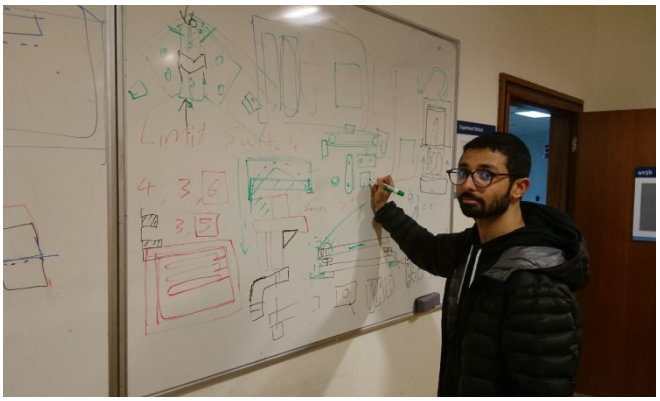


Figure 42 Brainstorming session in early February

The team was split into two technical groups: hardware and software. Umut and Pavan were responsible for the mechanical design with Alp, Omar and Kevin helping with assembly. Issa oversaw power distribution. Lukas developed the user application, and Arman created the Arduino for controlling the motors. However, these roles were not strict, especially in the final stages, where all members worked on multiple parts, helped with assembly and testing.

Regarding managerial positions, Lukas was the team leader, who organized team meetings and oversaw the progress of each subgroup. Alp was responsible for managing the budget and inventory purchases. Arman and Kevin were responsible for keeping track of the meetings and documentation.

The team held meetings every Tuesday, where team progress was shared, prototypes of mechanical pieces and software were shown and discussed. Each meeting would end with a conclusion and agenda for the upcoming week. Communication was done through WhatsApp, documents were stored on One Drive, software files on GitHub.

5.2 Cost and Budget

Even though the total budget provided for the project was 450 pounds, the amount spent was around 380 pounds. After the hardware plan was done, it was decided that the majority of the spending will be on the servo motors and the timing belts. The servo motors cost around 230 pounds whereas the timing belts were around 85 pounds. As the number of servo motors and timing belts to be used was not known definitely at the beginning, bulk purchase with a good deal was not possible, this contributed to an increase in the total spent.

Apart from the budget reserved for the external orders, there was an opportunity to 3D-print pieces (e.g. gears or rods for the belt) required for the prototype. Overall, the items purchased were researched in detail and compared with their alternatives to give the best choice in the manner of cost and efficiency. From our budget, we can see that we can potentially sell a device like this on the market for approximately 300 GBP which is very impressive considering the existing range of aid equipment for Blind people.

Please see the appendices for a detailed list of all purchases regarding our project.

5.3 Future Work

Further miniaturisation: The device as of now, although small enough to fit in a backpack, is still too large considering some of the space inside. A lot of empty space is wasted with the slack belts used in the mechanism. If the project was to be done again, a new method would be considered to bring each servo closer together and incorporate some of the circuitry in this wasted space. This would result in the system being smaller in dimensions including its thickness. By miniaturising the system, more strips can be added for the same given dimensions increasing the resolution of the device.

Hardware: The hardware could see an improvement in the choice of motors from a servo motor that works based on time delays, to a stepper motor that works based on number of rotations. This provides an added accuracy that is not too depend on the accuracy of manufacturing and the power levels in battery packs. The stepper motors in the form factor of a linear actuator.

Moreover, the current bread boards can be made into PCB boards to improve is durability and prevent loss of signals due to lose wires. Additionally, a wireless Arduino could be introduced to allow the tutor to teach multiple students at the same time. The device could also see further developments to add an on-board memory to store past graphs and allow the user to input data, making the device more interactive for the blind students.

Software: The user application can be improved further by adding higher-level mathematical functions, such as *rect(x)*, *sgn(x)*, etc.

Furthermore, future versions should include the ability to sample and display any 2D graph, uploaded as an image or other format. This way the device's usage would extend to not just mathematical functions, but also other graphs and charts, such as stock market graphs, oscilloscope readings and other uses.

In addition, the device should be shifted to be a standalone device for the blind person, such that an assisting person is not needed. This would mean including many ease-of-access capabilities, such as text-to-speech, inputs by voice, and other visual disability compliances.

5.4 Conclusion

In conclusion, the prototype was a success, with various commonly used graphs being correctly translated and mapped onto the device, meeting all the design specifications to a satisfactory level. The high degree of accuracy and resolution is accomplished through combining the thoughtfully designed mechanical structure and large quantity of belt actuators with a meticulously calibrated and well-coordinated control system that accounts for all its inevitable imperfections. Versatility is also achieved through a highly flexible user interface software that enables a wide variety of graphs at different step sizes to be represented through parsing.

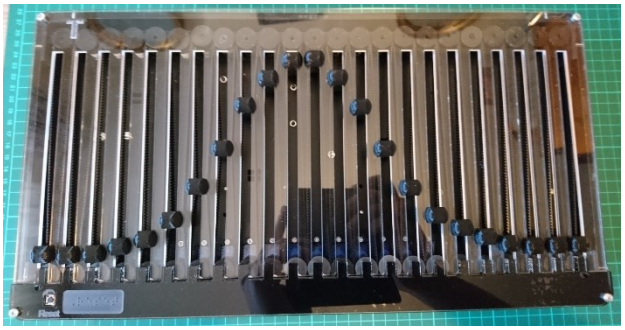


Figure 43 Example Gaussian Wave plotted with Calibration

Most importantly, it is also the first product of its kind, and is also widely accessible, developed with a cost (300 GBP) unmatched by other products on the blind assistance market. With additional improvements it could pave the way for a new niche of graphical interpretation products for the blind to be created, and potentially become a long-term solution for boosting education and job prospects among the blind.

6 References

[1] L. U. G. Department, "Blindness and Visual Impairment," [Online]. Available: https://www.loc.gov/nls/resources/blindness-and-vision-impairment/devices-aids/braille-displays-notetakers/ .

[2] National Federation of the Blind (NFB), "Blind Statistics regarding Education Levels," January 2019. [Online]. Available: https://www.nfb.org/resources/blindness-statistics.

[3] R. N. I. o. t. B. U. (RNIB), "Key Information and Statistics on Sight loss in the UK," 1 February 2018. [Online]. Available: https://www.rnib.org.uk/professionals/knowledge-and-research-hub/key-information-and-statistics.

[4] W. H. Organisation, "Global Data on Visual Impairment 2010," 1 December 2011. [Online]. Available: https://www.who.int/blindness/publications/globaldata/en/.

[5] TapTapSee App, "TapTapSee Application," [Online]. Available: https://taptapseeapp.com/. [Accessed 12 December 2019].

[6] R. N. I. o. t. B. Store, "Tiger SpotDot braille embosser printer - red," [Online]. Available: https://shop.rnib.org.uk/accessible-technology/note-taking-and-embossers/tiger-spotdot-braille-embosser-printer-red.html. [Accessed 18 October 2019].

[7] R. N. I. o. t. B. Store, "Braille Sense U2 Mini Braille Notetaker," [Online]. Available: https://shop.rnib.org.uk/accessible-technology/note-taking-and-embossers/braille-sense-u2-mini-braille-notetaker.html. [Accessed 18 October 2019].

[8] FeeTech, "FS90R Continous Servo Datasheet," [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/Adafruit%20PDFs/2442˙Web.pdf. [Accessed March 2020].

[9] Rapid Electronis, "Feetech FS90R Motors," [Online]. Available: https://www.rapidonline.com/feetech-fs90r-360-continuous-rotation-micro-servo-37-1335. [Accessed 25 March 2020].

[10] P. Printables, "Potent Printables YouTube Video," Potent Printables, [Online]. Available: https://www.youtube.com/watch?v=2vAoOYF3m8U&feature=youtu.be) . [Accessed 2020 February 27].

[11] Electron JS, "Electron JS Homepage," [Online]. Available: https://www.electronjs.org/.

[12] Chromium Organisation, "Open Source Chromium Webpages," [Online]. Available: https://www.chromium.org/.

[13] NodeJS, "NodeJS Opensource Homepage," [Online]. Available: https://nodejs.org/.

[14] ReactJS, "ReactJS Homepage," [Online]. Available: https://reactjs.org/. [Accessed March 2020].

[15] Material UI, "Material UI Homepages," [Online]. Available: https://material-ui.com/. [Accessed March 2020].

[16] E. React, "Electron React Boilerplate Source Files," [Online]. Available: https://github.com/electron-react-boilerplate/electron-react-boilerplate.

[17] Team 7 Group Project, "Visual Access System Source Files," [Online]. Available: https://github.com/saucena/blind-app. [Accessed 20 March 2020].

[18] Bugwheels94, "Math Expression Evaluator Github," [Online]. Available: https://github.com/bugwheels94/math-expression-evaluator.

[19] Plotly, "Plotly Home Page," [Online]. Available: https://plotly.com/.

[20] Arduino, "Arduino CLI Github Respository," [Online]. Available: https://github.com/arduino/arduino-cli.

[21] D. Darling, "Type Writer Mechanism in Use," [Online]. Available: https://www.daviddarling.info/encyclopedia/L/lever.html. [Accessed February 2020].

[22] M. Richmond, Artist, *Conveyor Belt Analogy.* [Art]. Simon Tulloch Figure.

[23] B. Siber, "Which 3D Printing Belt to consider," [Online]. Available: https://all3dp.com/2/3d-printer-belt-what-to-consider-and-which-to-buy/. [Accessed 15 March 2020].

7 Appendix

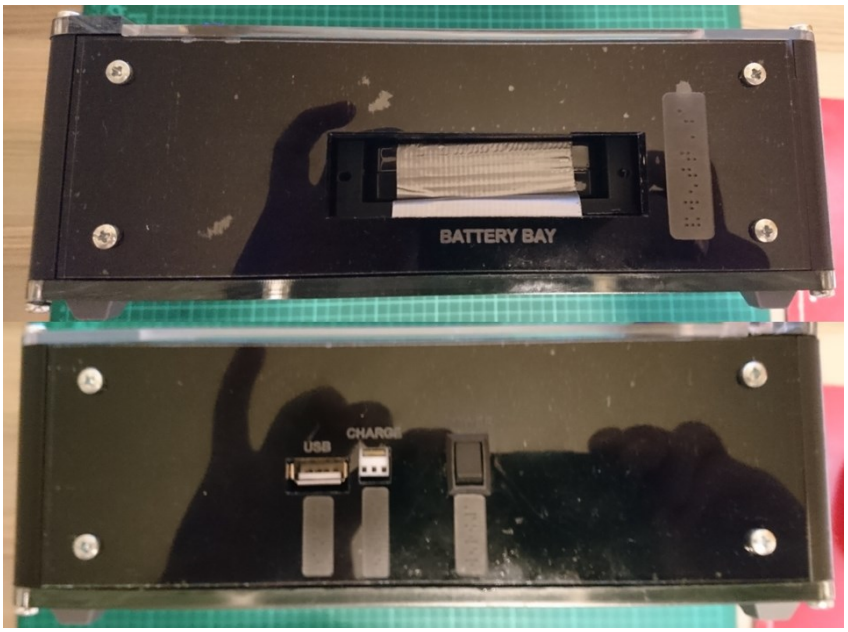
Below is some additional reading material for this report.

Final Device Pictures

Appendix Figure 2 Final Device (Plastic)

Appendix Figure 1 Back of Device

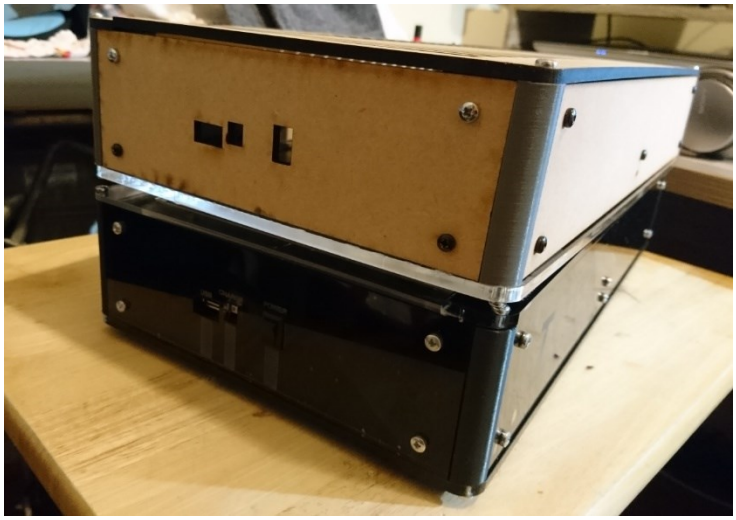
Appendix Figure 4 Back and Front of the Sides



Appendix Figure 3 Left and Right of the Sides

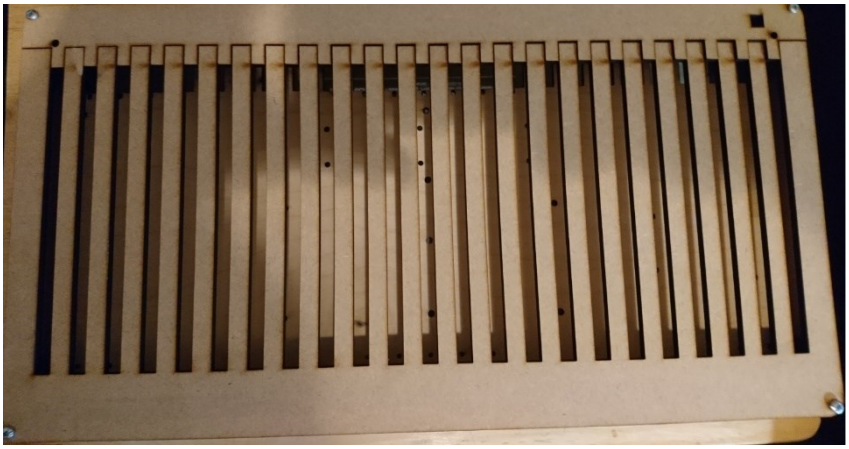
Wooden Prototype Device Pictures

Wooden and Plastic Comparisons. A wooden prototype was constructed before the final plastic version. Top prototype is wooden whereas the bottom one is the final plastic one.



Appendix Figure 7 Right Side of Device

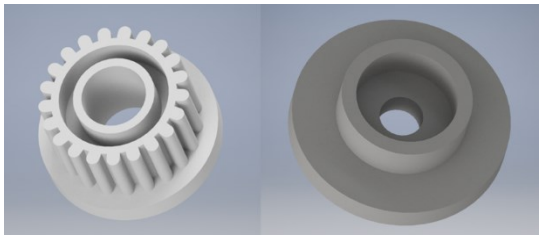
Appendix Figure 6 Left Side of Device



Appendix Figure 5 Top of Device

Gear and Rod Design

The gear was designed to mimic that of a 3D printer gear, and it was also designed to be printed with no supports, hence it was split into two parts the gear and the lid with an inter-locking mechanism shown below.



Appendix Figure 9 Interlocking Mechanism

Appendix Figure 8 T Shaped Shaft

Both the lid and the gear have different diameter openings which allows it to be dual purposed. The smaller diameter allows for it to be screwed into the servo and the larger opening allows it to freely rotate on the rod. The rod is designed to have four screw points a T shaped base for support to withstand the tension exerted by the tighten belt. The rod additionally has a fillet transition between the protruding surface and the T shaped base, to add some rigidity to the design shown in Appendix Figure 8.

Calculating the Step-size Limit

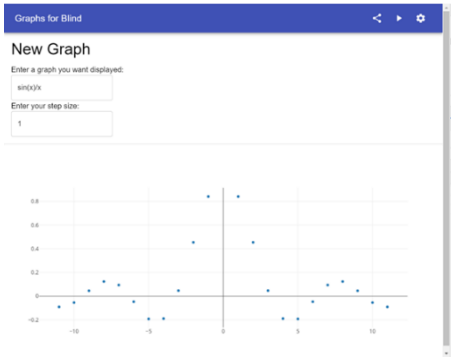
Appendix Figure 10 The boundary of step sizes

An increase in step size would imply a larger sampling interval, which implies a lower sampling frequency. With the number of samples remaining constant at 23, the sampling window is also larger, leading to more cycles of the same graph displayed.

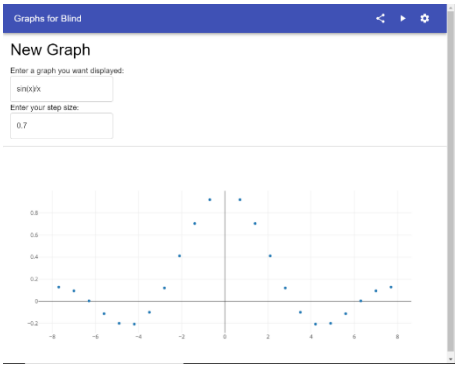
Therefore, increasing the step size for the same sine wave (or zooming out) directly corresponds to increasing the frequency of the sine wave for the same step size, with both leading to the same displayed output.

When the frequency of the sine wave is increased by four times for the same step size as shown on the right, the graph starts to lose its precision as large, irregular changes start occurring between points, making it difficult for a blind person to capture every detail of the graph. This also corresponds to a step size four times the original, which is 1.092, and is therefore recommended maximum step size for any graph with period 2π.

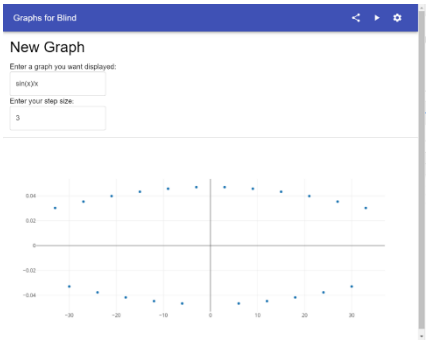
Sinc(x) of different Step Sizes



Appendix Figure 121 Step Size = 1

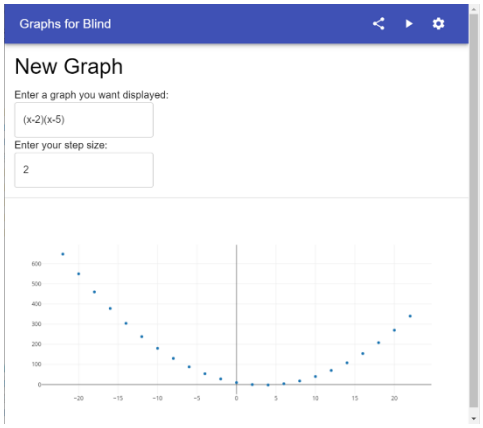


Appendix Figure 12 Step Size = 0.7



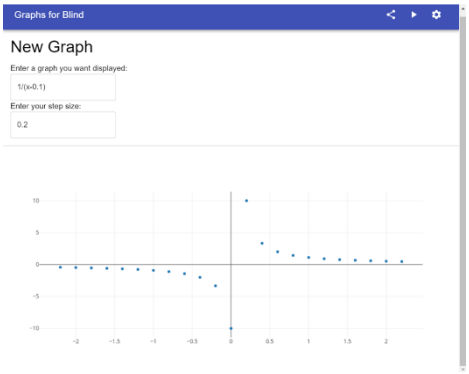
Appendix Figure 12 Step Size = 3

Other Examples of Graphs of different Step Sizes



Appendix Figure 13 (x-2)(x-5)

Appendix Figure 14 Log(x)



Appendix Figure 15 1/(x-0.1)

Different Operations and Functions available in the used Parser.

Operator/Function	Description
+	Addition operator
-	Subtraction operator
/	Division operator
*	Multiplication operator
(	Opening parenthesis
)	Closing parenthesis
pi	Math constant pi returns 3.14
e	Math constant e returns 2.71
log	Logarithmic function with base 10
ln	Natural log function with base e
^	Power operator
root	Square root function
sin	Sine function
cos	Cosine function
tan	Tangent function
asin	Inverse Sine function
acos	Inverse Cosine function
atan	Inverse Tangent function
sinh	Hyperbolic Sine function
cosh	Hyperbolic Cosine function
tanh	Hyperbolic Tangent function
asinh	Inverse Hyperbolic Sine function
acosh	Inverse Hyperbolic Cosine function
atanh	Inverse Hyperbolic Tangent function

			Total Spent (£)	376,42				
			Total Remaining (£)	73,58				
Order Number	Item Ordered	Date Executed	Responsible Contact	Price Per Quantity (£)	Quantity	Total Price (£)	Software / Hardware	Supplier
1	Feetech FS90R 360° Continuous Rotation Micro Servo	17.01.2020	Kevin	6,05	10	60,5	Motor	Rapid Online
2	Plastic Spacers	31.01.2020	Umut	0,1	10	1	PCB Assembly	EEED
3	3mm LaserPly	31.01.2020	Umut	1,84	1	1,84	Prototyping	EEED
4	4mm LaserPly	31.01.2020	Umut	2,22	1	2,22	Prototyping	EEED
5	Adafruit pca9685 16-channel servo driver	7.02.2020	Kevin	5,49	2	10,92	Motor Control	Hobby Components
6	Jumper Wire for PCB Breadboard	7.02.2020	Kevin	2,41	1	2,41	Prototyping	Rapid Online
7	GT2 6mm Timing Belt 2mm Pitch	13.02.2020	Umut	41,26	1	41,26	Transport Design	RoboSavvy
8	Feetech FS90R 360° Continuous Rotation Micro Servo	17.02.2020	Kevin	6,05	10	60,5	Motor	Rapid Online
9	Tactile SMD THT button	21.02.2020	Umut	0,24	1	0,24	Case Design	EEED
10	3mm LaserPly	21.02.2020	Umut	1,84	2	3,68	Prototyping	EEED
11	3mm Acrylic Clear	21.02.2020	Umut	4,9	1	4,9	Prototyping	EEED
12	SuperGlue All Purpose	21.02.2020	Umut	1,26	2	2,52	Prototyping	EEED
13	GT2 6mm Timing Belt 2mm Pitch	28.02.2020	Umut	41,26	1	41,26	Transport Design	RoboSavvy
14	Feetech FS90R 360° Continuous Rotation Micro Servo	28.02.2020	Umut	6,05	10	60,5	Motor	Rapid Online
15	3mm LaserPly	28.02.2020	Umut	1,84	3	5,52	Prototyping	EEED
16	4MM LASER-GRADE BIRCH PLY	6.03.2020	Pavan	2,22	3	6,66	Belt design struture	EEED
17	MINIATURE 240V SPDT VERTICAL PCB SLIDE	6.03.2020	Pavan	0,59	3	1,77	Belt design struture	EEED
18	TACTILE SMD 6 X 6MM	6.03.2020	Pavan	0,24	6	1,44	Switches (not used)	EEED
19	Plastic Spacers	11.03.2020	Umut	0,1	20	2	Prototyping	EEED
20	Small Stripboard	11.03.2020	Umut	0,24	4	0,96	Prototyping	EEED
21	Medium Stripboard	11.03.2020	Umut	2,19	2	4,38	Prototyping	EEED
22	USB Cable	11.03.2020	Umut	2,1	2	4,2	Final Construction	EEED
23	Feet	11.03.2020	Umut	0,1	10	1	Final Construction	EEED
24	4MM LASER-GRADE BIRCH PLY	12.03.2020	Pavan	2,22	2	4,44	Belt design struture	EEED
25	6MM LASER-GRADE BIRCH PLY	12.03.2020	Pavan	3,1	1	3,1	Belt design struture	EEED
26	3mm Acrylic Clear	12.03.2020	Pavan	4,9	1	4,9	Belt design struture	EEED
27	5mm Acrylic Clear	12.03.2020	Pavan	8,14	5	40,7	Belt design struture	EEED
28	Feet	12.03.2020	Umut	0,1	16	1,6	Hardware	EEED
29	Feetech FS90R 360° Continuous Rotation Micro Servo	12.03.2020	Kevin	4,8	10	48	Spare Motors	Rapid Online

Appendix Figure 16 External Orders

Order Number	Item Ordered	Date Executed	Responsible Contact	Price per quantity (£)	Quantity	Total Price (£)	Software / Hardware	Supplier
1	GT2 Timign Belt	2.02.2020	Umut	2,99	1	2,99	Hardware	Amazon
2	MDF Wood Boards	11.03.2020	Pavan	4	3	12	Belt design struture	Robotics Junk Box
3	Black vinyl adhesive	11.03.2020	Pavan	8	1	8	Sticker for glossy look	Wilko DIY Store
4	USB Cable	11.03.2020	Pavan	0	1	0	Electrical connectors	Robotics Junk Box
5	USB port	11.03.2020	Umut	0	1	0	Hardware	Home
6	Connectors and headers	11.03.2020	Umut	0	5	0	Hardware	Robotics Junk Box
7	RS Battery	11.03.2020	Umut	0	1	0	Hardware	Rover project
8	Silicon Wire and Heat Shrink	11.03.2020	Pavan	0	4	0	Wiring	Personal
9	Heat Strinks	11.03.2020	Umut	0	100	0	Hardware	Personal

Appendix Figure 17 Funds Spent outside of Budget

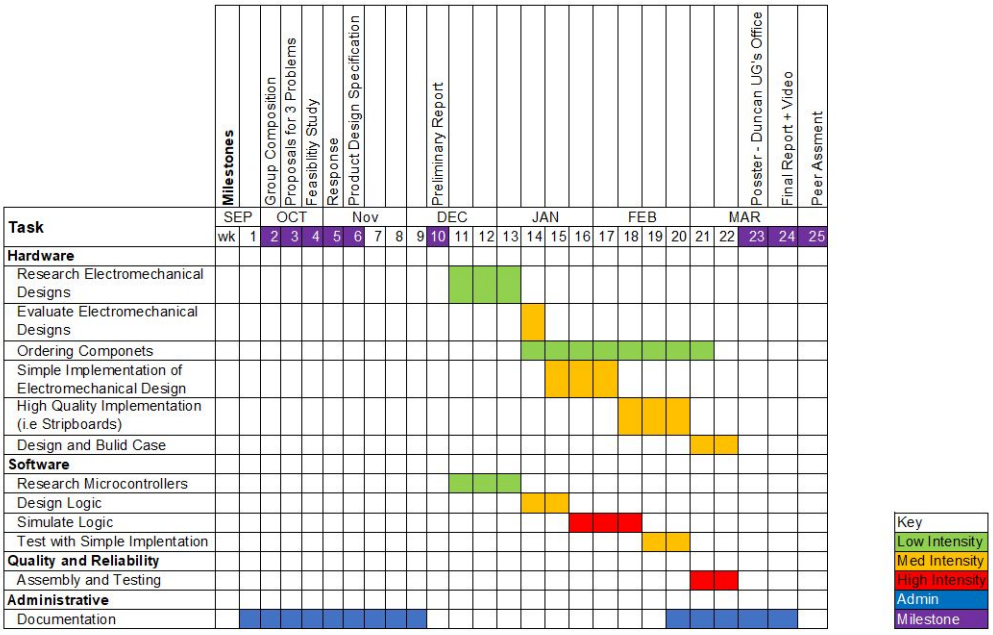
Order Number	Order Detail	Order Date	Order Quantity	3D Printed By	Hardware	Note
1	H-Design Clip	12.02.2020	3	Umut	Belt	Failed Print
2	Mini Gear	14.02.2020	2	Umut	Belt	Failed Print
3	Side Slots	18.02.2020	4	Umut	Belt	Success Print
4	Mini Gear Shaft	18.02.2020	3	Umut	Belt	Failed Print
5	Gears	21.02.2020	2	Pavan	Belt	Success Print
6	Gear Covers	21.02.2020	2	Pavan	Belt	Success Print
7	V1 of Rod	21.02.2020	1	Pavan	Belt	Success Print
8	V2 of Rod	21.02.2020	2	Pavan	Belt	Success Print
9	H-Design Dome Clip	27.02.2020	1	Pavan	Belt	Failed Print
10	Curved corner Clamps	10.03.2020	8	Umut	Belt	Success Print
11	Shaft	12.03.2020	6	Pavan	Belt	Success Print
12	Gears	16.03.2020	24	Pavan	Belt	Failed Print
13	Gear Covers	16.03.2020	24	Pavan	Belt	Failed Print

Appendix Figure 18 3D Printing in the 5th Floor EEE Labs

Order Number	Order Detail	Order Date	Order Quantity	3D Printed By	Hardware
1	Motor Brackets	17.01.2020	2	Kevin	Motor
2	Slider	19.01.2020	1	Pavan	Rack and Pinion
3	Cut_slider	20.01.2020	1	Pavan	Rack and Pinion
4	Slider20cm	21.01.2020	1	Pavan	Rack and Pinion
5	Sliderbase	21.01.2020	1	Pavan	Rack and Pinion
6	Sliderupdated	21.01.2020	1	Pavan	Rack and Pinion
7	Run	21.01.2020	1	Pavan	Rack and Pinion
8	Part1	22.01.2020	1	Pavan	Rack and Pinion
9	Part2	22.01.2020	1	Pavan	Rack and Pinion
10	RunV3	26.01.2020	1	Pavan	Rack and Pinion
11	SliderV3	26.01.2020	1	Pavan	Rack and Pinion
12	gearv1	26.01.2020	1	Pavan	Rack and Pinion
13	SliderV4	27.01.2020	1	Pavan	Rack and Pinion
14	gearv2	27.01.2020	1	Pavan	Rack and Pinion
15	Gear (timing)	5.02.2020	1	Pavan	Belt
16	Gear (timing-cover)	5.02.2020	2	Pavan	Belt
17	pully(timing)	5.02.2020	1	Pavan	Belt
18	Gear (timing)(v2)(5.5mm)	6.02.2020	1	Pavan	Belt
19	Gear (timing)(v2)(5mm)	6.02.2020	1	Pavan	Belt
20	Gear (timing-cover)(v2)	6.02.2020	3	Pavan	Belt
21	pully(timing)(v2)	6.02.2020	1	Pavan	Belt
22	ummut_at	7.02.2020	1	Pavan	Rack and Pinion
23	issa_knife	7.02.2020	1	Pavan	Rack and Pinion
24	cover(pull)	19.02.2020	1	Pavan	Belt
25	cover(gear)	19.02.2020	1	Pavan	Belt
26	Base	19.02.2020	1	Pavan	Belt
27	rod	21.02.2020	1	Pavan	Belt
28	slots	21.02.2020	2	Pavan	Belt
29	gearv4	21.02.2020	2	Pavan	Belt
30	cover v4	21.02.2020	1	Pavan	Belt
31	gear-cover v4	21.02.2020	1	Pavan	Belt
32	rod4	25.02.2020	20	Pavan	Belt
33	gearv4	25.02.2020	20	Pavan	Belt
34	gearcover4	25.02.2020	20	Pavan	Belt
35	gearcover4	27.02.2020	10	Pavan	Belt
36	gearv4	27.02.2020	10	Pavan	Belt
37	rod4	4.03.2020	25	Pavan	Belt
38	Dome H-design	4.03.2020	1	Pavan	Belt
39	Dome H-design	5.03.2020	25	Pavan	Belt
40	gear-cover v4	6.03.2020	26	Pavan	Belt
41	gearv4	6.03.2020	26	Pavan	Belt

Appendix Figure 19 General 3D Printing from the EE2 Labs (Ground Floor)

Final Gantt Chart



Appendix Figure 20 Final Gantt Chart