**Introduction**

Graphical calculators and software are widely used in advanced mathematics. They are an easy way to visualise and analyse graphs and are useful in a classroom setting as well as to verify solutions to problems. Free digital softwares such as Desmos and Geogebra are used extensively, but they work best on computers, and it is not desirable to have them on phones either in a lesson. Physical graphical calculators are therefore used, created by companies such as Casio and Texas Instruments.

However, these physical calculators are very expensive. The Casio fx-CG50, one of the leading devices in the field, costs £140 from the manufacturer [1]. These calculators are very good, with a multitude of features, but a lot of these features are very rarely used, adding to the price point unnecessarily. Additionally, these calculators function as scientific calculators as well, but people that own them generally also own a scientific calculator anyway that they use mainly for calculations.

I aim to create a physical calculator that is only for plotting graphs. This will be less costly to build and produce, and I will be able to tailor it to the end user. The hardware will be specialised, having a screen with a higher resolution than the CG50, and I will 3D print parts specifically for ergonomics.

**Features**

Through all my research, I have gathered many different ideas for the project, so I am creating a list here of the main features that I will add, both hardware and software.

* Explicitly defined graphs
  + Polynomials
  + Radicals
  + Exponentials
  + Reciprocals
  + Logarithmic graphs
  + Trigonometric graphs (sin, cos, tan and reciprocals)
  + Parametric graphs
  + Differential equations
  + Indefinite integrals
  + Combinations of the above
  + x = f(y) graphs
* Implicitly defined graphs
  + Circles
  + Ellipses
  + Parabolas
  + Horizontal parabolas
  + Hyperbolas
* Multiple graphs simultaneously
* Graph manipulation
  + Pan
  + Zoom
  + View window
    - Axis bounds
    - Axis scaling
  + Graph colouring
* Buttons
  + Golden button
  + Degrees-radians rocker switch toggle
  + Zoom momentary toggle switch
* Other features
  + Auto power-off

**End user identification**

The end user will be students that study mathematics or further mathematics at A Level, as well as teachers of the two subjects. The teachers and students have a high ability in mathematics and thus will be able to understand the concepts behind the graphical calculator, and how to use it. It will be helpful in lessons for quickly plotting graphs, giving an idea of roots and intersections, which will help to solve equations. The calculator will be small and portable, allowing anyone to use it quickly when needed, for example, in lessons.

**Requirements gathering**

To compile user requirements to inform the design stage of the project, I created a Microsoft Form, which I sent to A-Level maths students and teachers. I also conducted an interview with one of my further maths teachers to further gather requirements. I used the opportunity sampling method of sampling, which was acceptable for my requirements as I just needed any sort of feedback rather than a representative sample of students.

The form received 13 responses over 9 days, 2 of which were from teachers. It was split into 3 parts - firstly the background information, then the hardware requirements, and finally the software requirements. I will go into further detail about each of these sections below.

In the background information section, I firstly asked people which model of calculator they mainly use, with the available answers being the Casio FX-991EX (widely considered to be the preferred A-level non-graphical calculator model), the Casio FX-991CW (the newer version of the prior model, generally disliked), the Casio FX-CG50 (the most common graphical calculator model), and another model of scientific/graphical calculator. Some of my respondents own and use the CG50, but no-one said that they used it as their main calculator. The EX just beat the CW, as can be seen below. This shows that people mainly use scientific calculators, and graphical calculators are only considered for specialised use cases.

A screenshot of a computer

AI-generated content may be incorrect.

Next, I asked people how often they used some form of graphing software - either a physical calculator or an online tool such as Desmos. The majority of people said that they used it weekly, showing that people interacted with tools such as these regularly. The full results can be seen below.

A screen shot of a computer

AI-generated content may be incorrect.

I then asked people which graphs would be most useful to plot using software. I allowed multiple selections from a list of graph types that I decided to attempt to implement. The majority of people wanted to plot trigonometric graphs, followed closely by parametric graphs, exponentials, and logarithmic graphs. On the other hand, people didn’t seem to be as interested in plotting radicals or derivatives. These results helped inform me on which features to prioritize when I begin the software development process.

**A graph with colorful bars

AI-generated content may be incorrect.**

Moving onto the hardware section, I firstly asked respondents what they liked about their current calculator’s hardware. Lots of people talked about the buttons, which have good ergonomics and aesthetics according to the sampled population. People also commented on the differences between the EX and CW, noting that the CW was harder to navigate.

A screenshot of a calculator

AI-generated content may be incorrect.

A close-up of a cloud

AI-generated content may be incorrect.

I next asked people which hardware features they would like to improve. Users of the new Casio model (CW) mentioned that the button quality could be better, which indicates that I should base my buttons primarily off the EX model. They also talked about how it had some redundant features, and that some buttons present in the EX were not present in the CW, instead having to go into a menu to access the features, which people did not like.

A close up of a button

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a phone

AI-generated content may be incorrect.

I then asked people to rank a selection of hardware features in order of perceived importance for a calculator. Most people were concerned primarily about button layout, and then the specifications of the screen, and were not so interested in the weight and overall size of the calculator. This shows that most people value function over form, preferring a device that is easy to use over something perhaps more aesthetically pleasing.

A screenshot of a computer

AI-generated content may be incorrect.

**A screenshot of a computer

AI-generated content may be incorrect.**

To close the hardware section, I asked people for any other comments, which were mostly not noteworthy apart from a request to add a solar panel and for a large screen. I will be designing the hardware using a regular cell, but if I have time I should be able to implement a solar panel instead. I have decided on a screen with a 3.5 inch display, which should be small enough for a pocket-sized calculator but large enough to show a good amount of detail in the graphs.

I then asked respondents about the software of their calculator, firstly asking which features they found useful. The main features of interest were the range of modes, specifically for statistics, matrices, and solving polynomials/simultaneous equations. As my calculator will be purely graphical, it will not have a set of different modes, but the main takeaway from this is that people liked the ease of access of different features.

A screenshot of a white box

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A close-up of a white background

AI-generated content may be incorrect.

I next asked people what they disliked about the software of their current calculator. The majority of responses were related to ease of use, such as the number of button presses to activate certain features.

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A close-up of a button

AI-generated content may be incorrect.**

Next, I again asked respondents to rank a set of software features by perceived importance. People generally wanted some sort of intersection or root highlighting. This is problematic because in order to get an accurate root it requires solving the equation. As it is extremely tricky to implement a solver for a black-box equation symbolically, I would have to use numerical methods, which are not guaranteed to find all the roots. For these reasons, I probably will not be implementing this feature, but it should be clear given the large screen size where graphs cross the axes. People also wanted clear axis scaling and a pan/move feature for the graphs. They were less interested in custom colours or saving graphs, again showing that they generally valued functionality over aesthetics.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

I then asked whether people would rather have a touchscreen or buttons, and all bar one person responded with buttons. This question may have been misconstrued as whether all the buttons would be replaced or not - the intention was that just the arrow buttons for pan/zoom etc. would be replaced with touchscreen. However, people clearly preferred buttons, so I will be sticking with them. The screen I am looking at does have touchscreen capabilities though, so these could potentially be exploited later in the project to add extra features.

Finally, I asked if anyone had any other comments on the software. The only noteworthy one was a request to add plotting of implicitly defined functions. This poses a problem to implement as implicit functions have to be calculated numerically, which is costly in terms of processing power, and also tricky to implement [2]. For these reasons, I do not plan to add this as a feature, but if I am able to complete all the other features, I will consider adding them as there are algorithms that exist to plot them.

The general sentiments throughout the survey were that people cared more about functionality than aesthetics, and that ease of use was a major concern for people, manifesting generally in button layout and number of button presses required to achieve certain functionalities.

I also talked informally to some people about it, and as a result of this, have decided to add a physical rocker switch to convert from degrees to radians for ease of use and satisfaction, as well as make some buttons gold in appearance.

I then carried out an in-person interview with one of my further maths teachers, asking him generally the same questions that were given in the survey. He said that he did not care so much about plotting graphs of radicals, derivatives, and indefinite integrals, and that features to move graphs around would be useful. These are generally aligned with the findings from the survey.

**Research into current systems**

For my research, I have 4 main current systems to look at: the Casio CG50 graphing calculator, the Casio fx-9750GII, the Desmos graphical software, and the Geogebra graphical software. The CG50 is one of the most common graphical calculators, and I have borrowed the fx-9750GII from my teacher so I can analyse it in greater detail.

Firstly, I will look at the Casio CG50 [1]. This is a graphical calculator that also has features of a scientific calculator. As my project is only to create a device for graph plotting, I do not need to look at the scientific features of the calculator.

The main feature of this calculator seems to be its ability to plot 3D graphs, specifically spheres, cylinders, planes and lines. My calculator will not implement 3D rendering as the algorithms behind it are quite complex, and the majority of people I have spoken to do not use this feature and are not too interested in it. Casio’s calculator can plot multiple of these simultaneously, as well as being able to express them in terms of an equation, vector or points. An interesting feature is the different viewing methods - there are zoom, rotate, cross section, and axis views. On my calculator, I will implement a zoom function, so I will later look at how it is implemented on the Casio calculators. Graph rotation is not useful for 2D graphs, nor are the other two viewing features.

There are a few basic functions that are implemented on the calculator that will be useful for my project. One of these is the ability to change between degrees and radians, which I will do with a physical rocker switch. I will be including trigonometric functions and their inverses, as well as power functions. I will allow fractions as an input method, which will automatically simplify, but also decimals will be supported. I will not be implementing any statistics or matrix functions.

In terms of graphing software, the CG50 implements quite a few features. It allows graphing in terms of either polar or rectangular coordinates, integration graphing, parametric graphing, inequality graphing, multiple graphs at the same time, conic section graphing and dynamic graphing, among others. It also has features for zoom, with a few different implementations. It allows solving with roots and turning points and sketching normal and tangents to a graph at a point.

I will be implementing some of these features, which I will talk about in more detail in the case study on the Casio fx-9750GII. I will not be solving with roots and turning points, or sketching normal and tangents, as that is quite computationally intensive and the algorithms for it are very complicated to implement.

The calculator also has a few problems. The lines can only be graphed in 5 different colours even though the screen can display many more colours, which is something I will fix in my implementation [3]. It uses 4 AAA batteries rather than a rechargeable battery, and this also means that it is quite bulky and thick. I will try and source a rechargeable battery for my project for ease of use. It is purportedly quite complicated to use and requires extensive reading of the manual - I will attempt to make my calculator intuitive to use.

The hardware of this calculator is better than the fx-9750GII, which I will look at next. It uses 4 AAA batteries, giving an approximate battery life of 140 hours. It also has an auto power off feature, something which I will attempt to add to my implementation. The display is 216x384 pixels; mine will be 320x480 [4] which should allow for greater detail in the display. The screen is a backlit LCD, with colour display - the same as mine will be. Interestingly, there is an option to connect the calculator to a PC, to transfer data to and from it - I am not sure how this will be useful in a purely graphical calculator though, so will not be implementing it. It has all the buttons of a typical scientific calculator, as well as 6 function buttons, which seem less intuitive to use. I will be trying to keep my button layout fairly similar so that my calculator will be easy to use and not confusing.

Below is an image of the CG50 and the display screen for plotting graphs: [5] [6]



A screenshot of a computer

AI-generated content may be incorrect.

I am also looking into the Casio fx-9750GII [7], an older graphical calculator, as my teacher has lent me one to have a look at the implementations of functions on it. This calculator does a few different graph types, as well as general scientific calculations.

I am only looking at the graphical features on this calculator, as these are pertinent to my project. There are 3 graphing modes - conic, dynamic, and graph. The conic menu is specifically for graphing parabolas, circles, ellipses and hyperbolas, having a few presets with small icons that show what they do. The user can specify coefficients for each of these presets to graph the corresponding shape. This could be useful for my project, as I do not intend to add specific support for implicitly defined equations, but a special mode for certain implicit equations such as these could be good to implement. The dynamic graph mode allows different versions of the same graph to be drawn simultaneously by changing certain coefficients of the function. This does not seem particularly useful, so I will not be including an implementation of it.

The graphing function is the most useful, and allows for polar, parametric, x= and inequalities to be graphed. I will not be implementing polar graphs, but I am implementing explicit parametrics, and I will also allow x= graphs. Inequalities are not needed as the only change is to shade the area on one side of the graph, which will be computationally intensive and also not very useful. The graphing mode also allows multiple graphs to be plotted at the same time, which is something I will also include in mine.

The calculator has some features for graph manipulation that are quite useful, that I will be implementing in my calculator. These are the view window and zoom features. The view window allows the user to set a minimum and maximum x and y value as well as a scale for these, and to store these values in presets. There is a preset for trigonometric graphs defined in degrees, as well as an initial standard for how the graphs are scaled and zoomed. The zoom feature allows zooming in and out based on where the cursor on the graph is located. In my calculator, I will zoom based on the current graph centre, as I will not be implementing a cursor. I will also implement a view window feature so that users can explicitly define the window rather than zooming and panning, and so that the axes can be scaled.

This calculator however has a few problems with the implementation which I seek to fix in my project. Firstly, the screen resolution is quite low, so the screen is very pixelated, and so graphs look blocky. I intend to use a higher resolution screen to mitigate this issue. Graphs also take a little while to plot, which I will try and fix with my algorithm for interpolation. The view window feature is in a menu which is fine as it will not be used often, however the zoom feature is also, which is quite important. This makes it harder to use as more buttons are needed to be pressed each time. I intend to use a physical momentary rocker switch for the zoom feature, where the user can hold to zoom and release to stop, which should fix this issue. Physical switches also have more satisfying feedback than buttons, which should therefore make the calculator nicer to use.

The hardware specifications [8] are fairly low for this calculator, being from 2009. It uses the SuperH 3 CPU [9], a 32-bit reduced instruction set architecture, developed by Hitachi. It is mainly used for embedded systems, such as calculators. I will be using a Raspberry Pi Zero 2 W, which uses the ARM Cortex-A53 CPU [10], having a 64-bit RISC architecture. These CPUs are optimized for low cost and energy efficiency, and used a lot in microcontrollers, so should be suitable for my purposes. It has 64 KB of RAM (an eighth of the RPI Zero 2 W), and the screen is 128x64 pixels. My screen is going to be 320x480 pixels, which will look a lot better with smoother graphs. The screen is monochrome, whereas mine will be able to utilize colour to display multiple graphs simultaneously. It weighs 205g, which is somewhat heavy but still acceptable, and can be connected serially or via USB. It uses 4 AAA batteries, which brings into question whether a solar panel will provide enough power if I try and implement one in my calculator. The display is an LCD display, the same as the one I will use. It has the basic numeral and operation keys of a normal calculator, with an EXE instead of an equals button, a few scientific functions such as trigonometry, powers, logarithms and exponentials, an arrow key pad, and 6 F buttons. These 6 buttons are used for navigating through the menus, but I have not found them very intuitive to use, so I will try not to include things like that.

Below is an image of the fx-9750GII [11]. It can be seen that the calculator has an older style and aesthetic to it. My calculator’s design will be based on the more modern CG50 rather than this one.

A calculator with a screen

AI-generated content may be incorrect.

Next, I will look at the Desmos [12] and Geogebra [13] graphical softwares. Desmos is probably the most widely used graphing software, and has a web version as well as a mobile app. It is extremely powerful, parsing all types of equation and plotting them on a very nice, clean interface. Geogebra is more widely used in teaching, and offers a wider range of tools, such as geometric constructions. Desmos is easier to use than Geogebra however. Geogebra runs quite slowly compared to Desmos, and has a less neat GUI and is less user-friendly. Both of these softwares are very advanced however. The main differences between them and a graphical calculator are the hardware they run on, and the extra features. An example of the latter is in Desmos, where there is a feature to play audio, with the x value being time and the y value being amplitude. Features like this are not needed on a physical graphing calculator, so I will not be implementing them. The hardware makes tools like Desmos a lot better for some things. It runs on a computer or mobile CPU, which are far more powerful than calculator CPUs, so it can compute things that are a lot more intensive, such as implicit functions. The screen is very high resolution, so it can show an extreme amount of detail, and a mouse allows for pan and zoom features to work a lot easier.

However, a physical calculator also has some advantages. It is more portable, and nicer to use given the tactile feedback - this can also make it faster for some applications. It is useful in a classroom setting, particularly if mobile devices are not allowed. The axis scaling of Desmos and Geogebra is quite useful, so I will include that in my calculator for the view window.

Below are some screenshots of Desmos [12] and Geogebra [13]. We can observe the wide range of features of each of the softwares, and the clean interface, but also how they are more suited for a desktop application.

A screen shot of a graph

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a graph

AI-generated content may be incorrect.

A screenshot of a keyboard

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Modelling**

To create a graphical calculator, the focus will be the software. I will be coding it in Python, relying very little on external libraries (such as NumPy, SciPy etc.). The project needs three general algorithms, as well as some more specific ones for computing the different mathematical functions. The main algorithms needed are explained below.

The aim of the project is to be able to plot graphs of *explicit* (defined in terms of a single variable):

* Polynomials
* Radicals
* Exponentials
* Reciprocals
* Logarithmic graphs
* Trigonometric graphs (sin, cos, tan and reciprocals)
* Parametric graphs
* Differential equations
* Indefinite integrals
* Combinations of the above

Firstly, I need to create a **parser**.

The parser needs to be able to take an input of a human-readable function (with some constraints - all functions will be *explicitly* defined, that is, in terms of a single variable, and correct mathematical syntax is required)

The algorithm I will use will convert the entered function from infix to postfix notation, having pre-processed some operators and implicit multiplication using regular expressions. Infix notation is the regular notation in which calculations are generally written by humans, following the rules of BIDMAS to determine order of operation [14].

Postfix notation, which is also referred to as Reverse Polish notation, is where operators follow the operands, rather than when they precede the operands as in infix notation. Due to this, it allows the expression to be evaluated left to right, rather than having to consider the order of operations. This is done using a stack, which is useful for processing expressions automatically. It also removes the need for parentheses [15].

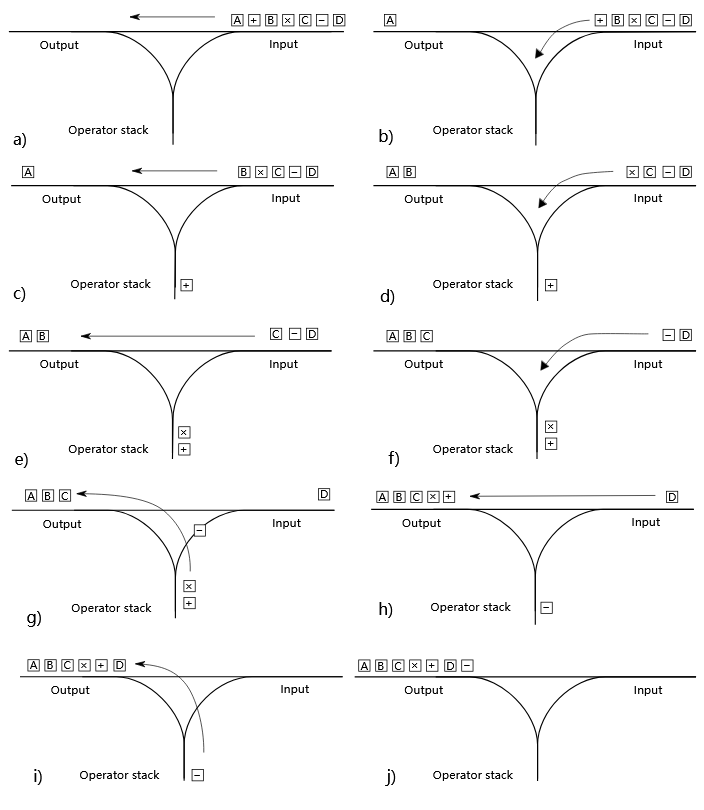
To convert from infix to postfix notation, I will use an algorithm developed by Dijkstra, known as the shunting yard algorithm. The way this algorithm works is that the expression is iterated through from left to right. There is an output queue and an operator stack, where tokens in the expression are added to when encountered, following set rules [16].

Associativity is also a factor in the algorithm. Left-associative operators, such as the multiply operator, are where operations can be grouped from left to right. This is in contrast with right-associative operators, such as exponentiation, where operations are grouped from right to left.

Whenever a number is encountered, it is pushed to an output queue. If a function, such as the sine function, is encountered, it is pushed to the operator stack. If an operator (*x*) is encountered, the algorithm iterates through each operator currently in the stack (*y*). If *y* is not a left parenthesis and has greater precedence than *x* or has the same precedence with *x* being left-associative, then *y* is popped from the stack to the output. When *y* is not popped, the iteration stops, and *x* is then pushed onto the stack.

There are more detailed rules based on functions that take multiple inputs, and parentheses that comprise functions, but these are not important to the base parsing function that I will implement. Once there are no tokens left to iterate through, all the operators on the stack are popped to the output queue.

Below is a graphical illustration of the shunting yard algorithm for basic operators: [17]



We can see how the tokens are processed from left to right, and the operators are pushed and popped from the stack as needed. In this example, the letters represent numbers, so are pushed to the output when encountered. The other operators are treated according to the BIDMAS order of operations. An explanation of this example is as follows:

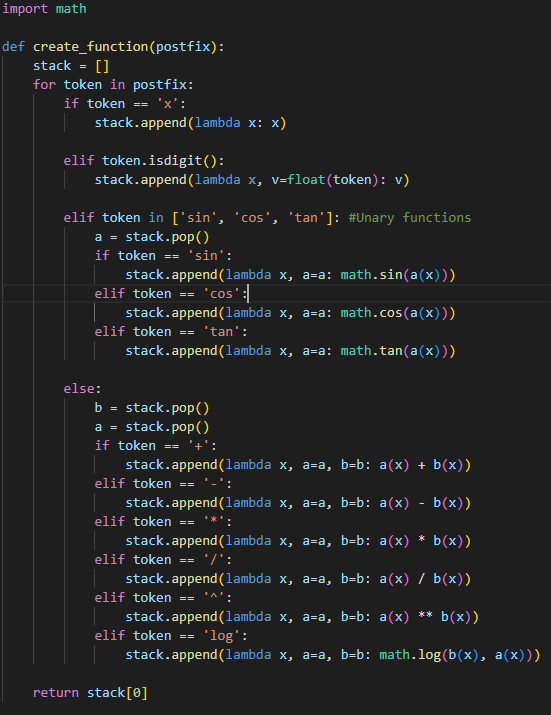
* 1. The number A is pushed to the output.
  2. The operator + is pushed to the stack.
  3. The number B is pushed to the output.
  4. The operator x is pushed to the stack. As the + operator has lower precedence, it is not popped from the stack.
  5. The number C is pushed to the output.
  6. The operator - is pushed to the stack. The x operator has a higher precedence so is popped to the output. The + operator has the same precedence and the - operator is left-associative, so it too is popped from the stack.
  7. The number D is pushed to the output.
  8. The operator - is popped from the stack to the output as there are no more tokens to process.

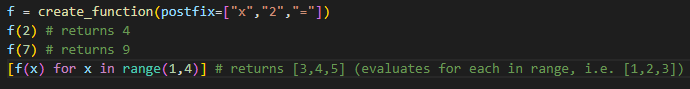
I also need to do some pre-processing as mentioned above to convert the inputted expression into a form suitable for processing with this algorithm. Firstly, I will need to write out operators explicitly. For some operators, such as exponentiation, it will only render a superscript on the user interface, but in the backend, it will use the carat operator, so I will not need to process it further. The main operation to process is implicit multiplication, that is, AB as opposed to A\*B or AxB. I will do this using regular expressions (regex). I can search for valid combinations of two letters next to each other, or a number to the immediate left of a letter and appropriately insert the multiplication operator.

Below is a small prototype in Python of the algorithm to generate values from the function in postfix notation. I am using lambda calculus to combine each term in the function, thus creating a new function in terms of x, so that an arbitrary x value can be put in, and the corresponding y value read out.

I read through each token in the postfix notation, and create lambda functions that output either x for x (this allows calling the function with any value and the same value being output), or a specific constant (such as in the case x + 2, the 2 is always constant so the function will always output 2). If an operator is encountered, I create a new lambda function by popping the previous value(s) from the stack and using them as parameters in the new function. By nesting lambda functions in this way, Python will recursively evaluate the function at any input value.

The return value is the first item in the stack - a lambda function, defined with one parameter, x, that calls all other operators within the function, thus outputting the y value for a specific x value. I would then be able to call it for each number in a range as well as specific values, as can be seen below.





Next steps will be to add support for negative numbers and decimals, which can be done using regexes, and I want to have lists of functions (e.g. unary functions) in a global constants file for readability and reusability.

Next, I will need to create a **plotter**.

The plotter algorithm will firstly define a set interval for the function to be evaluated across, and then it will evaluate the function at specific points. It will then use a method of interpolation to join up the curve between the evaluated points, hopefully producing a smooth render, thus converting the vector graphics of the curve into a bitmap format for pixel display in Pygame.

I have done some research on how best to define the interval. I was initially thinking to calculate a number of turning points of the function, and to define the interval so all the turning points lie within it, and also thought to scale the axes appropriately, for example, the sine function in degrees would need to have a larger scale on the x axis. However, after my research into current systems, which I talked about earlier, I realised that this was unnecessary. The majority of graphical calculators, both online and physical devices, have fixed axes, that every graph is plotted to, with features for users to pan and zoom the graph themselves. As I am intending to include these features anyway, it would be a waste of time to try and create an elaborate algorithm to automatically scale the graphs, given the number of edge cases I would have to consider.

As I will discuss later in the hardware section, I have decided on a screen that is 320x480 pixels. I will therefore evaluate points appropriately, at a frequency of x values that will produce a smooth curve through interpolation. I will test out various methods of interpolation with different amounts of values until I find a sweet spot that minimizes computational intensity whilst maximising the smoothness of the function.

I have researched a few different interpolation methods [18] and decided that I will firstly try linear interpolation [19]. This is where each point is joined to the next by a straight-line segment. This method draws a line between the two points, by finding the gradient between the points and then drawing the line. Pygame has a line draw function, which takes a start and end point, so I will use this for my implementation.

There are other types of interpolation [20], such as polynomial [21] and spline [22] interpolation. These will produce a smoother curve, but at the cost of processing power, and will take longer to implement. For this reason, I am intending to compute more points on the curve and then use linear interpolation, rather than computing less points but using a different type of interpolation. Hopefully, this should produce an adequately smooth curve [23].

Next, I will need a **translator**.

This will be a set of functions that can pan and zoom the graph on the screen. I will have a set of 4 arrow buttons on the calculator, as well as either two zoom buttons or a two-way rocker switch with the same function. I will decide which one to use based on the available space on the calculator. There will also be some sort of home button, which will reset the pan and zoom to the defaults. I will describe these buttons in more detail in the hardware section.

Whenever one of the arrow buttons is pressed, the boundaries of the screen will change a specified amount, based on how long it is pressed for. When the button is released, I will recalculate the plotting for the graph with the new bounds. This will help performance as I am only recalculating when the button is released, not while it is being pressed. The zoom button will scale both axes proportionally to the length of the button press, keeping the centre of the screen at the same point.

The maths behind this should be fairly simple - just scalar multiplication or addition. The main purpose of the function is to ensure an ease of use for the user, rather than to add any more information.

For the graphs themselves, simply calculating for a given value of x with the postfix notation should work, as they will all be defined explicitly. If an error occurs, such as an attempted division by zero, then I can mark that point as an asymptote to the curve. A potential hazard could be if two points are calculated that straddle an asymptote, so the linear interpolation joins them up in what appears to be a vertical line - this however should be fine for the purposes of graphical display as it is clear that an asymptote exists there.

Parametric graphs, differential equations, and indefinite integrals will each require a different algorithm to plot however, as they are not given in explicit Cartesian form. I will use numerical methods to plot them rather than attempting to manipulate the given expression, as it would require writing an entirely new method of parsing expressions that also enables manipulation - something that would be both extremely tricky to do and that has already been done (by the Sympy library).

Firstly, I will create an algorithm to deal with **parametric graphs**.

The parametric graphs will be defined explicitly as functions of t, that is, x(t) and y(t). An example is:

As we are working within set boundaries of the graph, the algorithm would firstly need to find the t value(s) that corresponds to each x value and then calculate the resulting y value(s) from it. I had come up with an idea about how to do this by searching for the roots of the equations formed [24] using root bisection [25], however, this is computationally intensive, will not work for every set of equations, and does not account for multiple roots.

Instead, I have come up with another method that I believe should work fairly well. I will firstly do a coarse sweep across a wide, arbitrary range (say -1000 to 1000 with intervals of 1). These values are for the parameter t, and whenever they give a point that lies within the bounds, I will store it in a list. I will then use recursion to search at finer intervals around all the t values within the bounds, and so on, probably with a recursion depth of 4 but depending on performance. I am not sure how well this method will work, but I will not know until I implement and test it, so for now, I will be taking this approach.

Next, I will use numerical methods to plot **differential equations**, for example:

I am intending to use finite difference approximations [26] of quite a low step size. I will calculate the gradient of the secant line between two points very close together on the curve (although the step size cannot be too small due to floating point precision errors). Firstly, I will need a list of x values within the boundaries, as with the other graphs. I will then evaluate the function at that x value, as well as the function at the value of x plus the small h value. The difference of these two y values, divided by h, will give an approximation for the slope at that point, which can be taken as the y value to plot on the screen, thus plotting a derivative. I will experiment with different difference methods - forward, backwards and central difference. These are essentially when h is either above or below x, or x is taken as the midpoint of the secant line. Each has a different error margin and is more suitable in certain situations so I may end up creating an algorithm to apply different difference types in different situations.



The above image [27] shows the different types of method graphically. We can see how a small interval is taken, and the gradient of the secant line is used to approximate the derivative at that point. The central difference method is preferred by most people as it is second order in h, that is, the error margin for the method is O(h²), as opposed to O(h) for the other two methods [28]. Due to this, I will implement the central difference method first and only try the other two if the resulting graph is not accurate enough.

I also need to use numerical methods to plot **indefinite integrals**, for example:

I will be using the trapezium rule [29], which numerically approximates the area under the curve. For each interval, the area will be cumulative and can be plotted as the y value of the corresponding x value. This will shift the graph down by the amount of area that is under the curve to the left of the graph, as there is no way of representing an infinite area. It will thus plot what is known as the antiderivative.

As with the other graphs, I will firstly need a discretized list of points spanning the domain. I will then apply the trapezium rule cumulatively, like so for each interval between the points:

This will give a list of areas to plot as the indefinite integral. Below shows a graphical illustration of how the trapezium rule works, and how the accuracy increases as the number of points in the domain does [30]. It can clearly be seen how the trapezia either overestimate or underestimate the curve, depending on the sign of the derivative at that point, and thus how with smaller trapezia the error margin is minimized. The trade-off is performance again - I will have to choose a number of strips that balances the compute time with accuracy.

A graph of a trap

AI-generated content may be incorrect.

A graph with red lines and numbers

AI-generated content may be incorrect.

A graph with red lines and numbers

AI-generated content may be incorrect.

A graph of a trap

AI-generated content may be incorrect.

A graph of a trap

AI-generated content may be incorrect.

A graph of a trap

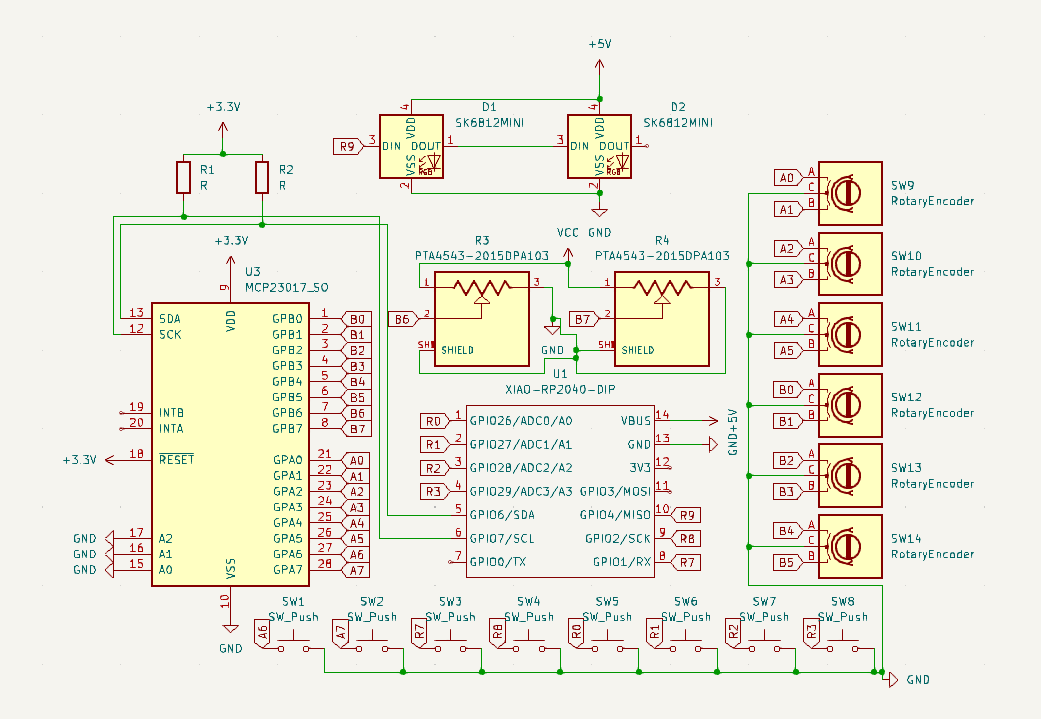
AI-generated content may be incorrect.

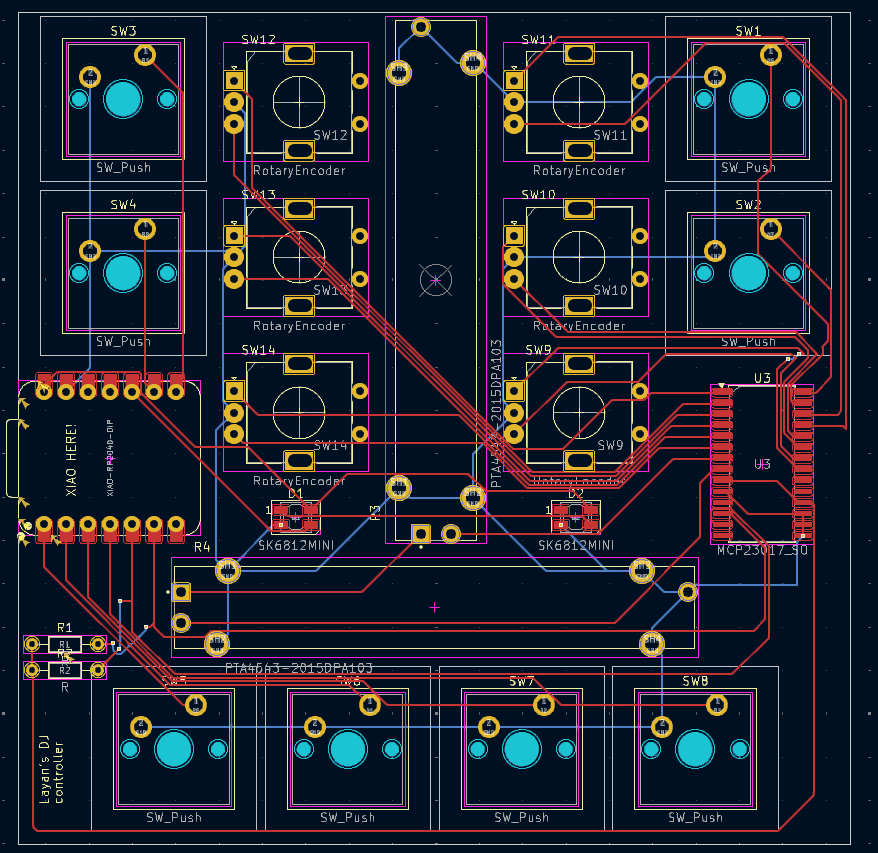
**SMART objectives**

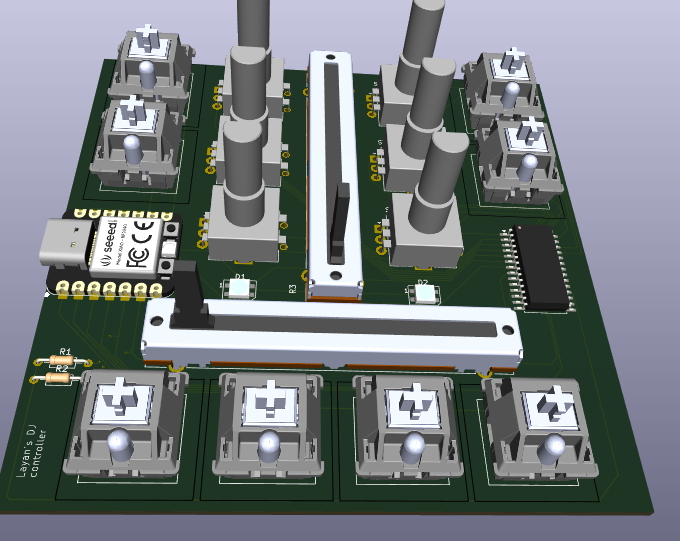
1. Create a parser for arbitrary polynomials
   1. Pre-processing
      1. Handle implicit multiplication (regex)
      2. Write powers with exponentiation operator
   2. Conversion from infix to postfix notation
2. Generate a list of points for the graph
   1. Define interval
   2. Decide frequency of points
   3. Calculate x values through postfix stack
3. Create a mock-up display for testing purposes
   1. Create Pygame display compatible with screen (320x480px)
   2. Draw axes with defined interval
   3. Plot points
   4. Interpolate between points
4. Add translation functions
   1. Add zoom function
      1. Scale axes by factor centred on middle of screen
      2. Recalculate points with new interval
      3. Replot and interpolate points
   2. Add move function
      1. Change interval and recalculate points
      2. Add reset button
5. Extend parser to other Cartesian equations
   1. Radical equations
      1. Parse roots with exponentiation operator
   2. Exponential equations
      1. Define constants (e, pi)
   3. Reciprocal graphs
      1. Parse fractions as division operator
      2. Handle asymptotes
         1. Draw dashed line OR approximate continuously
   4. Logarithmic equations
      1. Parse logarithms as single character, allow for bases
   5. Trigonometric equations
      1. Parse main functions
         1. Handle tan asymptote
      2. Parse reciprocal functions
         1. Handle asymptotes
6. Extend parser to parametric equations
   1. Allow for inputs of explicit y(t) and x(t)
   2. Determine bounds for t for given interval
   3. Parse each as Cartesian equations
7. Extend parser to calculus
   1. Differentiation
      1. Calculate x values for interval
      2. Approximate central difference
      3. Handle asymptotes (high gradient)
   2. Integration
      1. Calculate x values for interval
      2. Use trapezium rule to get area of slices
      3. Divide by strip width to plot y values
      4. Handle asymptotes (high area)
8. Add support for multiple equations
   1. Add multiple input option with colour key
   2. Plot both graphs (different colours)
9. Testing
   1. Check basic test cases for all types of equation
   2. Check test cases for multiple types combined
   3. Test cases where calculus doesn’t work well
10. Design physical calculator
    1. Make list of buttons from what I was able to implement
    2. Arrange buttons in a layout
    3. Design electronics
       1. Create schematic with battery connected to raspberry pi
       2. Connect buttons to raspberry pi in schematic
       3. Connect rotary encoder for zoom OR use touchscreen
          1. Order screen to test code
       4. Arrange components on PCB
       5. Add solar panel
          1. Connect to solar power manager and rechargeable battery
       6. Route and wire PCB
    4. CAD design
       1. Design case
       2. Design buttons
11. Build physical calculator
    1. Create BOM
    2. Order PCB
    3. Order other electronics
    4. 3D print CAD parts
    5. Solder remaining components to PCB
    6. Run program on raspberry pi
       1. Check power supply
       2. Load and run on boot
       3. Scale Pygame for the screen
       4. Change code for touchscreen if required
    7. Assemble the calculator

**Hardware**

One of the most important components of the calculator is the PCB (printed circuit board) [31]. This is a board with layers of wiring that will connect all the components together neatly and efficiently. The board will be shaped like the calculator, and the switches for the buttons will be mounted to it directly. It will have connectors for the screen, CPU, and battery as well. To create it, I will be using the software suite KiCad [32]. KiCad allows you to create schematics of the wiring, and then design the PCB layout, giving a 3D render as well. I have used this software previously to design PCBs, and it is very good and easy to use. To print the PCB, I will use an online company, likely either JLCPCB, or PCBWay. If possible, I will use the PCBA option, whereby components are soldered to the PCB by the company rather than having to do it manually. Below are some screenshots of the KiCad software - firstly the schematic design [33], then PCB editor [34], and then 3D render [35]. For all the screenshots and examples of the hardware, I will use a project that I have created, which is a custom-built DJ controller [36]. Below is also a photo I took myself of the printed PCB in a custom-designed 3D-printed case [37].



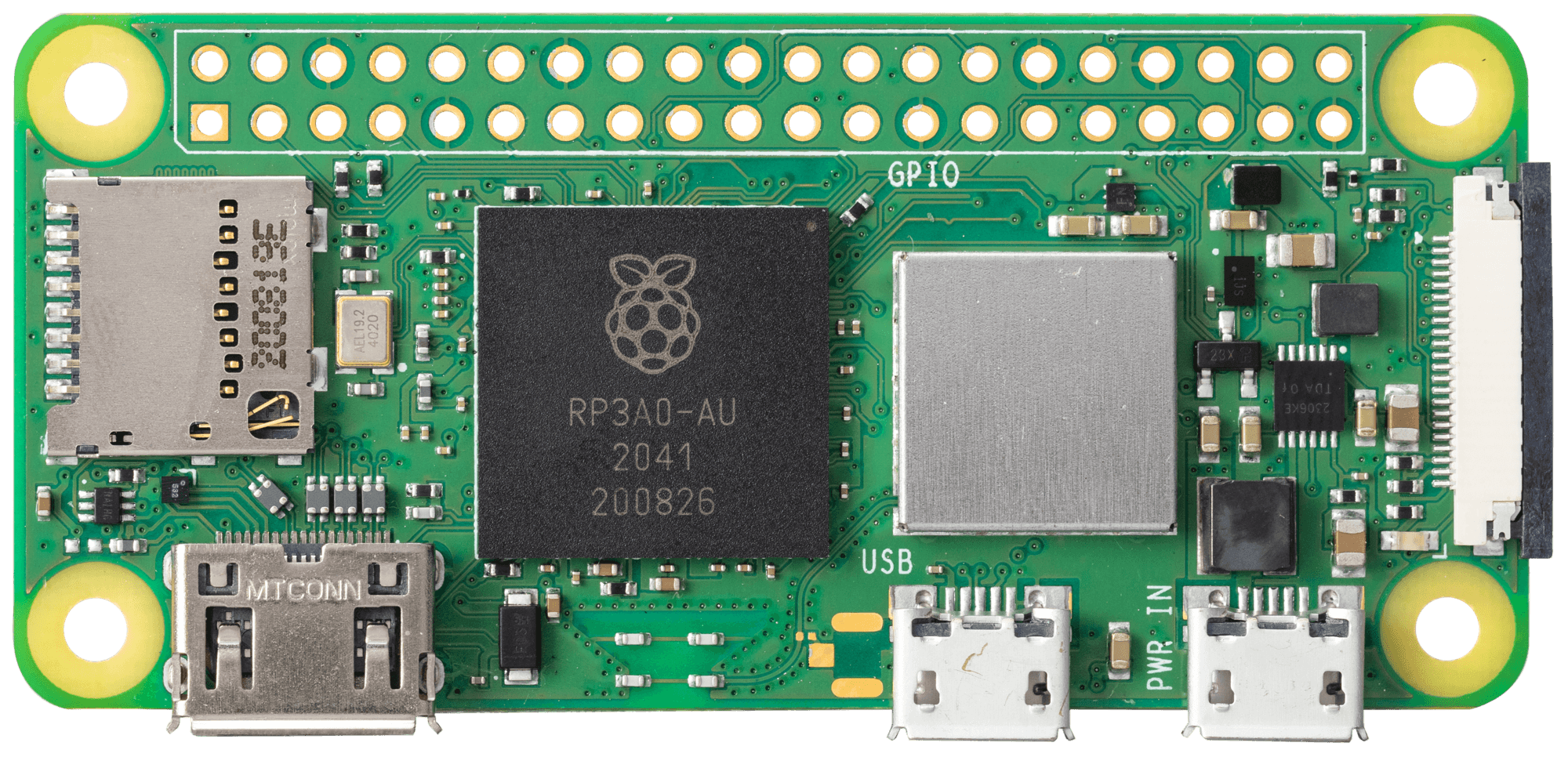






The components of the calculator are the CPU, screen, buttons, and battery.

Firstly, the CPU. The CPU I will be using is the Raspberry Pi Zero 2 W [38]. It is 65mm x 30mm, so fairly small and will fit nicely in the calculator. It has 512 MB of RAM, and a clock speed of 1GHz, so it will not be able to handle more intensive computations. It has WiFi and Bluetooth connectivity, although I am not planning to use them, as well as a USB 2.0 port, a microSD card slot, a mini HDMI port, and a 40 pin I/O header [39]. I will have to install an operating system on it, probably Raspbian, to run Python code. Below is a picture of the microcontroller [40]:

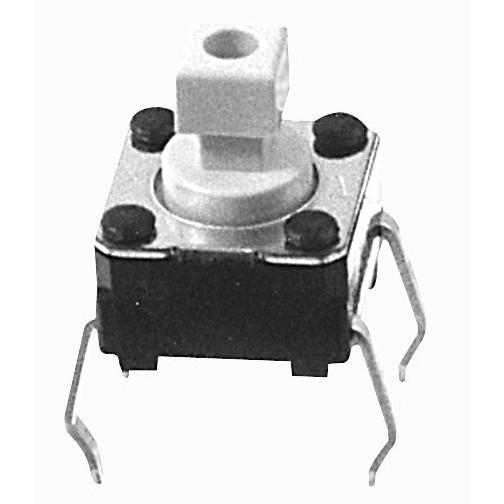


Next, the screen. The screen I have decided to use is a 3.5 inch LCD screen, intended for a display for a Raspberry Pi [4]. It has a 320x480 pixel resolution, and also features resistive touch control, though I probably will not be utilizing that feature of it. The resolution is higher than the Casio fx-CG50, one of the leading graphical calculators, so it should be more than enough for my purposes. The screen is £24 directly from the manufacturer, and below is a picture of the screen running the Raspbian OS [41]. The screen connects via HDMI, and is powered by a micro USB connection [42].

A device connected to a computer

AI-generated content may be incorrect.

Now, I will talk about the buttons. I have decided to use tactile switches with attached shafts, which, as mentioned, will be soldered directly to the PCB. I plan on using the Omron B3F-1050 switches [43], which I can get from DigiKey, at around 12.5p each for the number I will require. They have dimensions of 6 x 6 mm with a height of 7.3 mm [44], which should be suitable for my purposes. I can buy keycaps for these switches, or 3D print them myself. I found many sites selling caps - there are some round ones on AliExpress for £2.79 per 100 caps [45], and also some square ones for £2.47 per 50 caps [46]. I also found a model that I can 3D print, although this is for the larger 12x12x7.3 mm switches, but I can scale it down [47]. Below is a picture of the switch [48], and some pictures of a keyboard someone made with the same switches [49]:



A close up of a keyboard

AI-generated content may be incorrect.

A computer generated image of a green rectangular object

AI-generated content may be incorrect.

I am planning to use 4 switches for arrow buttons to move the screen around rather than a direction pad, as they will be easier and cheaper to implement. I want to use a rocker toggle switch for switching between degrees and radians, as this is a commonly used feature in calculators but requires going into the menus to find the correct option, which is time consuming and annoying. The switch will be a simple and effective fix to this. I will also use a momentary toggle switch for the zoom functionality - the user will have to hold the switch down in the direction they want to zoom. Below is a picture of a rocker switch that can be mounted to the PCB [50]. This series of switch has many different models for the different types (on, off, momentary) [51].

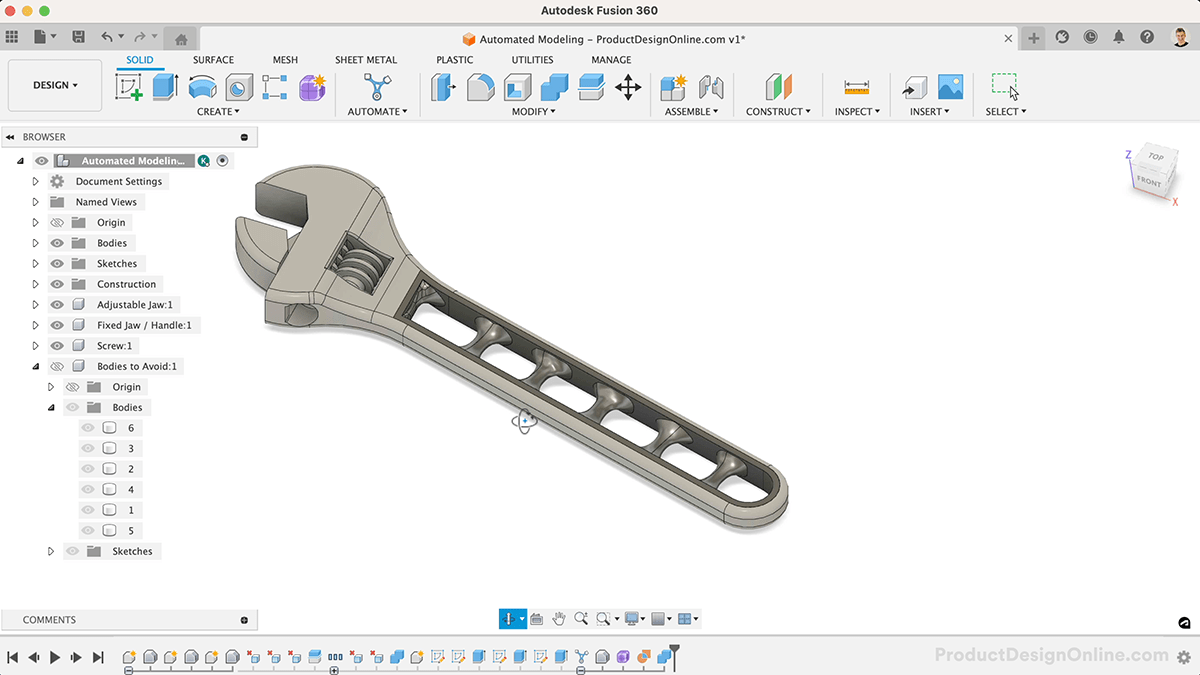


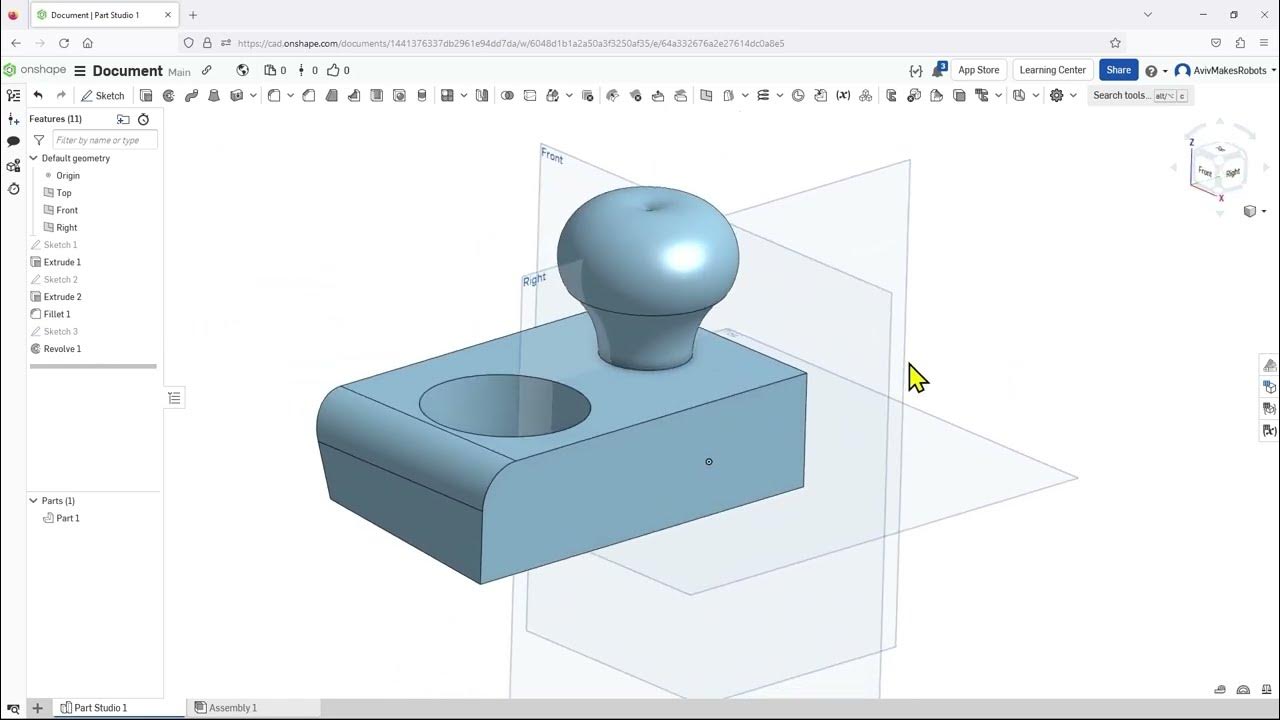
Next, I will talk about the batteries. I thought about using a solar panel initially, but they are expensive, and it may not provide enough power for the Raspberry Pi. I will therefore be using a 5V rechargeable battery. I need something fairly flat so that the case is not too bulky, and it needs to be fairly high power as well so that the calculator can run for quite a while. Most batteries are 3.7V which would need a boost regulator [52], but this will take up more space, so I will use a 5V battery instead. Someone powered a Pi Zero 2 W with a 10,000mAh battery to run a very basic Pygame program, which got over 8 hours of runtime [53]. I have found a 9000mAh battery that is 5V, which hopefully should be enough [54]. There is a picture of it below [55]:

A battery with a circuit board

AI-generated content may be incorrect.

Finally, I need to create a case for the calculator. I will use CAD (computer-aided design) software [56] to do this, and then 3D-print it myself. I will likely use the Fusion360 software [57], or the Onshape software [58], to do this. This will mean I can create a fully customized case that fits the calculator exactly, and I can do a process of refinement as I can just print another one if needed. Below is an example of a simple model in Fusion360 [59], and Onshape [60]. Fusion is a desktop application, which I have already through the education license, whereas Onshape runs in the browser, which is more helpful sometimes but requires an internet connection. Below is also a screenshot of a model I made in Fusion for the case of my DJ controller project, with the 3D render of the PCB shown inside it [61].





A computer generated grey box with many objects

AI-generated content may be incorrect.

As I have mentioned, I own a 3D printer, which I can use to print the case and button caps. The one I own is the Bambu Lab A1 Mini [62], which can print up to 180x180x180 mm^3 [63], which should be sufficient. Below is a picture of the printer [64]:



**Bibliography**

[1] <https://www.casio.com/uk/scientific-calculators/product.FX-CG50/> *Accessed on 19/07/2025*

[2] <https://stackoverflow.com/questions/1131815/how-to-plot-implicit-equations> *Accessed on 19/07/2025*

[3] <https://mathclasscalculator.com/calculators/graphing/casio-fx-cg50-full-review/> *Accessed on 19/07/2025*

[4] <https://osoyoo.store/en-eu/products/osoyoo-lcd-touch-screen-3-5-hdmi-monitor-display-tft-for-raspberry-pi-3-2-model-b-audio-output-with-stylus-pen> *Accessed on 19/07/2025*

[5] <https://www.casio.com/content/dam/casio/product-info/locales/uk/en-gb/calc/product/scientific/F/FX/FXC/fx-CG50/assets/fx-CG50_Seq1.jpg.transform/main-visual-pc/image.jpg> *Accessed on 19/07/2025*

[6] <https://www.casio.com/content/experience-fragments/casio/en-gb/product-info/calculator/product/f/fx/fxc/fx_cg50/fx-cg50/master/_jcr_content/root/container_copy_copy__1602485120/container_copy_copy__2055408557/image_1315325768.casiocoreimg.jpeg/1617335355380/img27.jpeg> *Accessed on 19/07/2025*

[7] <https://education.casio.co.uk/calculator/fx-9750gii/> *Accessed on 19/07/2025*

[8] <https://wiki.planet-casio.com/en/Fx-9750GII> *Accessed on 19/07/2025*

[9] <https://en.wikipedia.org/wiki/SuperH#SH-3_2> *Accessed on 19/07/2025*

[10] <https://en.wikipedia.org/wiki/ARM_Cortex-A53> *Accessed on 20/07/2025*

[11] <https://wiki.planet-casio.com/en/File:Fx-9750gii_xlarge.jpg> *Accessed on 19/07/2025*

[12] <https://www.desmos.com/calculator> *Accessed on 19/07/2025*

[13] <https://www.geogebra.org/graphing?lang=en> *Accessed on 19/07/2025*

[14] <https://en.wikipedia.org/wiki/Infix_notation> *Accessed on 23/06/2025*

[15] <https://en.wikipedia.org/wiki/Reverse_Polish_notation> *Accessed on 30/06/2025*

[16] <https://en.wikipedia.org/wiki/Shunting_yard_algorithm> *Accessed on 30/06/2025*

[17] <https://en.wikipedia.org/wiki/Shunting_yard_algorithm#/media/File:Shunting_yard.svg> *Accessed on 30/06/2025*

[18] <https://en.wikipedia.org/wiki/Interpolation> *Accessed on 01/07/2025*

[19] <https://en.wikipedia.org/wiki/Linear_interpolation> *Accessed on 01/07/2025*

[20] <https://interpolationcalculator.com/types-of-interpolation/> *Accessed on 01/07/2025*

[21] <https://en.wikipedia.org/wiki/Polynomial_interpolation> *Accessed on 01/07/2025*

[22] <https://en.wikipedia.org/wiki/Spline_interpolation> *Accessed on 01/07/2025*

[23] <https://paulbourke.net/miscellaneous/interpolation/> *Accessed on 01/07/2025*

[24] <https://en.wikipedia.org/wiki/Root-finding_algorithm> *Accessed on 02/07/2025*

[25] <https://en.wikipedia.org/wiki/Bisection_method> *Accessed on 02/07/2025*

[26] <https://en.wikipedia.org/wiki/Finite_difference> *Accessed on 02/07/2025*

[27] <https://www.multiphysics.us/FDM.html> *Accessed on 02/07/2025*

[28] <https://math.stackexchange.com/questions/2120946/why-is-central-difference-preferred-over-backward-and-forward-difference-in-conv> *Accessed on 02/07/2025*

[29] <https://en.wikipedia.org/wiki/Trapezoidal_rule> *Accessed on 02/07/2025*

[30] <https://en.wikipedia.org/wiki/Trapezoidal_rule#/media/File:Trapezium2.gif> *Accessed on 02/07/2025*

[31] <https://en.wikipedia.org/wiki/Printed_circuit_board> *Accessed on 20/07/2025*

[32] <https://www.kicad.org/> *Accessed on 20/07/2025*

[33] <https://github.com/LayanJethwa/dj-controller-hackpad/blob/main/assets/schematic.png> *Accessed on 20/07/2025*

[34] <https://github.com/LayanJethwa/dj-controller-hackpad/blob/main/assets/pcb.png> *Accessed on 20/07/2025*

[35] <https://github.com/LayanJethwa/dj-controller-hackpad/blob/main/assets/pcb-model.png> *Accessed on 20/07/2025*

[36] <https://github.com/LayanJethwa/dj-controller-hackpad> *Accessed on 20/07/2025*

[37] *Photo taken by Layan Jethwa, 19/05/2025*

[38] <https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/> *Accessed on 20/07/2025*

[39] <https://datasheets.raspberrypi.com/rpizero2/raspberry-pi-zero-2-w-product-brief.pdf> *Accessed on 20/07/2025*

[40] <https://assets.raspberrypi.com/static/f03a00485ac096c1033ad1c4a530d63b/74ad4/zero2-close-up.png> *Accessed on 20/07/2025*

[41] <https://osoyoo.store/cdn/shop/products/3_ac1dd358-6477-440b-bcb1-63df049d04b8.jpg?v=1566552898&width=1946> *Accessed on 20/07/2025*

[42] <https://osoyoo.com/driver/3.5_HDMI_LCD_User's_manual.pdf> *Accessed on 20/07/2025*

[43] <https://www.digikey.co.uk/en/products/detail/omron-electronics-inc-emc-div/B3F-1050/11889> *Accessed on 20/07/2025*

[44] <https://omronfs.omron.com/en_US/ecb/products/pdf/en-b3f.pdf> *Accessed on 20/07/2025*

[45] <https://www.aliexpress.com/item/1005005572825625.html> *Accessed on 20/07/2025*

[46] <https://www.aliexpress.com/item/1005005639740870.html> *Accessed on 20/07/2025*

[47] <https://www.printables.com/model/236991-tactile-switch-push-button-cap?lang=en> *Accessed on 20/07/2025*

[48] <https://cdn.soselectronic.com/productdata/86/ff/9e2fa8d0/b3f-1052.jpg> *Accessed on 20/07/2025*

[49] <https://hackaday.io/project/186263-finch-thumb-keyboard> *Accessed on 20/07/2025*

[50] <https://www.rjselectronics.com/products/rocker-switches-4a-spdt/> *Accessed on 21/07/2025*

[51] <https://www.rjselectronics.com/wp-content/uploads/4A-Series-Rocker-Switch-RJS-Electronics-Ltd.pdf> *Accessed on 21/07/2025*

[52] <https://forums.raspberrypi.com/viewtopic.php?t=81073> *Accessed on 21/07/2025*

[53] <https://www.reddit.com/r/raspberry_pi/comments/1cgccrv/battery_powering_pi_zero_2/> *Accessed on 21/07/2025*

[54] <http://www.pacbattery.com/rechargeable-lithium-polymer-battery-pack-9000mah-5v-with-usb-port-for-power-bank.html> *Accessed on 21/07/2025*

[55] <http://www.pacbattery.com/wp-content/uploads/2018/02/5V-9000mAh.jpg> *Accessed on 21/07/2025*

[56] <https://en.wikipedia.org/wiki/Computer-aided_design> *Accessed on 21/07/2025*

[57] <https://en.wikipedia.org/wiki/Fusion_360> *Accessed on 21/07/2025*

[58] <https://www.onshape.com/en/> *Accessed on 21/07/2025*

[59] <https://productdesignonline.com/automated-modeling-in-autodesk-fusion-360-what-is-automated-modeling/> *Accessed on 21/07/2025*

[60] <https://www.youtube.com/watch?v=2utLjjkXpIg&ab_channel=AvivMakesRobots> *Accessed on 21/07/2025*

[61] <https://github.com/LayanJethwa/dj-controller-hackpad/blob/main/assets/full-model.png> *Accessed on 21/07/2025*

[62] <https://uk.store.bambulab.com/products/a1-mini> *Accessed on 21/07/2025*

[63] <https://wiki.bambulab.com/en/a1-mini/manual/faq> *Accessed on 21/07/2025*

[64] <https://3dprinteracademy.com/blogs/3d-printers/introducing-the-bambu-lab-a1-mini> *Accessed on 21/07/2025*