Turtlebot Software for Schools Outreach

Cate Fitzpatrick

Department of Computer Science

Aberystwyth University

Wales

September

2025

This thesis is submitted in partial fulfilment of the requirements for the degree of Master of Science

Degree: MSc Advanced Computer Science

Module: Dissertation (CSM9060)

Supervisor: Dr Neal Snooke

Abstract

The abstract stands alone as a very short version of the dissertation.

The abstract should state the scope and principal objectives of the project, describe the methods, summarize the results and state the principal conclusions.

**(Max. 500 words.)**

Declaration of originality

I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the regulations on Unacceptable Academic Practice from the University’s Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
* In submitting this work, I understand and agree to abide by the University’s regulations governing these issues.

**Name:** Cate Fitzpatrick   
**Date:** dd/mm/yy

Consent to share this work

* By including my name below, I hereby agree to this thesis being made available to other students and academic staff of the Department of Computer Science, Aberystwyth University.

**Name:** Cate Fitzpatrick   
**Date:** dd/mm/yy

Acknowledgement

To whoever has the patience to read this :-)

This section is customary, but not obligatory. It makes a brief statement of thanks to those who have helped.

Table of Contents

[Chapter 1. Introduction 1](#_Toc205819992)

[Background 1](#_Toc205819993)

[Project aims and objectives 1](#_Toc205819994)

[How the project was tackled 1](#_Toc205819995)

[Chapter 2. Literature review 3](#_Toc205819996)

[Introduction 3](#_Toc205819997)

[The early history of Logo and Turtles 3](#_Toc205819998)

[Logo 3](#_Toc205819999)

[The introduction of Turtles 3](#_Toc205820000)

[The introduction of Turtle geometry 4](#_Toc205820001)

[Logo since 1975 4](#_Toc205820002)

[Later developments in Turtles and other robots 4](#_Toc205820003)

[BBC Buggy 5](#_Toc205820004)

[The Valiant Turtle 5](#_Toc205820005)

[LEGO Mindstorms 5](#_Toc205820006)

[TurtleBot 5](#_Toc205820007)

[Conclusion 5](#_Toc205820008)

[Chapter 3. Requirements 6](#_Toc205820009)

[Project requirements 6](#_Toc205820010)

[Methodology 6](#_Toc205820011)

[Chapter 4. Spike work 7](#_Toc205820012)

[Picking software 7](#_Toc205820013)

[Java 7](#_Toc205820014)

[Python 7](#_Toc205820015)

[Conclusion 8](#_Toc205820016)

[Executing files 8](#_Toc205820017)

[The subprocess module 9](#_Toc205820018)

[The os module 9](#_Toc205820019)

[The exec function 9](#_Toc205820020)

[Threading 10](#_Toc205820021)

[Syntax highlighting 10](#_Toc205820022)

[HTML pages 10](#_Toc205820023)

[tkhtmlview 11](#_Toc205820024)

[Webview 11](#_Toc205820025)

[Conclusion 11](#_Toc205820026)

[Chapter 5. Design 12](#_Toc205820027)

[Use cases 12](#_Toc205820028)

[UI discussion 12](#_Toc205820029)

[Classes 12](#_Toc205820030)

[Files 12](#_Toc205820031)

[Setup wizard 12](#_Toc205820032)

[Backlash 13](#_Toc205820033)

[Wheel diameter 15](#_Toc205820034)

[Axle Length 16](#_Toc205820035)

[Chapter 6. Implementation 18](#_Toc205820036)

[Chapter 7. Testing 19](#_Toc205820037)

[How testing was approached 19](#_Toc205820038)

[Conclusions 19](#_Toc205820039)

[Chapter 8. Packaging? 20](#_Toc205820040)

[Chapter 9. Critical Evaluation 21](#_Toc205820041)

[Chapter 10. Conclusion 22](#_Toc205820042)

[References 23](#_Toc205820043)

[References 23](#_Toc205820044)

[Appendix A 24](#_Toc205820045)

[Appendix B- Manual testing table 25](#_Toc205820046)

List of Figures

[Figure 1- The project's GitHub Board 8](#_Toc207036700)

[Figure 2- Screenshot of turtle made with Java with its code 9](#_Toc207036701)

[Figure 3- Screenshot of turtle made with Python with its code 10](#_Toc207036702)

[Figure 4- Screenshot of input window for testing execution libraries 11](#_Toc207036703)

[Figure 5- Screenshot of code and window it produces for testing Python's threading 12](#_Toc207036704)

[Figure 6- Turtlebot application's use case diagram 16](#_Toc207036705)

[Figure 7- UI initial mock-up 17](#_Toc207036706)

[Figure 8-UI after combining spike work 18](#_Toc207036707)

[Figure 9-Final UI 19](#_Toc207036708)

[Figure 10-Spyder's layout 19](#_Toc207036709)

[Figure 11- Setup wizard flow chart 25](#_Toc207036710)

[Figure 12- Setup wizard flow chart: Setting the backlash 26](#_Toc207036711)

[Figure 13- Setup wizard flow chart: Wheel diameters 27](#_Toc207036712)

[Figure 14- Setup wizard flow chart: Axle length 28](#_Toc207036713)

[Figure 15- Setup wizard axle drawing 29](#_Toc207036714)

[Figure 16- Extreme example of "spamming" the run button 30](#_Toc207036715)

[Figure 17-Horizontal original application logo 33](#_Toc207036716)

[Figure 18- Final logo design 34](#_Toc207036717)

[Figure 19-Original turtlebot bird’s eye view 34](#_Toc207036718)

[Figure 20-Picture with PowerPoint blocks overlaid 34](#_Toc207036719)

[Figure 21- Abstract turtlebot GIMP image 35](#_Toc207036720)

1. Introduction

## Background

Before the project began, a turtlebot had been created. This was done by designing it using OpenSCAD and then 3D printing the chassis. The designs of which can be seen in FIGURES. The turtlebot has four main features. Two wheels that are controlled by stepper motors to move the turtlebot, a screen display and a component where a pen can be put in which can then be moved vertically by a servo. The turtlebot’s firmware uses an Arduino and also has a Bluetooth component in order to receive and acknowledge commands from the PC.

Images would be nice here of scad

In addition to the turtlebot, some simple software had been made using Java to connect to the turtlebot and then send it commands. These commands included ones to control the pen servo to adjust the pen height and move the wheels. The movement commands have options to move in either millimetres or motor steps (with 4096 steps per rotation). Initially, the command options for moving forward, left, or right.

The intention for the turtlebot (and the ones produced later) was for them to be used for outreach and open day activities so that young people could have a go at controlling them and drawing shapes.

To this end, an application with a simple graphical user interface was desired so that users could type in their code and send it to the turtlebot. The application was also desired to have a virtual turtle, useful for quick testing or if turtles needed to be shared around. The creation of this application was the project.

* Maybe something more about why application?

## Project aims and objectives

The project aims to create an application that beginner programmers can use to develop their coding skills by controlling a turtlebot whether physically or virtually.

To do this the application needed to include a place for the users to input their code, somewhere for them to receive feedback for it (such as for errors) and a virtual turtle. For connecting to the turtlebot, the application needed a way to open and connect to the correct port and ideally show the user if they had managed to successfully connect, so they could then send the turtlebot their code.

Additional requirements for features were added as the project progressed to help the user have a better experience, including the ability to save and load the developed code, as well as an information page so users could find out more about the project and the commands they could use for the turtlebot.

## How the project was tackled

The project was created as a CustomTkinter application in the language Python. It was developed by breaking the project into feature chunks and tackling it in an agile manner. This was partially done to fit better with the project’s changing requirements as multiple were added during the projects lifetime.

First, the ability for the user to code and have a virtual turtle respond was tackled. From then, the turtlebot was introduced, so connecting the turtlebot and sending it commands became the focus. Once these two components were addressed, the project shifted its focus to additional features to enhance the user experience, which is where the additions of features came into play.

* Guide to subsequent chapters.

This paper blardy blar……. :

* Lit review to explore similar stuff
* Requirements
* Spike work
* Design
* Development
* Testing
* Packaging
* Evaluation

1. Literature review

## Introduction

The use of robots to help people develop their coding skills has been a long-standing practice spanning over 50 years. A large quantity of these robots have had options to draw lines and shapes. Although the aims and base function of these systems have not changed drastically, the hardware and software have developed over time to provide a better user experience. Examples of this are the addition of graphical tools and hardware parts becoming smaller and cheaper. The first Turtles were developed as part of the development of LOGO [1]. Many of the more modern versions, software and hardware, have stemmed from this, such as Python’s Turtle Graphics [2][3].

This literature will explore the history of Logo and turtles, as well as other systems with the same aim.

## The early history of Logo and Turtles

### Logo

Logo was first invented in 1966 by Seymour Papert, Wally Feurzeig and Cynthia Solomon. In 1966, Papert visited multiple classes using programming languages and decided that a programming language designed specifically for children was required [4]. This language was then made and used in labs within the same year [2]. By 1967 the language was being tried out in schools.

Logo is a dialect of Lisp with some changes. One of the most important change was the introduction of commands[2]. Before this there were only procedures that could not return values. By introducing the option to return values, the options for programs greatly increased. Other smaller changes were made to make the language more child friendly. Lisp uses many parentheses, putting them around everything. Logo removed these.

Another choice taken to make Logo simpler is that it does not have an IDE. The idea behind this was so that the focus was to learn the language rather than trying to learn an IDE as well[2].

The first Logo programs were used for playing with words and sentences, aiming to help children to play around with them and learn grammar in the process [2]. One example of this was turning inputted words into pig Latin. These programs were originally run on time-shared computers (Digital Equipment Corporation PDP-1) [2].

The new language was seen to be successful, especially when the Logo environment ideology was kept to, which involved a culture of learning by doing [1]. The Logo group was officially formed at the MIT AI Lab in 1969[4].

### The introduction of Turtles

The idea of having physical objects to further develop the project was developed between 1969-70 [2]. Two of the objects developed were Turtles that could draw lines based on the commands they were given. The name turtle was inspired by the two automations that William Grey Walter built, which he called tortoises (Elise and Elmer more specifically)[2]. The original turtles did not have a graphical counterpart, which would come slightly later. The early turtles were connected to the computer via wiring to receive commands and send feedback[5].

The turtles had six commands: FORWARD, BACKWARD, RIGHT, LEFT, PENUP and PENDOWN. These could be combined within procedures to make a sequence of actions in hopes of creating shapes[1]. The layout of such a procedure to draw a triangle would be:

TO TRIANGLE

1. FORWARD 50
2. RIGHT 120
3. FORWARD 50
4. RIGHT 120
5. FORWARD 50
6. RIGHT 120

END

TRIANGLE

It was found that mistakes were more likely to lead to objective thinking as students tended to blame the turtle rather than themselves [1]. This meant that problems were dealt with constructively, focusing on why an issue occurred rather than the error itself.

### The introduction of Turtle geometry

Turtle geometry was introduced in 1970. This allowed a user to view a turtle and control it in the same manner as the Turtlebots as they worked off the same main six commands. Turtle Geometry worked with local geometry based upon the ideas of current position and heading [2]. It used standalone graphics terminals[2].

One example of its use was by Jon Pledge, a 12 year old at the time, who used it to create a universal maze-solving algorithm during an activity (now known as Pledge’s algorithm)[2].

After a conference in 1972 a company called General Turtle was formed in order to make and sell turtles[2]. Roughly three years later Marvin Minsky designed a standalone graphics system for the company [2].

### Logo since 1975

After the boom in personal computers, the development of Logo versions took up speed[2]. As of 2016 Logo has over 300 versions according to Pavel Boytchev’s Logo Tree Project[6]. Some of these are very similar to the original uses while others differed. Various programming languages implement as version of Logo by having their own version of Turtle graphics. The language was also used to create block-based learning environments such as Scratch[6][7]

## Later developments in Turtles and other robots

Similar to how there have been many versions of Logo, there have also been many different robots that be used similarly to turtles, most of which have developed from the original Logo ones. Not all of them have the ability to draw though many have it as an option, using modular sets.

The original turtles relied on a cord for communications. This was because radio links were considered to be too expensive given the turtles were being developed for use in Schools[5]. Since then non wired transmission mediums have become cheaper (and smaller) so are now the normal.

### BBC Buggy

The BBC Buggy launched in 1983[8]. This came in a set that users needed to assemble themselves with a range of programs. This set was modular so kits could be switched out depending on their intended purpose. One such kit was the pen kit that could be used with Logo to draw.

### The Valiant Turtle

Also released in 1983, the Valiant Turtle is controlled using LOGO to draw. It also had an on-screen version[9]. The turtle was controlled with infrared.

### LEGO Mindstorms

In 1987, LEGO started a collaboration with Logo to create turtles for them[2]. The first was the LEGO TC Logo. This required being wired to send commands through its ports. Other ports could be connected to sensors, lights and motors[10].

After this work started to put a processor into a large “brick” so that the bot could store and process its own commands. This led to the LEGO Mindstorms RCX in 1998. This concept was carried forward. Mindstorms are still sold today, though have gone through a few evolutions and now use block-based programming [11].

### TurtleBot

The TurtleBot is more for adult users than the original turtles. They are based upon Logo[12]. People can buy kits to make the robots, and this can lead to diverse applications, including home use and research[13].

## Conclusion

Many of the turtlebots and alternatives have had similar characteristics to the turtlebot for this project. The basic functions are naturally the same as pen controls, and movement requirements have not changed since 1969. The transmission of commands is similar to most other bots mentioned, with the exception of having a different medium, using Bluetooth rather than wired or infrared options. Like the original turtles, the turtlebots produced for the project are also made with the mindset of keeping production costs as cheap as possible.

The turtle graphics library that is used for the project is based upon the original one using the same concepts of position and heading even if the technology has advanced to where it no longer requires a separate graphics terminal.

The aim of turtles has not changed between the originals and this project. They both have the objective of getting beginner programmers to adopt coding by having a go.

1. Requirements

## Basic project requirements

The project started with some simple basic requirements:

*Basic Requirement (BR)1: A user should have an application where they can write code.*

This requires having the application with a writeable textbox. It also implies picking the code language either by creating one or picking an existing one.

*BR2: A user should be able to view a turtle simulation to see the output of their code.*

This requires picking a turtle library (or creating one, but reinventing the wheel is unnecessary). This simulation needs to be placed on the application screen and requires a way to receive the commands from the user-written code.

*BR3: A user should be able to connect to and send commands to the turtlebot.*

The application will require the ability to connect to the turtlebot’s port and send it messages. It will need to be able to receive acknowledgements as well to function well. This means that the code written by the user needs to either match the turtlebot commands or be translated into them before sending.

*BR4: A user can configure the pen height*

This is so the pen can be configured to be just on the page to reduce damage to the pen. The user will require a way to adjust the down value to test out different values.

*BR5: A user should be able to receive error feedback on their code*

The application will require a place to display error messages. This will show the user why their code does not work, so that it may be improved upon rather than leading to frustration from not knowing why.

## Additional project requirements

As the project progressed, more requirements were added beyond the basic functionality as it became apparent that there would be time to do them.

Additional requirements:

*Additional Requirement (AR)1: A user should be able to stop the code when running*

This is in case of the code being incorrect from what the user expected to happen or in the case of needing the stop the turtlebot (e.g. it has run off the paper). This will require a stop button that can stop both the simulation and the turtlebot.

*AR2: The user’s code input box should have syntax highlighting*

This is so the user can get debugging information before trying to run the code. For instance, if what is intended to be a string is not highlighted as such, it visually indicates that the code has an error.

*AR3:A user should be able to access information pages*

This is so that the user can access additional information about the turtlebot. They can also view the possible commands that they can type. For this purpose, the pages should be viewable from a separate window so that the user can go back and forth easily between viewing them and the main application to type the commands.

*AR4: A user should be able to change the font size of the code text and output text*

The application should allow the user to change the font size. This allows the application to be used and the code to be read on different screens, and additionally increasing application accessibility.

*AR5: A user should be able to save their current code and load their code from a file*

The application should allow the user to save their current code that they have been developing as well as load it back in. This means that the user will need to buttons to access these functions. The user should also be able to name their file when saving so they can recognise it when trying to load it later.

*AR6: A user should be able to set up the configurations of the turtlebot so that the drawings will be precise.*

The application requires a setup wizard-type process so the turtlebot can be configured with accurate values for its axle length, wheel diameter and motor backlashes. This is not to be done by the general user but will be required during the initial setup of the turtlebot and potentially later if drawing imprecision is found.

This will require a separate window guiding a user through drawing steps and what to measure. This separate window will need to be accessible through the main application.

*AR7: A user should be able to insert premade characters and shapes.*

This will allow users to quickly generate words and shapes. One benefit of this is that beginner users can view how to make a shape and then play around with it to match their purposes. This introduces a stepping stone into learning the system for those who are struggling to get started with the program.

To do this, the application will require a list of available shapes that the user can select from. The code to make the selected shape will then need to be retrieved or generated and then inserted where the user types their code.

All requirements can be found in Appendix B.

## Methodology

To meet these changing requirements, a fairly agile approach was taken using a GitHub board to track progress. The requirements were broken down into smaller chunks.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 1- The project's GitHub Board

As seen in Figure 1, the board was loosely based upon scrum with the idea of picking items to do that week from the Project to-do list/backlog. Some requirements required more than one week, and some others went back into the to-do list once it became apparent that they needed redoing.

As the project progressed, the board was used less as the to-do list reduced, so tracking project progress became simpler, especially given its nature as an independent project. During later stages of the project, a simpler to-do list was kept, just crossing off items as they were done rather than using the whole board. Newer requirements were added to this list as required.

1. Spike work

## Picking software

The two obvious choices for writing the software were using either Java or Python.

### Java

Java had the benefit of already being proved to be able to communicate and send commands to the project.

Java also has a turtle library which can be used to create a virtual turtle and program it with it commands (as seen in Figure 2), and with an additional library such as JavaFX, an application could be produced using this.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2- Screenshot of turtle made with Java with its code

One major downside of using Turtle with Java is that there is limited documentation for it with no central place with most needed information. This makes it harder to develop a more complicated system with it.

### Python

Python also has a turtle library which can be used in a similar way to the Java one (as seen in Figure 3). It also has formal Turtle graphics documentation[3] which can be easily accessed. Spike work was additionally done to check that the turtle could be embedded into a window which could have additional features added to it such as a button. CustomTkinter was used for this given previous familiarity with it.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 3- Screenshot of turtle made with Python with its code

By creating the application in Python then it allows the users to also write their code in Python. This achieves the application's aim of getting users coding with a proper programming language, whilst making development easier as no translation would be needed between languages.

Python is quite commonly taught in schools at secondary schools so a fair few potential users may already have some familiarity with it. It is popular since it resembles spoken English in many ways so can be easier to pick up. Given this it may prove easier for activities for outreach and open days.

The downside of Python would be that connecting to the turtlebot has not previously been worked out, but given Python’s popularity and large amount of documentation, there will be something out there to achieve this purpose.

### Conclusion

Python was chosen for the reasons listed above. The CustomTkinter library was chosen to build the GUI as a turtle could be easily embedded. CustomTkinter builds upon the Tkinter library which is one of Python’s basic GUI building libraries. CustomTkinter allows Tkinter to be used alongside newer elements such as themes and more modernised GUI features.

Mention Python’s compile ability(SEE BELOW FOR FURTHER DETAIL)

## Executing a user’s code

For a user to be able to create code and then run it within the application it is crucial to have a way for the code to be retrieved and run dynamically. Python has multiple libraries and ways to do this with their own benefits and drawbacks.

These were explored for their suitability by having a simple python file that would print hello world if executed correctly. Another file with the chosen method to read in the file and execute it was then created. These were tried both with the hello world file being correct and when it has errors.

Once the options had been explored this way, they were tried with a CustomTkinter window containing a textbox where “print(“Hello world”)” was written and a button that called a command to run the code that was written. This can be seen in Figure 4. The command was changed to use each library option.

A screenshot of a computer error

AI-generated content may be incorrect.

Figure 4- Screenshot of input window for testing execution libraries

### The subprocess module

The subprocess library has multiple functions that were tested out. Call, run and Popen were all tested out. These all did not output the expected result of “hello world”. By using the check\_output function and then printing out its results, it can be found to return "b'hello world\r\n'”. After investigation, the subprocess library seems to be more for running command line processes rather than Python files making it not suitable for this project.

### The os module

The os module offers the system function that can be used to read and execute files. It proved to be effective for the first text, printing “Hello world” to the console. Once CustomTkinter was introduced it stopped functioning.

### The exec function

The exec function is built into Python’s basic functions and executes the string of code it is given. To use this the python file must be opened and read then the contents passed to the exec function. All these functions are within Python’s standard library. This method worked successfully for both tests. For the first test, code errors in the hello world file caused the system to stop. This was not a problem once CustomTkinter was introduced as the application would keep running, although throwing errors when the code was wrong.

This method has the original benefit that it does not require the input to come from a file, as the exec function is looking for a string. This means that the code can be retrieved straight from the textbox and passed to it, skipping the middle steps.

The main issue with the exec function is that it will try to run any code it is given. This means it has security issues though since the use of the application is likely to be monitored it should be a minimal issue for this application.

The exec function shall be used as it works for desired application.

## Threading

Research was done into threading with Python, as the application would require multiple processes to be handled at once. Python is single threaded by default, and many libraries rely on being on the main thread, such as CustomTkinter. Self-made processes can be run on separate threads. This was attempted, utilising the turtle module to get two turtles to run in parallel to test using the threading library (which can be seen in Figure 5). This was successful, so it will be usable for more complicated parts of the final system.

A screen shot of a computer screen

AI-generated content may be incorrect.

Figure 5- Screenshot of code and window it produces for testing Python's threading

## Syntax highlighting

For a better experience, coding it was decided that syntax highlighting would be useful for the box where users enter their code. This could be done for known keywords, such as for or strings, but would be better done with a library, as all the main keywords and syntax would already be known. There are a few libraries that serve this purpose, two of the most common ones for use with Tkinter are Idlelib and Pygments. Both libraries worked for their purpose when tried out, with the same result from the user's point of view.

Given that the application is intended to be kept simple and easily maintainable for developers, Pygments proved bulkier and required more lines of code to achieve the same purpose. Idlelib is more streamlined from a developer using its point of view and allows for easy choosing of colours for each kind of syntax highlighting that is wanted.

## HTML pages

The user will require some guidance about what code they can write and may wish to learn more about the software and hardware. To this end, some About pages would prove beneficial. To do this, it was decided to write the information in HTML for the application to then open and display on a separate window. This way, the pages could be updated without prior Python knowledge and without having to open the main code. A library that could provide this function was therefore required.

There are two main libraries that can open HTML pages inside Tkinter windows, both of which have their own benefits and drawbacks.

### tkhtmlview

The tkhtmlview library is specifically designed to work with Tkinter. HTML pages can be opened using it by putting files into objects such as its HTMLScrolledText, which can then be packed and displayed in a window. The main drawback of this library is that it doesn’t support the navigation tag. When used, it tries to open an unreachable webpage.

### Webview

Webview works by creating a new window using the HTML file. This works when it stands alone, and the navigation tag can be used. Problems arise when attempting to move between the About pages and the main window. Doing that causes the application to crash. Since the pages are designed to be referred to for command options whilst typing them, this is less than ideal. The Webview library only functions on the main thread, so putting it onto an alternative thread was not an option.

### Conclusion

The tkhtmlview library was chosen to be used since it mostly worked, and its drawback was minor in comparison.

1. Design

## Users

The application and turtlebot will need to be used by different types of users. These can be broken down into different groups, though there may be some overlap.

User types:

* Young and/or prospective students via open days or outreach activities who are new to coding
* Other students who have had more coding experience
* People demonstrating the turtlebot’s capabilities to draw more complex designs or leading activities using the turtlebot.
* The person who is doing the initial configuring of the turtlebot (using a setup tool)

The varying range of coding experience means that the application needs to be designed in a way that is beginner-friendly friendly with additional guidance available, whilst allowing those who are more familiar with the concept to get on with coding. This means that the application should allow the users to access more information rather than have it forced upon them, i.e. having them have to complete a step-by-step tutorial before starting. Help/ Information pages are useful for beginners, along with an example which can be provided as a default.

Given that some students will have had experience, it is better to have the application laid out in a way that is not too dissimilar to similar applications so that they can navigate easily without having to relearn aspects. It should also include some features that they would expect, such as syntax highlighting.

For those demonstrating the software, they may require having the screen display on different sizes, so the ability to alter the text size so it could be read across a room would be beneficial. It may also prove useful to be able to insert or load pre-made drawings/code to be able to demonstrate a design practised on a prior occasion.

The person configuring the turtlebot will need a way to step through the stages to do so. More information on this is available later in the chapter.

### Use cases

Based upon the requirements discussed within the requirements section and the types of users, certain use cases became apparent (see Figure 6).

UC1 Writing a program: When a user wants to write a program they have various options they can do within this.

UC1.1 Enter code: The user can click and type their code into a textbox

UC1.2 Run code: The user can click a run button to run their code on the simulation (and the turtlebot). The user can also choose to stop the program running whilst it is running in case of an issue or undesired behaviour.

UC1.3 Clear program: The user can choose to clear the code textbox and start with an empty one by clicking the clear program button

UC1.4 Reset turtle: The user can choose to reset the virtual turtle to its original starting position

UC2 Connecting the turtlebot: When a user wants to connect a turtlebot, they can pick an available port and the application will display if it is successfully connected with the turtlebot. The application will then send commands to the turtlebot as well when the user says to run the code.

UC3 Setup wizard: When a user wishes to configure the turtlebot they can open a setup wizard that takes them through steps to configure the wheels’ backlash, diameter and the axle length.

UC4 Adjust the pen height: The user can change the height of the turtlebot’s pen until they are happy by selecting different ones.

UC5 Change the font size: The user can open the menu and select a new font size to change the text size of the code text and output text.

UC6 Save code: The user can select to save their current code from the menu and name the file.

UC7 Load code: The user can select to load a file. They can then pick a file to load into the code textbox.

UC8 Insert shape: The user can select a shape or character to load into the code textbox via the menu

UC9 Reading the About page: The user can click on the about button on the top menu to load an additional window with more information about the project, for those curious, and with extra directions about coding and what commands are available to them.

A diagram of a diagram

AI-generated content may be incorrect.

Figure 6- Turtlebot application's use case diagram

## User Interface discussion

The user interface (UI) needed to be kept as simple as possible whilst still being attractive to passers by and young people. This meant intuitive buttons, colour all whilst not overcrowding the application. The less baseline features were planned to be kept near the top of the screen where they would be out of the way of main use.

To get an idea of the initial design, Microsoft PowerPoint was used to create some mock-ups based on the initial requirements; one of these mock-ups can be seen in Figure 7.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 7- UI initial mock-up

From the beginning, it was clear that the user would require a large place to type their code, a place to view it in action via the simulation and various buttons to run the code, pause/stop it and to reset it. These buttons would later be moved to the right-hand side.

During the spike work process, an output box proved useful to give notifications when there were errors with the code. This was deemed an important requirement for the end UI, so it was incorporated into the design as can be seen in one of the early iterations of the design. Figure 8 shows a design with this new box incorporated.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 8-UI after combining spike work

Over time other features were added to the design to support the added features. This mainly consisted of what can be found on the top menu now.

The colour green was picked as the application's main colour. This was picked, given that turtles are green so it made sense.

The final design can be seen in Figure 9. This includes the applications icon which is based upon Python’s turtle library’s turtle.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 9-Final UI

The final UI layout shares a lot of similarities with Spyder’s layout, which was used to create the code. This can be seen in Figure 10. There is a chance of unconscious bias towards this design; however, it does seem to work for Spyder. There are some design differences, for instance, the run button is in a different place, and Spyder does not have a virtual turtle.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 10-Spyder's layout

## Classes

*CLASS DIAGRAMS AND DESCRIPTIONS HERE*

* *Contents/Purpose of each class*
* *How classes link/are used by others*
* *Any additional files they use*

### Classes for programming and operating the turtle

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

## Other Files

The project consists of eleven Python files. In addition to this, there are four other types of files used. Graphics (Portable Network Graphics and ICO) files for the graphics used within the application and for the application’s icon. Most of these file are found within the project’s graphics folder.

A HyperText Markup Language file is used for the information pages. This is so it can be edited and updated without the need to change the application’s code. It also has the benefit of the editor not needing to be familiar with Python. This file contains information about the turtlebot as well as guidance on how to use the application, including code command information. This file can be found within the project’s html\_info folder along with the images it uses.

The other file is a JSON file. This defines the application’s colour themes for CustomTkinter aspects. This theme is used by the application’s main class and defines the button colours and the top menu colour among other elements. The file can be found within the main src folder.

Text files are found within the characters folder’s subfolders. These contain code that can be inserted into the code editor of the application for quick design building.

## Setup wizard

The setup wizard is designed so that each turtlebot can be configured to the correct dimensions. These dimensions can vary slightly mainly due to shrinking as the plastic cools down once the parts are 3D printed. When the dimensions are slightly off it can lead to results such as circles and squares not matching up or lines drawn not being the expected length. Although this is minor was smaller drawings, when larger drawings are attempted this error is significant.

To stop these problems, a setup wizard is produced so that a user can use it to work through various steps to calibrate the turtlebot to the correct dimensions and values. The dimensions in question are both wheels’ diameters and the distance between the two wheels (axle length). It is also important to measure the backlash of the motors so they can be factored into any lines to be drawn.

By drawing shapes rather than using equipment to measure, the accuracy is increased. As an example by getting the robot to draw a line twice the circumference of the wheels, then the percentage error from measuring equipment is halved. Drawing the shapes, also means that a simple ruler can be used rather than specialist equipment (callipers).

The setup wizard works in three stages: working out the backlash, working out the wheels’ diameter and working out the axle length (see Figure 11). After the dimensions are calibrated to a sufficient level they are saved to the EEPROM. After this, the turtlebot loads these values to use within the main program upon future startups.

A diagram of a process

AI-generated content may be incorrect.

Figure 11- Setup wizard flow chart

### Backlash

The first calibrations are done for the turtlebot’s backlash. This is done by having the turtlebot move backwards and then try to move it forward. If the turtlebot moves then it was instructed to move more than its backlash.

The wheels rotate by using geared motors. The gears have teeth which have a certain number of steps and rotation before the next tooth is moved to. This is distance, measured in the number of steps in this case, is called the backlash. The backlash needs to be factored in when the motors first start moving after stopping or when the direction is reversed. The backlash being uncalibrated means that the lines drawn will always be wrong by the same amount leading to high precision but low accuracy for the line we wanted.

A diagram of a flowchart

AI-generated content may be incorrect.

Figure 12- Setup wizard flow chart: Setting the backlash

As seen in Figure 12, the turtle originally moves back by a large amount of steps. The increment starts as a fifth of the max amount. The turtle attempts to move forward by this amount.

If no forward movement occurs, then the turtle attempts to move forward by another increment amount. If movement does occur and a fifth of the increment is a whole number then the increment becomes a fifth of itself. The turtle then moves backwards by the last value it moved forward by and the moving forward process starts again.

If the increment becomes less than one, then the new backlash has been found and is saved. This is because the backlash is saved as the number of motor steps, which can only be a whole number.

### Wheel diameter

The wheel diameter can be worked out whether a line drawn is the correct length. If a straight line drawn is shorter than expected then the wheel diameters are smaller than expected. A longer line indicates that the diameters are bigger than expected. We assume that the diameters of both wheels are equal. This is likely as they are printed at the same time so should have been effected equally by factors. If this is not true, it also becomes apparent during the line drawing test, though it cannot be fixed using the setup wizard.

A diagram of a flowchart

AI-generated content may be incorrect.

Figure 13- Setup wizard flow chart: Wheel diameters

As seen in Figure 13, the user prompts the turtle to draw a line that should then measure 300mm. The user then uses a ruler to measure the actual length of the line and inputs this value. If this value matches the expected value, then the diameters are already correct, so the wizard can move to the next step. If this value does not match, then the new diameters are calculated by the fraction that the length was off by multiplied by the expected diameter. This new diameter is saved and the user can have the turtle draw the line again to see if the new diameter value is correct.

### Axle Length

Axle length becomes important when the turtlebot attempts turns. If the axle length is shorter than expected then the turtlebot will under turn and vice versa. This leads to shapes not connecting properly, either forming a gap, overlap or cross shape.

A diagram of a flowchart

AI-generated content may be incorrect.

Figure 14- Setup wizard flow chart: Axle length

As seen in Figure 14, a sequence of steps is undertaken to calibrate this. The turtlebot first draws two circles. Each circle uses one wheel as a pivot point while moving the other wheel to draw a circle. Each circle is calculated by working out how many wheel motor steps need to be taken to do a circle with the diameter of the axle. This is done by dividing the circumference of an axle diameter circle by the circumference of a wheel diameter circle and then multiplying this by the number of steps in a full wheel rotation. An example of the circles drawn can be seen in Figure 15.

*A green robot with a purple and green wheel and two circles on a white surface

AI-generated content may be incorrect.*

Figure 15- Setup wizard axle drawing

The user is then prompted to input two length values. If there is a gap in the circle the user inputs how much by as a negative. If there is an overlap the user puts in how much by as a positive number. If the circles are good then the user can input two zeros, in which case the setup wizard will move onto finishing. If there is a gap or overlap, the two inputted values are averaged. The expected axle value then has this averaged value divided by pi subtracted from it to get the new axle value. The user can then repeat the drawing process in hopes of seeing an improvement.

1. Implementation

## Introduction

Due to the nature of the project, many of the aspects were done in parallel. At the beginning of the project, there was no access to the turtlebot, so the code input and turtle simulation were the focus. As time progressed, a turtlebot was acquired. Once the basic functionality was achieved, additional features were added, along with the firmware being changed to support aspects such as the saving of turtlebot dimensions, which could then lead to a setup wizard being able to be created.

For the purpose of describing the implementation of the project, it has been broken up into blocks of functionality or of significance in implementing, with the addition of some key events.

## Basic functionality (pre-turtlebot)

After the spike work, multiple components existed that, if combined, would give a virtual turtle that could be moved via code that a user wrote. The components already made were a textbox with syntax highlighting, a virtual turtle simulation, a text box that showed error information and a run button. These were combined together resulting in three classes: a main class that ran the application, a turtle simulation class that handled the code and a code input class that created the textbox. After combining, the user could write their turtle code in, and when they pressed run, the simulation would run their code. The inputted code matched Python’s Turtle library.

A reset button for the turtle simulation and a clear all button for clearing the current program were then added. The application at this stage can be seen in Figure 8. The clear all button was later moved to be within the input box to make it clearer what it was clearing.

It was then found that some accidentally interesting results could be produced as the run button could be pressed repeatedly. This interrupted the current running code and replaced it with the new code again. An extreme example of this can be seen in Figure 15. To fix this, disabling the run button until the program had finished running was introduced.

A screenshot of a game

AI-generated content may be incorrect.

Figure 16- Extreme example of "spamming" the run button

## Responsive design

Responsive design was added to the project after the basic features were created so that the code boxes and simulation boxes always took up half the screen each, even as the window size changed. Additionally, code was added to alter the run and stop button layout when it can no longer fit horizontally to display them vertically.

## The challenge of stopping a running program

Being able to pause or stop the program lead to many of the design decisions of hw to have the code run. Originally the code was run by creating a thread for it and passing it into the exec() function within this thread. In Python there is no way to stop a thread. The closest you can get is to pause it for a given number of seconds. Given that a user does not want there program to unexpectedly start running again an alternative solution was required.

Python has a multiprocessing library which works in a similar manner to its threading library but can be stopped or paused. This library cannot be used successfully with Tkinter as it causes the application to stop working. No other libraries were successful for achieving the desired result so an alternative approach to handle the code was chosen.

### Taking a line by line approach

A new approach to handling the code was chosen. This was the process the code line by line to the turtle simulation so that it could have a conditional statement saying whether it should be paused. If paused, it would not do the action. This version also did not use threading.

Code was checked using Python’s compile() function to check for syntax errors. It was then converted into a list of lines of code. These lines were then passed to a exec one by one to be carried out and printed to the output box.. Within this an exception was added to check for name errors that the original compile could not catch. An additional step was later added to convert each line to turtlebot commands to put into another list using a dictionary.

Once additional language features were added in, this method reduced in benefit. To process more complex code features such as for loops this resulted in creating a half-made compiler that could unroll the for loops to put them into the list correctly. Additional tweaks had to be done so the code could handle lines with just whitespace.

Later it became apparent that variables would be useful so they were introduced by saving their names and values into a dictionary. Before the code was run the names were substituted for their values. A recursive function was then added to handle nested variables.

By choosing this method, the program could be stopped, however, it greatly reduced the code features that could be used as each one had to be added in separately which as a learning to code is counterproductive. A new approach was therefore required.

### Using an additional class approach

The code was changed so that it used a new class which knew how to handle each command. This version went back to putting the code into the exec function to run. The exec function handles to features of Python so the code does not need to do itself. The new class acts as a middleman to then process and translate as needed to send the commands to the classes that handle communicating with the turtle simulation and turtlebot. The functions within this class match up to the functions that a user can type to control the turtle.

This version was far more streamlined than the previous version, with the same benefits and all the normal Python features available, including being able to import modules so users can use features such as randomly generated numbers.

To stop the turtle, an else condition was added to the turtle simulation’s check if it was stopped. If triggered, an exception is raised, stopping the running program.

## Adding the turtlebot to the project

The first step of getting the turtlebot connected was getting the available serial ports. This was originally done separately to the project for testing out purposes. Python’s serial library [14] was used to do this. A continuous read was then done on the port to read in incoming messages. This was done in preparation of getting the turtlebot, so there were some parts to change.

When the turtlebot was connected, the program managed to get the port successfully and read from it however, there were issues with connecting to the turtlebot via it. To establish the link when a message containing “AT” is received, an “OKPC” message needs to be sent back. Originally, this was not working, but later it turned out that the command required a newline character on the end. The “OKPC\n” command does not return any message the the system probes the connection by sending out hello messages until it receives an acknowledgement. After this, the port can be sent commands to control the turtlebot.

This code was then added in via the Port\_Manager class. This class keeps track of the setup process to know whether the USB dongle is connected and whether the turtlebot is connected. These two connections are shown visibly on the application’s top bar using another class, so the user can also be aware.

Originally, these came with a refresh button to reset the connection, but this was later replaced with having the user be able to pick their chosen port.

The Port\_Manager runs the port reading within a separate thread from the main program, so it can still run. The writes are done within the main thread. Once the turtlebot had a connection established, the turtle graphics commands needed to be translated into turtlebot commands (which can be found in Appendix C). This was originally done with a dictionary to translate and then send the commands to the Port\_Manager class, but was later changed to just directly send the commands to the class without the dictionary.

Later changes to the Port\_Manager class included adding a buffer limit value so that the system waits until the turtlebot is likely to be able to hold the command in its buffer. This was done as when sending small, rapid commands, some were being lost as the buffer was overflowing. The system keeps track of how many commands it is waiting for acknowledgements for, and if this number is higher than the buffer allowance, then it waits before moving on to something else.

## Moving the pen

Once the turtlebot was connected a way to adjust the pen height was needed so the drawings would definitely be drawn without damaging the pen. To do this a pop up with a slider was made. When the slider moved so did the pen. This is so the user can move it until it is just touching the paper underneath. The code for this is fairly simple. The issues arose with the turtlebot’s pen servo not always responding. This issue with the turtlebot was seemingly fixed by the time of project completion as it now responds in the expected way.

## UI developments

After the main functionality of the application was achieved. The user interface was decided to be improved to be more appealing and have some standardisation. Before this, the application looked as it does in Figure 8.

First, the colour green was decided as the main theme colour, given its association with turtles. A darker colour was picked for the top bar, which at this point consisted of two non-functioning standard buttons. The other buttons were changed to be a lighter green to contrast against the rest of the application.

Additional features were then added to the application. One of these features was the ability to change the font size of the text boxes of the application. Another was the addition of default text. Both of these changes were done to increase the versatility of the application and help users get started.

Later, the code was changed so it used a JSON file to define its CustomTkinter component’s theme. This was used to set the used components' default colour to green, so it does not need to be defined for each button, only overwritten for special components. This file was taken and altered from the official CustomTkinter documentation [15] to meet the desired theme.

### Graphics

#### Creating the logo

All graphics were made using GIMP [16]. Microsoft PowerPoint was additionally used for producing the abstracted turtlebot image for the setup wizard.

The application’s logo was made using GIMP. It was originally made horizontally using Python’s Turtle as a rough visual guide for proportions. This was then turned into a blocker version, as seen in Figure 16.

A green puzzle piece with a white background

AI-generated content may be incorrect.

Figure 17-Horizontal original application logo

The logo was then rotated to be diagonal to fill the square space better. It was then also smoothed out further. There is the option to have a green edge or a white one, depending on use. The application uses the green one, and the user manual uses the white outline. The final GIMP file design can be seen in Figure 17.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 18- Final logo design

#### Creating the setup wizard graphics

Multiple graphics were created for this. The path drawings were made using GIMP’s pencil tool. The abstract turtlebot was a little more complicated.

A picture of the top of the turtlebot was taken, see Figure 18. Key shapes blocks where they picked and overlaid with PowerPoint shapes, see Figure 19. The shapes were then simplified further, and outlines were added with GIMP. The final result can be seen in Figure 20. This image was then added to other images to demonstrate how the turtlebot would move during the setup wizard.

A green robot with wires

AI-generated content may be incorrect.

Figure 19-Original turtlebot bird’s eye view

A green and blue object with black lines

AI-generated content may be incorrect.

Figure 20-Picture with PowerPoint blocks overlaid

A white and green camera

AI-generated content may be incorrect.

Figure 21- Abstract turtlebot GIMP image

## The creation of the top menu

### Menu buttons

Buttons were required to access the additional functions of the application. These preferably would need to be dropdowns so that the application was not cluttered with a the features. The additional challenge was to try to check the connection states on the same line as the drop down buttons.

Originally, Tkinter’s menu bars were investigated however these only allowed having menu buttons on them and other components would appear below the bar and two bars would not be space efficient. The menu buttons by themselves had potential though.

Tkinter Menubuttons can be normal buttons or contain submenus with normal options or radio button options. These Menubuttons were used to construct a menu bar on a normal frame, which allowed the connection states to be displayed upon it. The first tackled part of the menu was getting the font size to be able to be changed using it.

### Font size issues

The picking of the font size used radio buttons. A loop was attempted for adding buttons for each size in a list of possible sizes. This originally had issues as it always carried out the last option in the menu despite picking another size. This turned out to be an issue with lambda use. The code originally said “lambda: change\_textsize(i)” and needed to say “lambda i=i: change\_textsize(i)”. Once this fix was done the changing of font size behaved correctly.

### Saving and loading to and from files

The File\_Handler class was introduced, whose methods are called when saving or loading a file. They both use CustomTkinter’s filedialog module, which has classes to handle opening a sav as file dialogue or opening a file dialogue. Both of these can be used to get a filename which can then be used to open the file and read or write to it as needed. For saving the code in the code textbox is passed to the function. For loading the function has access to the method to set the code textbox’s text.

### Insert premade designs

To provide the option of inserting simple premade shapes the File\_Handler class had two new functions added. One to find the available text files within a specified folder name and one for handling the reading from a chosen file and inserting the contents. The list of available files are added to the menu under the insert option under submenus of their folder name. This was done dynamically so that additional shapes and alphanumeric options can be added as desired.

### HTML pages

Details about the development of the HTML pages can be found within HTML pages section of the Spike work chapter. The project ended up using tkhtmlview’s HTMLScrolledText widget to pack the HTML page into a window. The user can use this to scroll through the page contents to find what they are looking for.

### Port picking/swapping

Originally the application worked by always trying to open the first port in the list. This is not always the desired port however o a way to pick a port was needed. This was done by introducing a pop up that contains a dropdown of the port options for the user to select from. It was originally hoped that the list could change in real-time to reflect the current ports available but that would unfortunately require putting this in a thread and Tkinter will not work with a thread outside of the main one. Instead a refresh button was introduced so the user can update the dropdown themselves. This new method allowed a port to be picked and opened however only one serial port was available to test on.

During the next meeting a second turtlebot with USB dongle was used to test this new feature o so ports could be swapped between. This showed that the swapping of ports was not correctly working still. After three hours of trying to find the issue, it became apparent that the method to set the port was not calling the method to actually open a the new port. Once opening the new port was fixed, swapping ports worked successfully. After the meeting the Port\_Manager class that handled the ports was changed so that when a new port is selected the old one is closed, the new port is opened, and a thread is started to read from it. The class was also generally streamlined to reduce code duplication so that any further issues could be more easily identified. A label was then added to the top bar to show which port was opened last, done largely because it proved easy to forget which port was attempted to be opened last while working during the meeting.

## Open day tweaks

Since the application was in a state where it could be tried out by users, it was taken to an open day for some informal testing. This meant it could be seen how it performed on the lab computers and with users.

It was also a good way to see if any additional features would be desirable. Some of these features have already been described such as having a theme for consistency and having the ability to insert shapes. The difference between the text size between the code textbox and the output text was more obvious on the bigger screens so it was decided to have changing the font size change the size of both.

One of the prospective students tried to use variables and import libraries, so it became apparent that, ideally, the application should be changed to be able to handle this, so it was. It also became apparent that an undo option on the code textbox once needed which was implemented by setting the ScrolledText widget’s parameter of undo to true.

The virtual turtle was found to commonly run off the simulation screen which was undesirable, so a way to fix this was needed. Originally, it was planned to do this with scrollbars, though this proved unfruitful.

It also proved a good opportunity to see how the application ran on a Mac device. Here it was found that the menu buttons displayed as white (the default on Mac) despite being specified to be green. This led to specifying if the platform was a Mac one to have the writing be green instead of white so it was legible. Loading a file was also found to not work on Mac which turned out to be a difference in whether the operating system can handle looking for any text file without the asterisk wildcard character. Word can, whilst Mac can not.

## Scaling the turtle

A way needed to be found to have the virtual turtle be visible or at least findable on the screen at all times. This presented some challenges due to how turtle graphics is designed. The turtle works on a coordinate system but has no information about where the borders of its screen or the canvas it is on are. Originally the plan was to introducescrolling bars so the turtle could still be found. This had its own issues as they could not be made dynamically as the turtle moved in that direction as there was no way of keeping track of where the turtle travelled. Having a static bigger canvas with scrollbars to view it would result in still having the same problem, just having the turtle draw a bit more of the line first.

The decision was made to scale the turtle’s drawings based on the space available. If the drawing would result in leaving the canvas then the line would be scaled to fit the canvas. The achieve this the desired lines were all taken before drawing to work out how much overall space the shape drawn would require. The lines were fed into an algorithm to do this.

First the available space is worked out based on roughly where the turtle is and the canvas size. The exact location of the turtle cannot be determined as its coordinates will be (0,0) but where this appears on the canvas can change. This is mostly fixed though is the source of the issue when scaling does not work.

The amount the turtle moves in the North, East, South and West directions are tracked so the scale needed for each direction can be calculated. The angle the turtle is facing in is also tracked.

For each line, the algorithm determines what type of command it is.

* If it is a forward command, it works out the vertical and horizontal distance that is travelled based on the current angle. These distances are added to the corresponding directions moved in.
* If it is a curve command, it is first processed into lines, then follows the same process as a forward command. More detail is given later in the chapter
* If it is a right or left command, then the current angle is altered to match the angle the turtle would now be facing out of 360°

Once all the lines have been processed, the scale the turtle would have to be to fit within each particular direction is calculated based upon the available space. The smallest of these scales is then set to the current scale. When drawing, this scale is used to shrink each line before drawing, as well as shrinking the turtle to match.

## Setup wizard

### The calculations

Before the setup wizard could be created the shapes it would draw had to be decided. To work out the axle length of the turtlebot based on a shape the wheel diameters needed to be known. Therefore the diameters needed to be worked out before the axle length. It was decided to assume that both wheels had the same diameter which is likely if they are printed at the same time. Various shapes were tried out for working out the axle length. In the end the simplest approach was given by the project’s supervisor along with the equations to work out the configurations based upon the shapes drawn. This approach worked by working out the percentage error of the given shape to then adjust the value that was expected by.

The original calculation given for working out the correct axle length was not correct. This became apparent when running the program and the circles drawn became worse rather than better. The equation given was:

This is slightly incorrect due to the pen being in the middle of the turtlebot so the circle drawn has a radius of half the axle length and a diameter of axle length. This difference can be seen in Figure 22 where the solid line is where the pen draws and the dashed line is where the wheels move.

A white board with writing on it

AI-generated content may be incorrect.

Figure 22- Axle and pen paths for a circle

This changes the equation to:

Which can be simplified to:

This change led to the circles meeting up when corrected by the equation.

### The wizard

The wizard was developed incrementally by each calibration stage. First, a pop-up was made that attempted to send a get command to the turtlebot to get its current calibrations. The pop-up also needed to handle prompting the user if no turtlebot was connected. The pop was first made to be functional, then changed to be more user-friendly with images for guidance.

#### Backlash

After that, the next stage was working on the backlash, which took some trial and error. Variables were made to hold the maximum value that the backlash could be, the minimum value that could be, the value currently being trialled, and the increments between values being trialled.

Two buttons were introduced to handle user input. The no difference button informs the system that the turtlebot did not move, meaning that the backlash must be higher than the current value. The moved button informs the system that the turtlebot moved, indicating that the backlash is equal to or less than the current value.

This then works iteratively by making the current value the new maximum value and the minimum value the previously tried value. The increment is then reduced, and the moving process starts again. The trial and error came into play, getting this concept correct. It later had to be tweaked as the code was written with the assumption that the backlash was measured in millimetres instead of steps. This meant that the command for moving forward had to be changed to a lowercase f instead of an uppercase one. Steps are also integers, so the values and increments needed to be as well. The process was changed to stop when increments would no longer be in integers or if the value had already been found.

#### Other calibrations

The other stages of the setup wizard were fairly straightforward to produce. They simply have a view with a picture and a button to draw the shape, then swap to a view with text inputs. If the correct value is not given, they calculate the needed change, then swap back to the draw view. If the correct value is given, then they move to the next stage.

Both the axle and wheel diameter values were found to need to be rounded. Otherwise, they were too large for the double size that the turtlebot can handle, leading to a “What?” message with the leftover numbers since they were split from the rest of the command.

## Curves

Near the end of the project, a new movement was introduced. This was to allow the turtlebot to draw a smooth curve and takes the arc length desired and the arc’s angle. To add this to the application a new function was added to the User\_Turtle class called curve which took in the values. A turtle graphics corresponding function then needed to be found, the closest of which was the circle function which has parameters of the radius of the circle and the angle. This meant that the radius has to be first calculated from the arc length and the angle then passed to the turtle graphics function.

Once the new command was working, the scaling needed to be worked out to keep it on the screen. To achieve scaling, the amount the curve would go in each direction was needed. The angle of the curve is split into 90° segments. Each segment is then converted into a line, which is the arc’s chord, to work out how far it travels, and the angle of that line is calculated. The normal scaling for forward motion is then done to divide it into vertical and horizontal components.

1. Testing

## How testing was approached

Testing was carried out via a manual testing table, which can be found in Appendix D. Manual testing was chosen due to the heavy UI element of the application, as well as monitoring that the turtlebot did the correct action.

Automated unit testing was considered. However, there are very few occasions when they are suitable for use in the project, so testing was done solely using manual testing. Very few of the functions within the application return anything, and when they do, it is generally a UI element which wouldn’t be suitable for Python’s unit testing. Instead, it is more sensible to manually test by seeing if it shows on the application correctly.

## Conclusions

All the tests carried out passed, as can be seen within the test table in the appendices. The only slight exception to this is the setup wizard, which most of the time displays the correct behaviour. On occasion, during the saving and closing portion, in particular, of the wizard, it causes the application to crash. It does manage to save the configurations to the turtlebot first, but the application cannot be used after crashing. This happens independent of the order of saving and making UI changes.

1. Packaging

## Packaging

The application was packed using the Auto Py to Exe library[17]. This was done to turn the project into a runnable executable with a directory structure easily. The directory is a requirement so that the additional files can be located. These files are the HTML file, the theme file, the graphics files and the insert files and all their respective directory structures.

One benefit of this is that the HTML file can be updated without having to repackage the application. Once the application is packaged it can be compressed and distributed to the devices it is required on.

To turn the project into an executable Auto Py to Exe was launched. From here a window is opened where the location of the main.py file can be specified along with the applications name and icon image (found in the graphics folder). The projects files and folders other than the Python ones need to be specified so they are included in the project. These folders are the characters, graphics and html\_info folders with the addition of the turtlebot\_theme.json file.

These configurations can be imported from the exe\_config.json file in the project’s directory. The project can then be converted. Once done, the earlier specified folders and file need to be taken out of the \_internal folder and put into the project folder.

## Using the application

The application has been primarily designed for use on Windows, and this will produce the best results. The application will work on macOS, but Tkinter is not as well designed for it so it makes components such as the top menu buttons display differently. The application has not been tested on Linux, so using it may lead to unexpected results.

The device used will require a USB port or another port with a converter for USB attached if the turtlebot is to be used. This is so the USB dongle for the Bluetooth connection can be plugged in.

1. Critical Evaluation

The critical evaluation consists of a discussion, leading to conclusion. It is an essential part of a master’s degree.

It shows that you can not only carry out a substantial piece of work, but that you can reflect on it, and think critically about how you might have done it better.

Examiners view the critical evaluation as very important.

Critical evaluation should contain.

* Strengths and weaknesses of your project
* If you were unable to attain any deliverables, then why.
* What are the future plans for your project if you are to continue

You will be presenting this during demonstration but here you need to discuss them in detail.

* Achieved aims and more.
* Some elements could be better e.g. scaling turtle
* Usable and those who’ve tried it enjoyed
* Future -integration of svg stuff?
* Library choices and whether worked well
* Changes to bot for curves
* I think port picking is a little faffy (button on menu rather than through settings?)

1. Conclusion

A brief summary of all that has gone before, including the key results of the project.

May also include some directions for future work.

# References

[1] S. Papert, “A COMPUTER LABORATORY FOR ELEMENTARY SCHOOLS,” MASSACHUSETTS INSTITUTE OF TECHNOLOGY A.I. LABORATORY, Oct. 1971. Accessed: Jul. 30, 2025. [Online]. Available: <http://hdl.handle.net/1721.1/5834>

[2] C. Solomon *et al.*, “History of Logo,” *Proceedings of the ACM on Programming Languages*, vol. 4, no. HOPL, pp. 1–66, Jun. 2020, doi: <https://doi.org/10.1145/3386329>.

[3] Python, “turtle — Turtle graphics — Python 3.7.4 documentation,” *Python.org*, 2019. <https://docs.python.org/3/library/turtle.html> (accessed Aug. 06, 2025).

[4] C. Solomon, “Logo Things,” logothings, 2016. <https://logothings.github.io/logothings/Home.html#HOME> (accessed Aug. 06, 2025).

[5] S. Papert and C. Solomon, “TWENTY THINGS TO DO WITH A COMPUTER,” pp. 1–5, Jun. 1971. Accessed: Aug. 06, 2025. [Online]. Available: <https://dspace.mit.edu/bitstream/handle/1721.1/5836/AIM-248.pdf?sequence=2&isAllowed=y>

[6] P. Boytchev, “About,” Uni-sofia.bg, 2015. <https://pavel.it.fmi.uni-sofia.bg/logotree/about.html> (accessed Aug. 06, 2025).

[7] Scratch, “Scratch - Imagine, Program, Share,” *Scratch*, 2007. <https://scratch.mit.edu/> (accessed Aug. 07, 2025).

[8] N. Fazakerley, “BeebControl’s BBC Buggy Page,” Riscy.uk, 2025. <http://www.riscy.uk/beebcontrol/buggies/bbc_buggy/index.html> (accessed Aug. 06, 2025).

[9] The Centre for Computing History “Valiant Turtle - Peripheral - Computing History,” *Computinghistory.org.uk*, 2025. <https://www.computinghistory.org.uk/det/39597/Valiant-Turtle/> (accessed Aug. 07, 2025).

[10] “LEGO® TC logo.” Accessed: Aug. 06, 2025. [Online]. Available: <https://ia802903.us.archive.org/4/items/lego-tc-logo-reference-guide/lego-tc-logo-reference-guide.pdf>

[11] “LEGO® MINDSTORMS® About | Official LEGO® Shop GB,” *www.lego.com*. <https://www.lego.com/en-gb/themes/mindstorms/about> (accessed Aug. 07, 2025).

[12] E. Ackerman, “TurtleBot Inventors Tell Us Everything About the Robot,” *IEEE Spectrum*, Mar. 26, 2013. <https://spectrum.ieee.org/interview-turtlebot-inventors-tell-us-everything-about-the-robot> (accessed Aug. 07, 2025).

[13] TurtleBot, “TurtleBot,” *www.turtlebot.com*. <https://www.turtlebot.com/> (accessed Aug. 07, 2025).

[14] Chris Liechti, “pySerial — pySerial 3.4 documentation,” *Readthedocs.io*, 2016. <https://pyserial.readthedocs.io/en/latest/pyserial.html> (accessed Aug. 26, 2025).

[15] T. Schimansky, “Color and Themes | CustomTkinter,” *Tomschimansky.com*, 2025. <https://customtkinter.tomschimansky.com/documentation/color> (accessed Aug. 08, 2025).

[16] GIMP, “GIMP,” *GIMP*, 2019. <https://www.gimp.org/> (accessed Aug. 26, 2025).

[17] brentvollebregt, “GitHub - brentvollebregt/auto-py-to-exe: Converts .py to .exe using a simple graphical interface,” *GitHub*, Feb. 24, 2025. <https://github.com/brentvollebregt/auto-py-to-exe> (accessed Aug. 26, 2025).

Appendix A

Generative AI

1. No AI was used for the project.

Third Party Code and Software Libraries

1. .NET’s ASP.NET libraries have been used for this project. The document template created by the dotnet CLI tool produces a set structure for the Model-View-Controller parts of the project. The CLI tools were also used to generate the Entity Framework Core code in most Controllers. The CLI generated code was then adjusted to make it relevant to this application.

ADD USER MANUAL TO APPENDICIES

Appendix B- Project Requirements

## Basic Requirements (BR)

*BR1: A user should have an application where they can write code.*

*BR2: A user should be able to view a turtle simulation to see the output of their code.*

*BR3: A user should be able to connect to and send commands to the turtlebot.*

*BR4: A user can configure the pen height*

*BR5: A user should be able to receive error feedback on their code*

## Additional Requirements (AR)

*AR1: A user should be able to stop the code when running*

*AR2: The user’s code input box should have syntax highlighting*

*AR3:A user should be able to access information pages*

*AR4: A user should be able to change the font size of the code text and output text*

*AR5: A user should be able to save their current code and load their code from a file*

*AR6: A user should be able to set up the configurations of the turtlebot so that the drawings will be precise.*

*AR7: A user should be able to insert premade characters and shapes.*

Appendix C- Turtlebot commands (created by Dr Neal Snooke)

# Turtle basics

In the following list \*d\* is a decimal value and \*n\* is an integer value.

## Turtle Commands

\* l\*n\* - left wheel \*n\* steps (4096 per wheel rotation)

\* r\*n\* - right wheel \*n\* steps (4096 per wheel rotation)

\* f\*n\* - forward \*n\* steps (-\*n\* backward)

\* t\*n\* - turn \*n\* steps (+ clockwise, -anticlockwise)

\* F\*d\* - forward \*d\* millimetres (-mm backward)

\* T\*d\* - turn \*d\* degrees (+ clockwise, -anticlockwise)

\* C\*d\* \*d\*- forward parameter 1 degrees and simultaneously turn parameter 2 degrees

\* a\*n\* - acceleration number of steps between slow and fast speed

\* o - motors off

\* u - pen up to max height

\* d - pen down to min height

\* U\*d\* - pen up to \*d\* height (0<\*d\*<=1, 0=u height, 1=d height). Note this does not persist after power down. Use s5 setting to set power up default.

\* U - pen up to previously set height

\* D\*d\* - pen down to d height (0<\*d\*<=1, 0=u height, 1=\*d\* height). Note this does not persist after power down. Use s6 setting to set power up default.

\* D - pen down to previously set height

\* =text - display text on OLED screen

\* x\*d\* run step streaming program \*d\*

\* K\*n\* set the acknowledgement counter (0 if no \*n\*) specified.

\* save - save the current set of configs to the EEPROM

\* get - return the list of configurations and their values

Note that the lower case commands f,t do not include backlash compensation and directly drive the motors. The normal commands (capital letters) do include backlash compensation (to reduce motor gear slop effects when motors change direction) if non zero compensation values are set.

## Settings

\* s1\*n\* - set motor speed in nS per step (1000 default)

\* s2\*d\* - set left wheel diameter in mm

\* s3\*d\* - set right wheel diameter in mm

\* s4\*d\* - set wheel spacing (axle length) in mm

\* s5\*d\* - set default 'U' command pen up position (0.1 - 1.0)

\* s6\*d\* - set default 'D' connamd pen down position (0.0 - 1.0)

\* s7\*i\* - set left wheel backlash (in motor steps)

\* s8\*i\* - set right wheel backlash (in motor steps)

###Permanent settings are

Settings values s2 - s8 are stored in EEPROM and are permanent after power cycling. They are removed after firmware update however.

#Java code

the following only applied to the application which intercepts these commands to perform special functions for the turtle. e.g. loading/creating a file that is then used to send the commands to the turtle.

## File commands

### send commands (ie. not step mode)

\* tc filename - send command file to turtle

\* tcsvg filename - send lines and turns to turtle

### stream steps (ie. wheel steps)

\* svg filename - stream file as lines and arc'd beziers

\* svgl filename - stream as lines and turns (segmented beziers)

\* svga filename - stream steps only arcs (except moves)

## Settings only for the java svg code

These are not part of the Turtle firmware, but are intercepted by the Java terminal application to set values used in the wheel step streaming protocol intended for drawing very complex curve based designs.

\* j3\*n\* - set STEP\\_DIFF for extra steps if L/R diff

\* j4\*n\* - set EXTRA\\_STEPS steps to add if STEP\\_DIFF occurs

## EEPROM

\* get - applies EEPROM settings and sends to the serial interface

\* save - saves the current settings to the EEPROM so they will be restored on power up. This is designed for initial configuration of a new turtle or after mechanical changes. Should not be needed for 'everyday' use. ge5

# Java code notes

The `CommandMaker` class main method creates the svg output for wheel paths (as a visualisation)

The Turtle class main method is the entry point for the terminal based Turtle controller

After updating the Turtle firmware issue a `tc cfg` command to transfer configs to the turtle and save in EEPROM.

The `svg` command series streams motor step commands to the turtle. The configurations are irrelevant here except for the `motorSpeedFast` which must match the java`Turtle.STEP\_TIME`. The Java code the generates the pattern must therefore have the correct wheel and axle lengths, as well as any backlash required. This is all done in the PatternMaker class using `stepsLinesAndCurveArcs` and related methods to convert an svg file to steps.

Since the PatternMaker needs the correct config for the turtle to be used it is run from `CommandMaker.main` which loads cfgJ.txt and then generates steps and visualisations for the input svg file.

Appendix D- Manual testing table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Ref | Test Content | Input | Output | Pass Criteria | Pass/ Fail |
| T-01 | Does the application start | Click on the Turtlebot.exe | The application opens with the default values and layout as shown: | No port is currently displayed in the top right, along with two red boxes. The default code of “turtle.down()  turtle.forward(20)  turtle.right(90)  turtle.up()  turtle.forward(20)” is in the textbox | **Pass** |
| T-02 | Can the default code be run on the simulation? | Click on run after launching the application | The right-hand side shows the turtle drawing and the box underneath the run button displays the commands in the input screen as they are carried out |  | **Pass** |
| T-03 | Can the virtual turtle be reset? | Click on the Reset button in the turtle box’s top right corner after state left by T-02 | The turtle returns to its default position | No more lines are visible | **Pass** |
| T-04 | Does a dialogue appear when trying to clear the program? | After the state left by T-02, click the “Clear Program” button in the top right of the code box. | A confirmation dialogue is shown asking “Are you sure you want to clear the program?” | The user is presented with two buttons on the dialogue, one for Yes and one for No | **Pass** |
| T-05 | Can the program be cleared? | After T-04, click Yes | The dialogue box disappears, and the code box is now empty | The turtle is reset, and no drawings can be seen. The output box is cleared | **Pass** |
| T-06 | Can clearing the program be cancelled? | After T-04, click No | The dialogue box disappears. | No other change occurs | **Pass** |
| T-07 | Can a pop-up be opened to select a port? - no ports available | After launching the application:   1. Click settings. 2. Click Select port when no USB ports are available | A dropdown is shown with “No ports” and a Refresh ports button | Shows until Refresh ports is pressed | **Pass** |
| T-08 | Can refresh be pressed to see new available ports? | After T-07:   1. Insert the USB dongle. 2. Click Refresh ports | Select port is now shown on dropdown | Ports are available if click on dropdown | **Pass** |
| T-09 | Can a pop up be opened to select a port? - ports available | After launching the application:   1. Click settings. 2. Click Select port when USB ports are available | A pop-up containing dropdown is shown with “Select port” and a Refresh ports button | Ports are available if click on dropdown | **Pass** |
| T-10 | Can a port be selected? | After the state of either T-08 or 09:   1. Click the dropdown. 2. Click on COM3 | The popup disappears, COM3 is displayed in the white box on the top bar | The red box next to the word USB in the top bar turns green | **Pass** |
| T-11 | Can the application show that the USB connection has been stopped? | After T-10, unplug the USB dongle | The green box next to the word USB in the top bar turns red | The box next to Turtle is also red | **Pass** |
| T-12 | Can the turtlebot be connected? | After T-10, press the button on the turtlebot on the right of the display | The box next to the word Turtle turns green from red. | The turtlebot displays the word “Hello” | **Pass** |
| T-13 | Can the turtle move the pen down? | Following the state after T-12:   1. Type “turtle. down()” in the code box 2. Press run. | The turtlebot moves its pen down, and “turtle.down()” is displayed in the output box. | No visual differences to the virtual turtle. | **Pass** |
| T-14 | Can the turtle move forward? | Following T-13:   1. Type “turtle. forward(20)” 2. Press run | The virtual turtle moves forward and “turtle.forward(20)” is also displayed in the output box. | The turtlebot moves forward 20mm. | **Pass** |
| T-15 | Can the turtle move forward? | Following T-13:   1. Type “turtle. forward(55)” 2. Press run | The virtual turtle moves forward and “turtle.forward(55)” is also displayed in the output box. | The turtlebot moves forward 55mm. | **Pass** |
| T-16 | Are spelling errors for the turtle caught? | Following T-13:   1. Type “turte. forward(20)” 2. Press run. | The output box displays “name 'turte' is not defined” | Nothing else happens | **Pass** |
| T-17 | Are spelling errors for methods caught? | Following T-13:   1. Type “turtle. foward(20)” 2. Press run | The output box displays “'User\_Turtle' object has no attribute 'foward'” | Nothing else happens | **Pass** |
| T-18 | Are syntax errors caught? | Following T-13:   1. Type “turtle. forward(20” 2. Press run | The output box displays “'(' was never closed (<string>, line 2)” | Nothing else happens | **Pass** |
| T-19 | Can the turtle move backwards? | Following T-13:   1. Type “turtle. forward(-20)” 2. Press run | The virtual turtle moves backwards, “turtle.forward(-20)” is displayed in the output box. | The turtlebot moves backwards 20mm. | **Pass** |
| T-20 | Can the turtle move the pen up? | Following T-12:   1. Type “turtle.up()” 2. Press run | The turtlebot moves its pen up and, “turtle.up()” is displayed in the output box. | No visual differences to the virtual turtle. | **Pass** |
| T-21 | Can the turtle move 90° right? | Following T-12:   1. Type “turtle.right(90)” 2. Press run | The turtlebot turns 90° to the right and, “turtle.right(90)” is displayed in the output box. | The virtual turtle also turns right so it is now facing south. | **Pass** |
| T-22 | Can the turtle move 180° right? | Following T-12:   1. Type “turtle.right(180)” 2. Press run | The turtlebot turns 180° to the right and, “turtle.right(180)” is displayed in the output box. | The virtual turtle also turns right so it is now facing West. | **Pass** |
| T-23 | Can the turtle move 360° right? | Following T-12:   1. Type “turtle.right(360)” 2. Press run | The turtlebot turns 360° to the right and, “turtle.right(360)” is displayed in the output box. | The virtual turtle also turns right so it is now facing East. | **Pass** |
| T-24 | Can a negative be input for right? | Following T-12:   1. Type “turtle.right(-90)” 2. Press run | The turtlebot turns 90° to the left and, “turtle.right(-90)” is displayed in the output box. | The virtual turtle also turns left so it is now facing North. | **Pass** |
| T-25 | Can the turtle move 90° left? | Following T-12:   1. Type “turtle.left(90)” 2. Press run | The turtlebot turns 90° to the left, and “turtle.left(90)” is displayed in the output box. | The virtual turtle also turns left so it is now facing north. | **Pass** |
| T-26 | Can the turtle move 180° left? | Following T-12:   1. Type “turtle.left(180)” 2. Press run | The turtlebot turns 180° to the left, and “turtle.left(180)” is displayed in the output box. | The virtual turtle also turns left so it is now facing West. | **Pass** |
| T-27 | Can the turtle move 360° left? | Following T-12:   1. Type “turtle.left(360)” 2. Press run | The turtlebot turns 360° to the left, and “turtle.left(360)” is displayed in the output box. | The virtual turtle also turns left so it is now facing East, its original position. | **Pass** |
| T-28 | Can a negative be input for left? | Following T-12:   1. Type “turtle.left(-90)” 2. Press run | The turtlebot turns 90° to the right, and “turtle.left(-90)” is displayed in the output box. | The virtual turtle also turns right, so it is now facing South. | **Pass** |
| T-29 | Can the turtle move in using the curve function? | Following T-12:   1. Type “turtle.curve(100, 60)” 2. Press run | Arc drawn with a chord length of about 96mm, and “turtle.curve(100, 60)” is displayed in the output box. | The virtual turtle also draws an arc mirroring the amount turned by the turtlebot. | **Pass** |
| T-30 | Can semicircles be drawn? | Following T-13:   1. Type “turtle.curve(200, 180)” 2. Press run | Arc drawn of a chord length of about 127.5mm, and “turtle.curve(200, 180)” is displayed in the output box. | The virtual turtle also draws an arc mirroring the amount turned by the turtlebot. | **Pass** |
| T-31 | Can the turtle do a turn using the curve function? | Following T-13:   1. Type “turtle.curve(0, 360)” 2. Press run | The turtlebot does a full turn clockwise, and “turtle.curve(0, 360)” is displayed in the output box. | The virtual turtle also turns in a circle clockwise, so it is now facing East. | **Pass** |
| T-32 | Can the turtle move in a straight line using the curve function? | Following T-13:   1. Type “turtle.curve(180, 0)” 2. Press run | The turtlebot moves forward 180mm and “turtle.curve(180, 0)” is displayed in the output box. | The virtual turtle moves forward. | **Pass** |
| T-33 | Can zero values be entered for the curve function? | Following T-13:   1. Type “turtle.curve(0, 0)” 2. Press run | The turtlebot does not move, and “turtle.curve(0, 0)” is displayed in the output box. | The virtual turtle also does not move. | **Pass** |
| T-34 | Can a variable be used for a forward value? | Following T-12:   1. Type “x=50” 2. Add a new line. 3. Type “turtle.forward(x)” 4. Press run | The virtual turtle moves forward, “turtle.forward(50)” is displayed in the output box. | The turtlebot moves forward 50mm. | **Pass** |
| T-35 | Can a variable be used for a right value? | Following T-12:   1. Type “x=180” 2. Add a new line. 3. Type “turtle.right(x)” 4. Press run | The turtlebot turns 180° to the right and, “turtle.right(180)” is displayed in the output box. | The virtual turtle also turns right so it is now facing west. | **Pass** |
| T-36 | Can a variable be used for a left value? | Following T-12:   1. Type “x=180” 2. Add a new line. 3. Type “turtle.left(x)” 4. Press run | The turtlebot turns 180° to the left and, “turtle.left(180)” is displayed in the output box. | The virtual turtle also turns left so it is now facing west. | **Pass** |
| T-37 | Can for loops be used? | Following T-13:   1. Type “for i in range(4):” 2. Add a new line. 3. Add a tab. 4. Type “turtle.forward(20)” 5. Add a new line. 6. Add a tab. 7. Type “turtle.right(90)” 8. Press run | The turtlebot draws a square of dimensions of 20mm by 20mm. The output box displays “turtle.down()  turtle.forward(20)  turtle.right(90)  turtle.forward(20)  turtle.right(90)  turtle.forward(20)  turtle.right(90)  turtle.forward(20)  turtle.right(90)” | The virtual turtle draws a square: | **Pass** |
| T-38 | Are indent errors caught? | Following T-13:   1. Type “for i in range(4):” 2. Add a new line. 3. Type “turtle.forward(20)” 4. Add a new line. 5. Type “turtle.right(90)” 6. Press run | The output box displays “expected an indented block after 'for' statement on line 2 (<string>, line 3)” | Nothing else happens | **Pass** |
| T-39 | Can the turtle be stopped? | Straight after T-37 press stop | The turtlebot stops after the current command. The output box says stop after the current command. | The virtual turtle stops after the current command. Nothing else is added to the output box. The run button is available to click again. | **Pass** |
| T-40 | Can a Save As dialogue be opened | Following T-15:   1. Click on File on the top bar 2. Click Save | A pop-up appears with the option to enter a filename and save and cancel buttons | Other saving locations can be navigated to within the pop up | **Pass** |
| T-41 | Can the current code be saved? | Following T-40:   1. Enter “my\_code” as the filename 2. Click Save | The pop-up closes | A text file named my\_code is created containing:  “turtle.down()  turtle.forward(55)” | **Pass** |
| T-42 | Can saving the file be cancelled | Following T-40:   1. Click Cancel | The pop-up closes | No new file is created | **Pass** |
| T-43 | Can code be loaded? | Following T-41:   1. Click on clear program 2. Click on File on the top bar 3. Click Load 4. Select the file named my\_code 5. Click open | The code text box contents get replaced with the code:  turtle.down()  turtle.forward(55) | The pop-up closes | **Pass** |
| T-44 | Can loading code from a file be cancelled? | Following T-41:   1. Click on File on the top bar 2. Click Load 3. Click cancel | The pop-up closes | Nothing else changes | **Pass** |
| T-45 | Can a number be inserted | Following T-12:   1. Click on Insert on the top bar 2. Hover over Numbers 3. Click 6 | Code that produces the number six gets inserted into the code text box. | The code is inserted where the cursor is | **Pass** |
| T-46 | Can the font size be increased? | Following T-12:   1. Click on Settings on the top bar 2. Hover over Font size 3. Click 32 | Text in the code textbox and the output text box increase in size | Nothing else | **Pass** |
| T-47 | Can the font size be decreased? | Following T-46:   1. Click on Settings on the top bar 2. Hover over Font size 3. Click 14 | Text in the code textbox and the output text box increase in size | Nothing else | **Pass** |
| T-48 | Can the pop-up for changing the pen height be opened? | Following T-12:   1. Click on Settings on the top bar 2. Click on pen height | A pop up with a slider and save button appears | The main application can not be accessed whilst the pop up is open | **Pass** |
| T-49 | Can the pen height be decreased? | Following T-48:   1. Click on a lower slider value | The turtlebot pen moves down | The slider dot stops where it is dragged to | **Pass** |
| T-50 | Can the pen height be increased? | Following T-48:   1. Click on a higher slider value | The turtlebot pen moves up | The slider dot stops where it is dragged to | **Pass** |
| T-51 | Can the setup wizard be done when no turtlebot is connected? | Following the state after T-01:   1. Click on Settings on the top bar 2. Click on Setup Wizard | Pop up opens telling the user to connect the turtlebot first with a close button to close the pop up |  | **Pass** |
| T-52 | Can the setup wizard be done when the turtlebot is connected? | Following T-12:   1. Click on Settings on the top bar 2. Click on Setup Wizard | Pop up opens displaying the required equipment and a start button |  | **Pass** |
| T-53 | Does the backlash process start correctly? | Following T-52, click Start | View changes to display two buttons and the current backlash steps | Turtlebot moves backwards then forwards by a smaller amount | **Pass** |
| T-54 | Does the setup wizard act correctly when told the turtlebot moved? | Following T-53, click Moved | The steps change to say 25, and the reset button becomes clickable | Turtlebot moves backwards, then forwards by a smaller amount | **Pass** |
| T-55 | Does the setup wizard move correctly with no difference clicked? | Following T-54, click No difference | The steps change to say 50. | Turtlebot moves forward | **Pass** |
| T-56 | Does the setup wizard draw the line and display the correct view for the wheel diameters? | Follow the steps after T-55 until the window changes to say “Setup Turtlebot: Wheels”, click Draw | View changes to have an area to input a length and a next button | The turtlebot moves forwards, drop the pen, draws a line and lifts the pen | **Pass** |
| T-57 | Are shorter values for wheel diameters handled correctly? | Following the state after T-56:   1. Enter 295 into the text field 2. Click Next | View goes back to previous screen | When the turtlebot next draws the line it is longer than the previous one | **Pass** |
| T-58 | Are longer values for wheel diameters handled correctly? | Following the state after T-56:   1. Enter 305 into the text field 2. Click Next | View goes back to previous screen | When the turtlebot next draws the line it is shorter than the previous one | **Pass** |
| T-59 | Are expected values for wheel diameters handled correctly? | Following the state after T-56:   1. Enter 300 into the text field 2. Click Next | View goes onto the axle length view | Nothing else | **Pass** |
| T-60 | Does the setup wizard draw the shapes and display the correct view for the axle length? | Following the state after T-59, click Draw | View changes to have an area to input two lengths and a next button | The turtlebot draws two circles, one with each wheel. | **Pass** |
| T-61 | Are gap values for axle length handled correctly? | Following the state after T-60:   1. Enter -2 into both boxes 2. Click Done | View goes back to previous screen | When the turtlebot next draws the shape the circles are drawn 2mm more | **Pass** |
| T-62 | Are overlap values for axle length handled correctly? | Following the state after T-60:   1. Enter 2 into both boxes 2. Click Done | View goes back to previous screen | When the turtlebot next draws the shape the circles are drawn 2mm less | **Pass** |
| T-63 | Are expected values for axle length handled correctly? | Following the state after T-60:   1. Enter 0 into both boxes 2. Click Done |  | The calibrated values are saved. Turtlebot will display save | **Pass\*** |
| T-64 | About | After T-01, click on the About button on the top bar | A window opens with additional information which matches the layout defined in the HTML file. | The main window can be used (clicked on and typed on) whilst the About window is open | **Pass** |

**\***During the main testing, this passed successfully. This has been known to crash at other times when using the system, though additional pass criteria is met