Spacecraft flightpath planner

Euan Johnston

MSc Applied Computing, University of Dundee

Supervisor - Dr Iain Martin

Submitted on 21/09/2020

Abstract

PANGU is a software application critical to the development of vision-based spacecraft landers. It is used by organisations like the ESA to create simulations of planetary surfaces as viewed from a spacecraft on its approach. One method of mapping flightpath trajectories is by using a flight file which specifies the spacecrafts position in 3D space at different time steps. PANGU currently has a viewer GUI to observe models and a surface modeller GUI to create models but has no GUI to operate flight files and show the images generated from those files. The purpose of this project has been to create a new GUI that will allow a user to upload and edit flight files in addition to giving a model view of the spacecraft as it moves along in its flightpath. Most importantly the spacecraft flightpath planner GUI v1.0 shows the planetary surface images generated from the cameras position on the spacecraft. The benefit of this is that a user can view the images created to make sure that they satisfy the requirements for resolution and detail and save only the images a user wants. This first version of the flightpath planner GUI v1.0 is not yet the complete product but offers a substantial improvement on the current cumbersome process of generating images from flight files. Future revisions of the GUI may include improved editing capabilities, a 3D model view of the spacecraft, support for more flight file commands and SPICE ephemerides systems.

"I certify that Euan Johnston has satisfied the conditions of the Ordinance and Regulations and is qualified to submit this dissertation in application for the degree of Master of Science."

Acknowledgements

I would like to dedicate this project to my Granny, Katherine Fairlie who I sadly lost right at the beginning of this project at the end of May. Without her support from the moment I was born I probably would not be the person I am today; she was always there for me when I needed her, and she still is now. I miss you. I would like to thank my long-time partner Martha who has supported me for the now 14 years that we have been together. I know I have been difficult for the last few months, but we got through it and we will move on to a positive future together. I also want to thank my parents Neil and Sharon Johnston for encouraging me and supporting me through a third degree! Without their moral and financial support, I probably would not have gone back into academia or completed this course. Thank you to my best friend Brendan who I have known for 20 years now, Brendan has always made himself available to help when I needed it and he is one of the most intelligent people I know. Finally, I would like to thank my supervisor Dr Iain Martin, who gave me an interesting and challenging project to work on and was always patient when I did not understand something first time around!

Table of Contents

1. List of Figures

2. Introduction

The main objective of the spacecraft flightpath planner project is to build a graphical user interface (GUI) tool for a software package called PANGU (Planetary and Asteroid Natural Scene Generation Utility). PANGU was developed by the Space Systems Research group of the European Space Agency (ESA), and was created with the purpose of supporting the development of landing spacecraft reliant on computer vision on planetary bodies from planets like Mercury or Mars, to its moons Phobos and Deimos or smaller rocky asteroids in our solar system [1]. PANGU already has a viewer GUI tool which a user can interact with to view models of planetary surfaces and a surface modeller GUI which can be used to generate models.

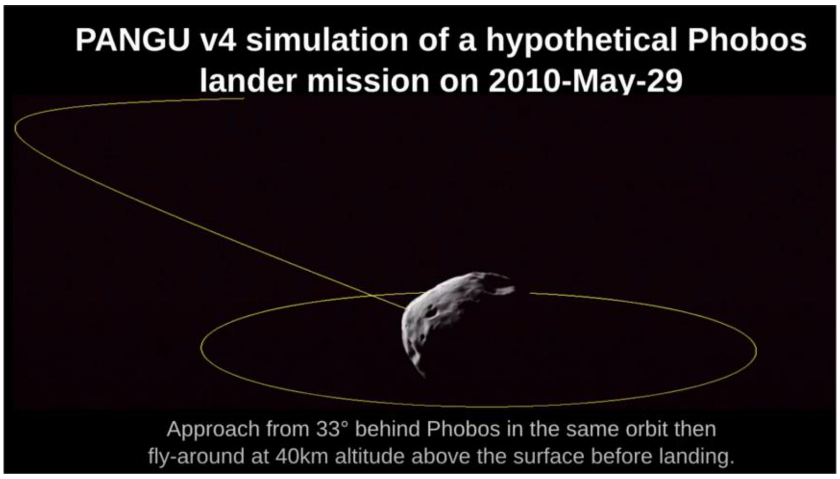

PANGU v4 simulation of a hypothetical Phobos lander mission on 2010-May-29
Approach from 33° behind Phobos in the same orbit then fly-around at 40km altitude above the surface before landing.

Figure 1 *– Shows a hypothetical flightpath of a lander mission on Phobos, one of Mars two moons. The lander mission is simulated as if it had taken place on the 29th of May 2010. The yellow line represents the flightpath of the spacecraft from where the camera renders the planetary surface based on the model parameters defined in the PANGU software package* [7]

2.1 Outline of the problem

The problem is that there is no GUI tool included in the software that allows a user to specify a flightpath for a spacecraft on its approach to the surface of a planetary body whilst showing the images being rendered by the camera on the spacecraft. The way the software currently operates is that a user will run the viewer tool in movie mode with a flight file and images will be automatically generated and saved to a folder called frames, which is not ideal.

The flightpaths of a spacecraft are currently defined using a .fli file, which is essentially a text file that allows a user to specify the position of the spacecraft and the cameras orientation. A flight file may also include other commands such as the ability to change the suns position or set the time. In most simple cases a simulation may not care about time as a variable but for more advanced cases a user may need to work with the true positions of objects in the solar system in real time (known as the ephemeris). PANGU can make use of ephemeris files which can be used to specify some property of an object at some time. In addition, PANGU has the capability of using SPICE (Spacecraft Planet Instrument C-matrix Events) data. SPICE data gives an accurate spatial position of planetary bodies as a function of time and is used widely within the larger space exploration community [2]. The project at hand will focus on using basic flight files, although there should be scope for a future project to further build on and to incorporate the use flight files that use SPICE data too.

2.2 Purpose of the new GUI tool

The purpose of this new GUI tool is to a allow a user to create simulations of a realistic flightpath on its approach to the surface of a planetary body. This new GUI should have the power to allow a user to upload and edit .fli files, view the trajectory information of the space craft on screen (model view) that corresponds to the input data given from the flight file and view the images generated from the spacecrafts view. With this information a user can specify changes that they would want to make to a model based on the resulting images created from the simulation. This will hopefully aid users of PANGU in running simulations using flight files and make sure that the generated images are of an appropriate resolution for the model. The aim is to simplify the process and make it easier for users of PANGU to create flight paths and generate planetary images appropriate for their simulation.

3. Background

This chapter will give a more detailed overview of the PANGU software application and how it can be used to create models of planetary surfaces. It will also cover how PANGU can be used in conjunction with a client application like the GUI that we will be creating as part of this project. We conclude by covering the technologies and programming language that will be used to create the new tool.

3.1 Overview of the PANGU software application

Historically space lander missions were required only to land safely and protect the payload on a planetary surface to within reasonable proximity of a designated target [3]. These locations were chosen because they were broad, open spaces with relatively few boulders or obstacles for a landing spacecraft. As new missions required greater accuracy of landing gear and as computing power improved towards the turn of the 21st century computer vision-based landing systems had become more feasible [1]. The ESA built 3D planetary models based on digital elevation model (DEM) data taken from previous space missions, but these models did not initially meet all the needs of the sensors used in lander missions [1]. This led to development of PANGU which was created with the concept of aiding these computer vision based lander missions by generating artificial models of planetary surfaces that included boulders, craters and any other obstacles that may obstruct the safe landing of a spacecraft. The advantage of using PANGU is that a computer simulation can cover a wide range of scenarios, sensor measurements are noise free and it can also include fog and dust as well as shadow modelling meaning that the software has the ability to cover many different environmental conditions.

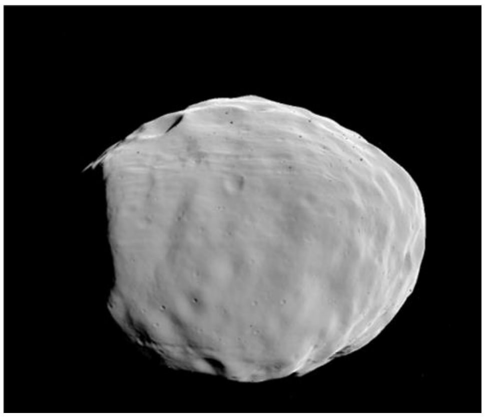


Figure 2 *– This is a model of Phobos as shown in the simulation from* **figure 1**. *There are several versions of this model built into the software, this one is known as Phobos 4 as it has 4 times the resolution of the original model. This version also includes visible cratering and small boulders which are just visible at this distance.*

3.2 The creation of a PANGU model

PANGU can generate planetary surfaces using both real and synthetic data by incorporating DEM data from previous space missions such as MOLA (Mars Orbiting Laser Altimeter) which took place during the late 1990's [4]. MOLA is a measuring instrument which over the course of four and a half years successfully mapped the elevation features of the surface of Mars by sending light signals to the surface which were then reflected to the sensor. The time it took for the light beam to traverse this distance was then used to determine the surface elevation. The higher the resolution of the data then the more accurate of a simulation that can be created, ultimately if the resolution of the data is good then it can be used to generate a complete model of a planetary surface. If the resolution of this data is not particularly high, then a combination of real and synthetic data can be used to complete the model. PANGU uses intelligent fractal interpolation of the data using several different methods as fractals are regarded as the best way to simulate a natural landscape [5].

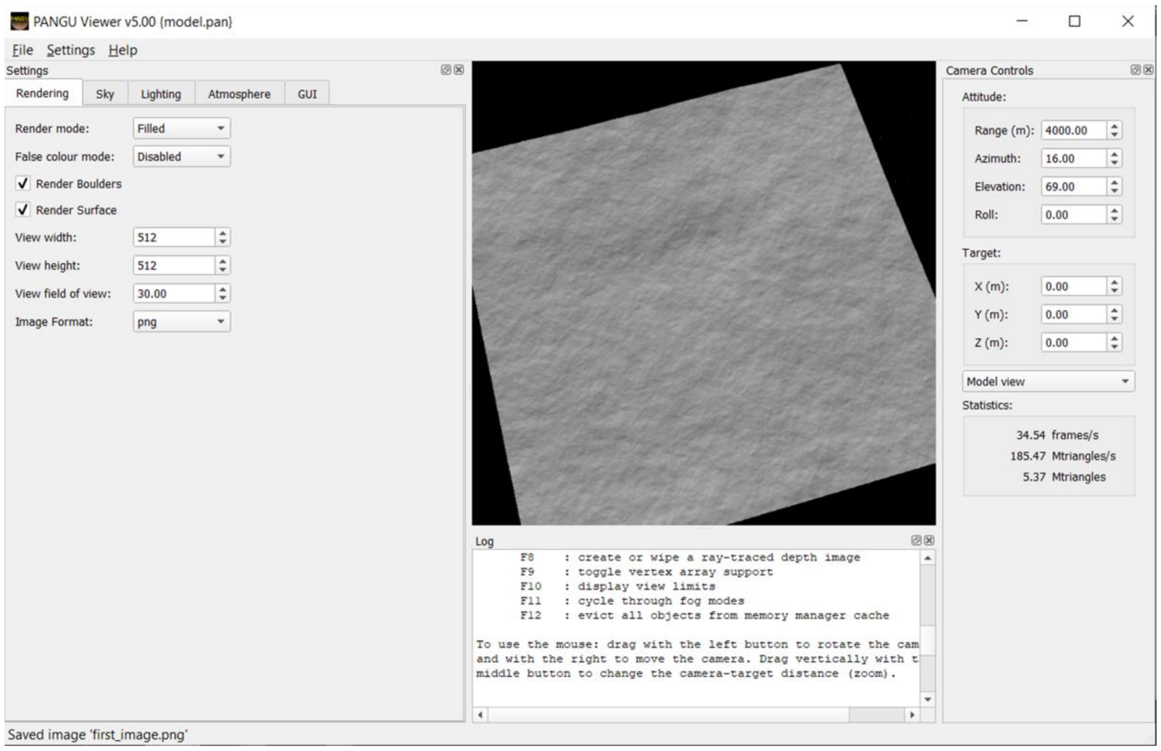
PANGU begins creating a surface model by starting with an initially smooth grid which is distorted to represent the elevated surface terrain with either the synthetic or real DEM data. Once the basic planetary surface is created other morphological features can be added such as craters, boulders, sand dunes and lava flows [3]. The craters are built using fractals and modelled on the existing scientific literature about their shape and formation, craters can be either added manually or spread out using a statistical distribution to simulate what we know about a particular planet's surface morphology [3]

PANGU models are built using different text files which are used to specify model parameters and a batch file usually named '*runme.bat*' for Windows users or '*runme.sh'* for Linux users is used to execute the model and open the viewer application.

3.3 About the viewer tool and observing a model as a user

The viewer tool is the main GUI of PANGU for observing the models that a user creates in addition to saving images of a model. The viewer is also useful for making sure that a model has a high enough resolution and that morphological features appear as expected given the input parameters specified in the text files.

The viewer tool consists of four main sections (see ***Figure 3***.) labelled as the settings, log, camera controls and statistics. The settings section consists of four tabs that allow a user to change the rendering mode, change the colour of the sky, change the position of the sun through the lighting tab, and add atmospheric effects although costly computationally. The final GUI tab just allows a user to specify the sensitivity of the camera's movements. The log section lists some key keyboard commands and specifies any changes that a user makes in the viewer window, this information is then saved to a log file within the directory of the model that is being inspected. The camera controls section is self-explanatory, and the statistics just gives some performance statistics of the model. Part of the process of this project will be to familiarise myself with the PANGU software package, thankfully the program comes with a user manual that explains how to use the software and includes some tutorials that will allow give me in experience in using the viewer tool.



Render mode:	Filled
False colour mode:	Disabled
✓ Render Boulders	
✓ Render Surface	
View width:	512
View height:	512
View field of view:	30.00
Image Format:	png

Range (m):	4000.00
Azimuth:	16.00
Elevation:	69.00
Roll:	0.00
X (m):	0.00
Y (m):	0.00
Z (m):	0.00

Model view

```
34.54  frames/s
185.47  Mtriangles/s
5.37  Mtriangles
```

```
      F8    : create or wipe a ray-traced depth image
      F9    : toggle vertex array support
      F10   : display view limits
      F11   : cycle through fog modes
      F12   : evict all objects from memory manager cache

To use the mouse: drag with the left button to rotate the cam
and with the right to move the camera. Drag vertically with t
middle button to change the camera-target distance (zoom).
```

Saved image 'first_image.png'

Figure 3 *– Shows the viewer tool in action. The viewer tool is the main GUI for PANGU and is used to observe planetary models and save images from a model. A user also has the options change camera settings, adjust the lighting or other settings through this tool.*

The viewer in PANGU was created using a platform called Qt which is an open source, cross-platform application that runs on Windows, MacOS, Linux and Android. Qt supports the use of different programming languages such as C++, Python or JavaScript. It makes sense for the purpose of continuity that any new GUI tool such as the flightpath planner we intend to build is done so using the Qt framework. Qt is free to download and as my most recent experience in programming is using the C++ programming language it is the most logical option moving forward into development. This will mean that I am required to learn how to use Qt, learn how to use its built-in classes, functions and interact with its user interface.

3.4 PANGU client applications and sockets

For PANGU to be able to send and receive information to a new GUI tool, the server (PANGU) must connect to the client (our GUI) through a TCP/IP socket connection. Since both client and server will be running on a local machine, the address can be either "*localhost*" or the equivalent IP address of 127.0.0.1. Thankfully PANGU provides a user manual that describes how to setup a client application. In addition to some classes and header files written in C that a user can incorporate into their project to create a socket connection between the client and server applications.

The documentation that comes with PANGU outlines how to set up this connection by first making sure that you incorporate the following header files, into your program.

```
#include "socket_stuff.h"
#include "pan_protocol_lib.h"
```

socket_stuff.h provides a uniform interface for socket functions for both UNIX and Windows, the second header file gives a user access to the `pan_protocol_lib` class which provides the user with a large set of functions that they can use to interact with PANGU. The `get_and_save_image()` function for example will do exactly as the function states and provide a user with an image of the model they are viewing and will save the image to some location on their computer. There are hundreds of other built in functions that a user can incorporate into their project if they so wish but for the GUI application I will try and build, using the `get_and_save_image()` will go a long way to towards completing the project.

3.5 Background Summary

In summary I think it is quite straight forward to justify the need to create a GUI for a spacecraft flightpath planner. PANGU Is a very niche software application that is designed for a very specific purpose of generating models of planetary surfaces for testing vision-based spacecraft landers. There is currently no GUI tool available with the software that allows a user to easily view images generated from the camera position on a flightpath. In addition, it is very difficult to simply look at a text file of flightpath data and visualise what is happening as the position of the spacecraft changes relative to its target. This tool will make it much easier for a user to open a flightpath file, view or change the data in that file and see the change in position of the spacecraft as it moves through the simulation. In addition, model images should be generated in real time on the screen and can easily be visually associated with the coordinates on a flight path as opposed to be saved directly to a folder on the user's hard drive.

I have also concluded that the GUI should be built using the Qt application platform for a couple of different reasons. The first is that the viewer tool in PANGU was created using this platform and it makes sense as a matter of consistency. Secondly that my experience in programming through this course has mainly been in C++ and as a programmer it is the language I am most comfortable working in, fortunately Qt is a platform that supports the use of C++ and is the standard to build a GUI using this programming language. I also considered using Java and the JavaFX library to build the GUI as I do have previous experience using the language but because of the aforementioned reasons I decided that using C++ and Qt was the best option.

4. Specification

The specification will outline the project schedule from beginning to end and cover the learning process of getting to grips with Qt and a lesser extent PANGU. The chapter will also outline the requirements for the system before moving into the design and development phase.

4.1 Project plan and overview

Due to some unforeseen circumstances this year and for other personal reasons the beginning of the project was delayed, and it was not until the 8th of June that I had my first meeting with my supervisor Dr Iain Martin. In this first meeting we agreed to continue by having a meeting at 1pm on each Monday, we also discussed an overview of the project and what the potential choices would be going forward. An overall project plan, with time scale can be found in the appendix in the form of a Gantt chart which details how the project was broken down into different areas. Areas such as learning the required tools to build the GUI and the PANGU software application itself, prototyping, and producing the final deliverables which are a video demonstration of the software in action and this report.

Since the timescale for which I could work on the project was reduced, so the scope of the project was also reduced. Initially part of the plan was also to make use of OpenGL which is a graphics API used for rendering both 2D and 3D vector graphics, but this had to be dropped due to the time constraints.

4.2 Learning PANGU

Dr Martin sent me a license file so that I could set up and install PANGU on my local machine. PANGU is quite a computationally expensive program to run requiring as much RAM as is possible and a fast, single, discrete graphics card. There was a concern that my machine would not be able to process some of the models as it is 5 years old, but the concern turned out to be mostly unwarranted. PANGU can operate on both Linux and Windows operating systems and on my current laptop I have Windows 10 running as my main OS with the option of using Ubuntu via virtual machine. I opted to install PANGU on the Windows side as it would require less from my hardware and the Linux installation process is slightly more complex.

The initial plan and schedule for the project would be that I would spend the next month or so learning how to use PANGU. PANGU comes with a documents folder and within that folder is a user manual that contains the full documentation for installing the software, using the viewer, surface modelling and has different case studies to look at.

4.3 Learning Qt

One of my first tasks was to learn how to use Qt, the software can be downloaded for free from their website but requires that you give some personal information as part of the registration process. Once I had downloaded and installed the software, I began to look for resources online that would help me get to grips with the platform.

One of my favourite resources is Udemy which usually provides some good video lectures at a reasonable price (usually £11.99). I signed up for three different courses which included;

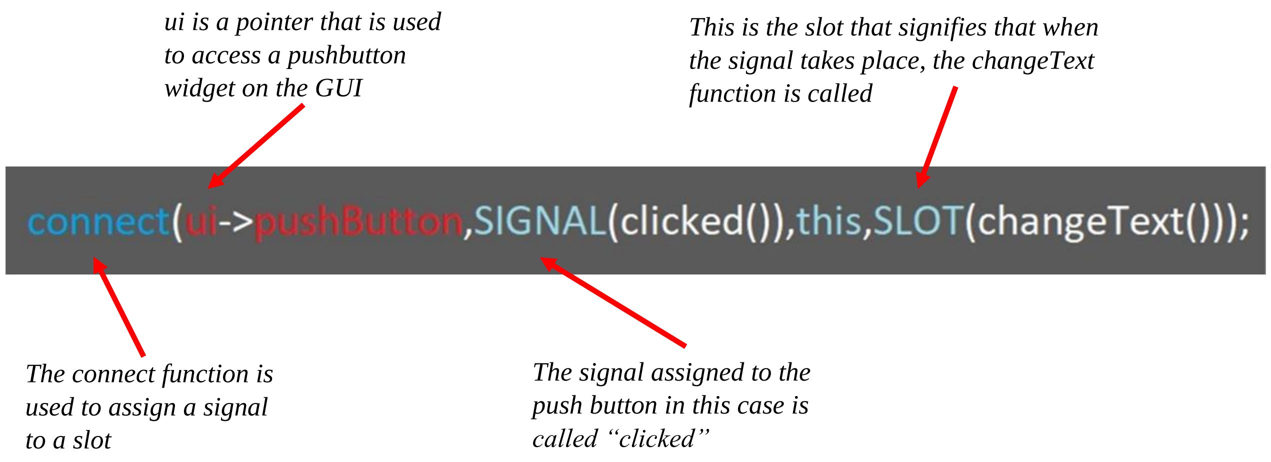
- *Qt 5 C++ GUI development for beginners: The fundamentals*
- *Qt C++ GUI Development – Intermediate*
- *Robust Qt & C++ GUI Programming a 2D Graphics application*

I started with the beginner course which covers areas such as C++ fundamentals, signals and slots, Qt widgets, Dialogs, resource files, working with files and directories and some other material that was not necessarily relevant to the project. Learning the fundamentals of Qt later led to developing a couple of simple prototypes which provide some of the functional that I am looking to use as part of the final product. The development of the prototypes are covered in section 5. As part of the design stage.

4.3.1 Signals and slots

Qt describes signals and slots as being the method by which objects communicate with one another. The best way to think about a signal is as an *event* that happens or is triggered by something that you do. The slot is a function that gets called because of the event being triggered. The simplest example of the signal and slot mechanism is that of a button being pushed on an interface and some function being called that perhaps opens a new window.

There are different notation methods for applying a signal and slot but here is one example using the string notation. But it is a fundamental process that is required to be able to really do anything of any significance on the platform.



```
connect(ui->pushButton,SIGNAL(clicked()),this,SLOT(changeText()));
```

ui is a pointer that is used to access a pushbutton widget on the GUI
This is the slot that signifies that when the signal takes place, the changeText function is called
The connect function is used to assign a signal to a slot
The signal assigned to the push button in this case is called "clicked"

Figure 4 *– a line of code that outline the signal and slot mechanism in Qt, an integral part of the functionality of the platform*

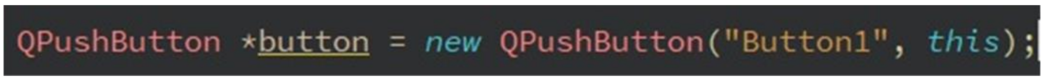
4.3.2 Qt Widgets

Widgets are the main component that make up the interface of a GUI built in Qt, each widget has its own class and functions which can be applied to it. A widget can be used to display data, take user input, or provide the functionality for some action, they can receive information from a mouse or keyboard or other events. Every single widget in Qt is a subclass of the parent class QWidget which is used to render the widget to the UI. Even the main window itself is a widget in Qt and is considered the parent of any widgets that appear within its frame.

It is also important to note at this point that there are a couple of different ways of incorporating widgets into the user interface. You can code all the widgets in manually of your own accord but Qt also provides a very useful UI of its own where you can very simply drag and drop the components you would like to add onto the form which is a *.ui* file in the project.

Some of the main widgets that we would like to incorporate into our GUI include the following

QPushButton



```
QPushButton *button = new QPushButton("Button1", this);
```

Figure 5 *– This is the code that is used to generate a button widget on the UI. Other widgets follow a similar format but may have differing constructor parameters depending on the type of widget.*

The code above illustrates how it is that we can add a button to our interface. We create a pointer of the class called button and this is used to access the button object which in this example is given the text "Button 1" and a parent class of "this" which is generally the main window.

There are also a number of slots that can be applied to a button such as *clicked()*, *pressed()*, *released()* or *toggled()* but in almost all cases we are only interested in the clicked function.

QLabel

The QLabel widget is very important to almost any GUI created in Qt, a QLabel can be used to display text which is usually critical in making it clear to a user can what part of an interface does, or to give an instruction. A QLabel can also be used to display images on the screen, although in our case we will be using the get_image function from the socket connection and taking another approach to displaying those images which we will cover soon. A QLabel can also change its text depending on some event having taken place on the interface, this is important as it may be a good way of giving a user feedback about what has occurred when an event has taken place. An example of how this may be important to our GUI is when the Z spatial coordinate of our spacecraft has changed, and we need to update that information on the screen.

QTextEdit

The QTextEdit class is almost self-explanatory as it is the widget, we need to use in our GUI to display the flight files that a user uploads to the interface. The text edit itself is quite straightforward and allows a user to type in text as you would in a normal text editor but is not much good without having code that will give it the functionality that we require.

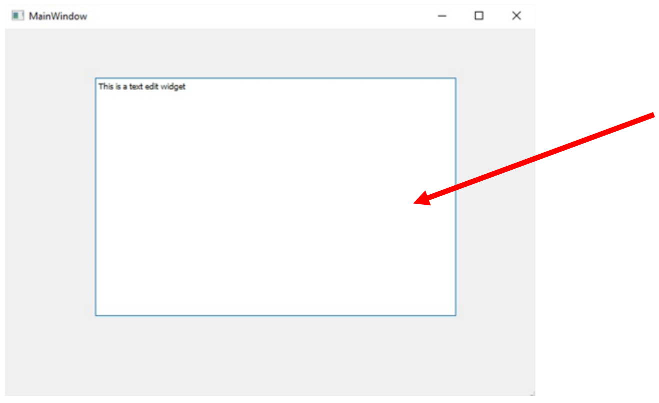


Figure 6 *– An example of a simple text edit widget placed into a simple GUI window in Qt without any functionality applied to it.*

QGroupBox

A QGroupBox is a widget that can be used for organising and structuring different groups of widgets in a logical fashion. Since it is the case that our GUI will consist of three different components using a group box would be a good way to separate our text editor from our model view and our spacecraft views on the GUI. It should be important to note that any widget that is placed within a group box then becomes the child of that widget. A group box also has a header label so that you can clearly identify what each group is for, an example of this can be seen in *figure 8*.

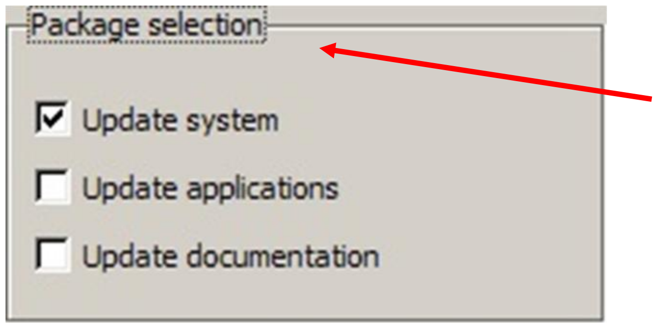


Figure 7 *– A QGroupBox widget with a title labelled "Package selection" which contains three tickbox widgets. A Groupbox can contain any number of widgets that can be grouped in a logical structure*

QGraphicsView/QGraphicsScene and drawing

For us to be able to create our model view of the spacecraft we need to be able to draw objects in 2D. Qt provides a graphics view framework from which we can work to generate our trajectory path. The QGraphicsView creates a widget which is like a canvas on which you can generate objects.

I think it is important to mention how the coordinate system works in Qt before moving forward. There are two main coordinate systems called the *logical* coordinate system and the

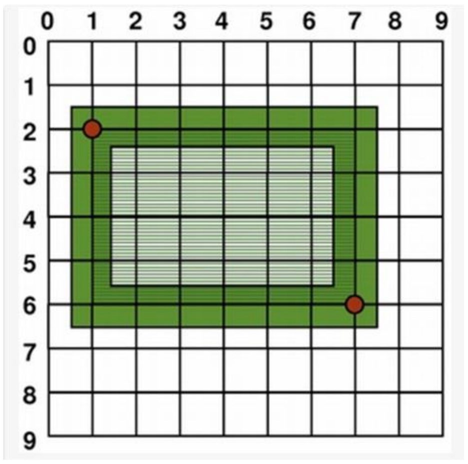


Figure 8 – *The diagram represents the coordinate system used in Qt where the origin begins at the top left. The X coordinate increases horizontally, and the Y coordinate increases vertically. Each coordinate position represents one pixel on the screen. This is the basic logical coordinate system used by Qt.* [8]

physical coordinate system, the logical coordinate system refers to the position of an item relative to the canvas itself and the physical coordinate system is the position of one item on the canvas compared to another. Qt uses cartesian coordinates with X and Y values that are measured in a unit of pixels. The origin, or (0,0) coordinate starts at the top left of the screen which is counterintuitive as we are used to working with cartesian coordinates from the bottom left position on an axis. In addition, there is no way to change the coordinate system so that is something that we must bear in mind.

The *QGraphicsView* is a widget from which you display all of your visual information, layered within the *QGraphicsView* is the *QGraphicsScene* which is used to store data items and a *QGraphicsItem* represents the data item itself, all of these three components represents the graphics view framework.

One of the things that I need to be able to do to model the trajectory path is draw lines and shapes on the screen, drawing functionality comes from the QPainter class. A subclass of the QPainter class is the QPen class which allows you to use a pen on screen to draw lines based on the coordinate points as demonstrated in ***figure 9***. The QBrush class allows you to fill in shaped objects with some colour and you can a class like QRect to draw shapes on screen.

We will play around with the different components from Qt mentioned in this chapter at the design stage when we start to build some prototypes that incorporate a lot of these different design features, as seen in chapter 5.

4.4 GUI requirements

Part of the process in understanding what the GUI needs to be is in understanding the requirements of the system, (see appendix for full list of the requirements). Most of the requirements of the system stem from discussion with Dr Martin about what it is that the GUI needs to be able to do. After some thought it became clear that the GUI would consist of three main components which are the following.

1) A *text editor* with the ability of a user to upload and modify a flight file.

2) A *model view* of the flightpath which would show the relative position of the spacecraft as it moved along its trajectory.

3) A *spacecraft view* which displays the rendered images observed from the spacecraft on its flightpath.

If we begin by discussing the text editor, Qt comes with a built-in widget class called a text edit that can be used to create our text editor. For the text editor to be of any use there also must be the ability of a user to select a file to be uploaded into the editor for manipulation if necessary. There are different ways of doing this such as having a toolbar with a file drop down option from which a user could select to upload a file. Or there could be an upload file button that uses the signal/slot mechanism so that when it is clicked a new window would open and a user could browse their file directory to find the right file. There are in addition some obvious pit falls at this juncture about making sure that a user only can upload the right kind of file into the GUI. If a user could upload a file into the editor that was not a flight file, then it could cause potential issues that need to be addressed. Furthermore, if a user uploads a flight file and they make some changes then a user should have the ability to save those changes made to that file as it’s not going to be much use if all of that data is lost as soon as the GUI is closed. Lastly a user should be able to create a flight from scratch within the editor and be able to save that file somewhere on their hard disk.

The model view is a more complex problem as we are trying to represent the position of a 3 dimensional space craft on a 2 dimensional plane. This is a problem that we would have looked to solve using OpenGL but as stated earlier was not really an option due to the time constraints. Dr Martin expressed his view of perhaps having a line that would represent the trajectory of the spacecraft from two of the three spatial dimensions and have the third spatial dimension be represented as a number in another place on the GUI. He also suggested that I use a symbol like an X to represent the final position of the spacecraft and some other symbol like a square or whatever makes most sense to represent the starting position of the sequence. To be able to create a plot like this requires being able to read the data from the flight file which gives the coordinate data of the spacecraft and being able to draw a line on the screen that shows this trajectory path.

The final component of the GUI is the spacecraft view which shows the images of the planetary surface from the camera's perspective. This requires the ability of the GUI to be able to connect to PANGU through the socket connection so that the images can be sent to the GUI. There is a function in the pan_protocol.cpp library called *get_image()* which must be used to access and display the images. A user should be able to save an image if they want to, this could be done by clicking on a save image button when a rendered surface is on screen. In addition, it is stated in the user manual for PANGU that the client software must be running in server mode for this to be possible. Setting up PANGU in server mode is relatively straight forward as there is a configuration file that comes with the software where you can simply edit it to set serve mode to true.

Other fundamental aspects of the requirements include the ability of a user to be able to run a simulation itself. So far, we have only described the three components of the interface, but it is not much good if we cannot begin to a run the simulation itself. This means that we also need a mechanism such as a button to begin the process of running through the different flightpath positions. Other safety features should also be put in place such as having a warning dialog that appears when a user brashly tries to close the application without having saved their work first, there are many things to consider that will become more clear through the development process. Therefore, the project shall try to best comply with using an iterative process where the requirements and design shall be revisited through each stage of development.

5. Design

This chapter will cover the design considerations of our system and look at the development of a couple of prototype projects that are going to be produced before working on the final model. It is important to be able to understand how certain functionality can be used in Qt before moving forward to creating our GUI.

5.1 GUI design concepts and principles

When creating a GUI for a desktop application it is important to consider some of the written literature in the field and some of the key design principles in place that are important in creating a good GUI.

The book "*The essential guide to the user interface*" [6] gives a list of general principles that should be followed during the design process. and it is with these principles in mind that I have tried to create an interface that additionally meets all the functional requirements of the design.

- **Aesthetically pleasing**
  has visual appeal with clear groupings and alignments
- **Clarity**
  visually, conceptually, and linguistically clear
- **Compatibility**
  compatible with, the user, the task, the product
- **Comprehensibility**
  User should know what, when, where, why and how to do it
- **Configurability**
  Easy personalisation and configuration of settings
- **Consistency**
  System should look, act and operate similarly throughout
- **Control**
  The user must control the interaction
- **Directness**
  Provide direct ways to accomplish tasks

- **Efficiency**

  Minimise hand and eye movements
- **Familiarity**

  Employ familiar concepts and use familiar language
- **Flexibility**

  A system must be sensitive to the different needs of its users.
- **Forgiveness**

  Tolerate and forgive common human errors
- **Predictability**

  User should be able to anticipate the natural progression of each task
- **Recovery**

  System should permit actions to be reversed, ensure users do not lose work
- **Responsiveness**

  System responds to user requests in a visual, textual, or auditory manner
- **Simplicity**

  Provide as simple an interface as is possible

Not all of these design principles are necessarily relevant to the flightpath planner tool, nor may all of these principles be adhered to in the time constraint given to the project but perhaps in later iterations of the GUI, more of these principles can be taken into account. In my view it is important to make sure that the GUI is as simple as it needs to be and no less so than is necessary to meet its functional requirements. It should be efficient such that operations can be completed in a clear and simple fashion, users should know exactly what the tool is for, for what purpose everything on the interface does and how they can use it. The GUI does not necessarily need to be aesthetically pleasing, to me and indeed, Don Norman who is one of leading thinkers in user interface design, does not agree that the design needs to be aesthetically pleasing as long as the functional operations are clear.

5.2 GUI early design sketch

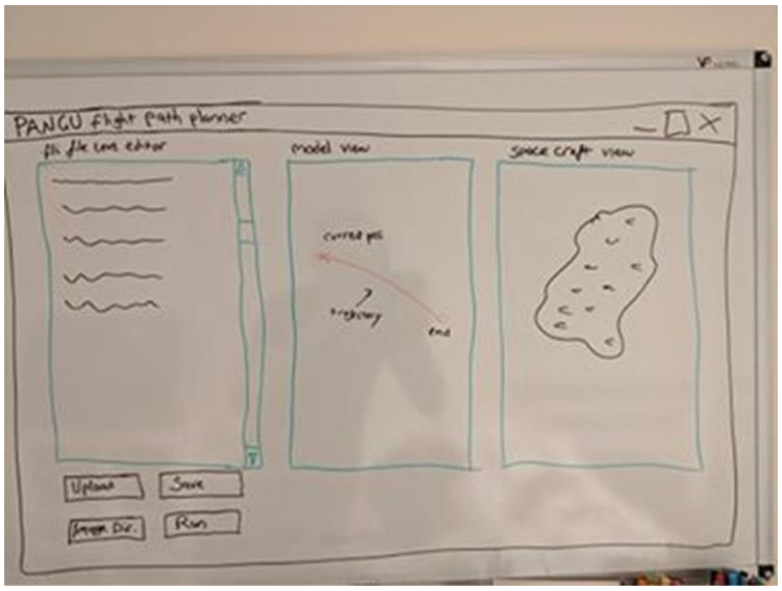


Figure 9 *– An early whiteboard sketch of the main GUI interface. The design includes the flight file editor, the model view illustrating the trajectory path of the spacecraft and the spacecraft view.*

Figure 10 as seen above is an early sketch and the first real visualisation of how the final GUI might look. The design is based on simplicity and functionality and is divided into three clear areas. On the left there is the flight file editor which is used to display the contents of a flight file, below the editor window are the buttons, *upload*, *save*, *image dir* and *run.* The upload button would be used to select a flight file to load into the editor, the save button would be used to save any changes made to the flight file, the run button would run the simulation and the image dir button is where a user would select a location to save the images generated in the spacecraft view.

5.3 Prototype development

One of the first prototype developed was about all about understanding the drawing functionality of the Qt application framework which is a requirement for the model view of the GUI. As mentioned in section 4.3 I signed up for three Udemy courses on how to use Qt and one of them "*Robust Qt & C++ GUI Programming a 2D Graphics application*" focuses on how to draw shapes using a set of parametric equations and the built in classes in Qt. The final GUI created in this tutorial is seen in figure 11

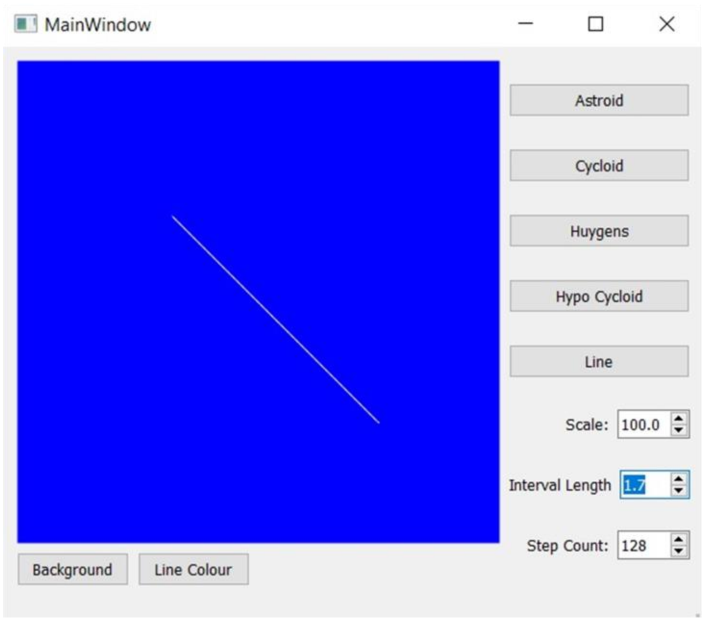
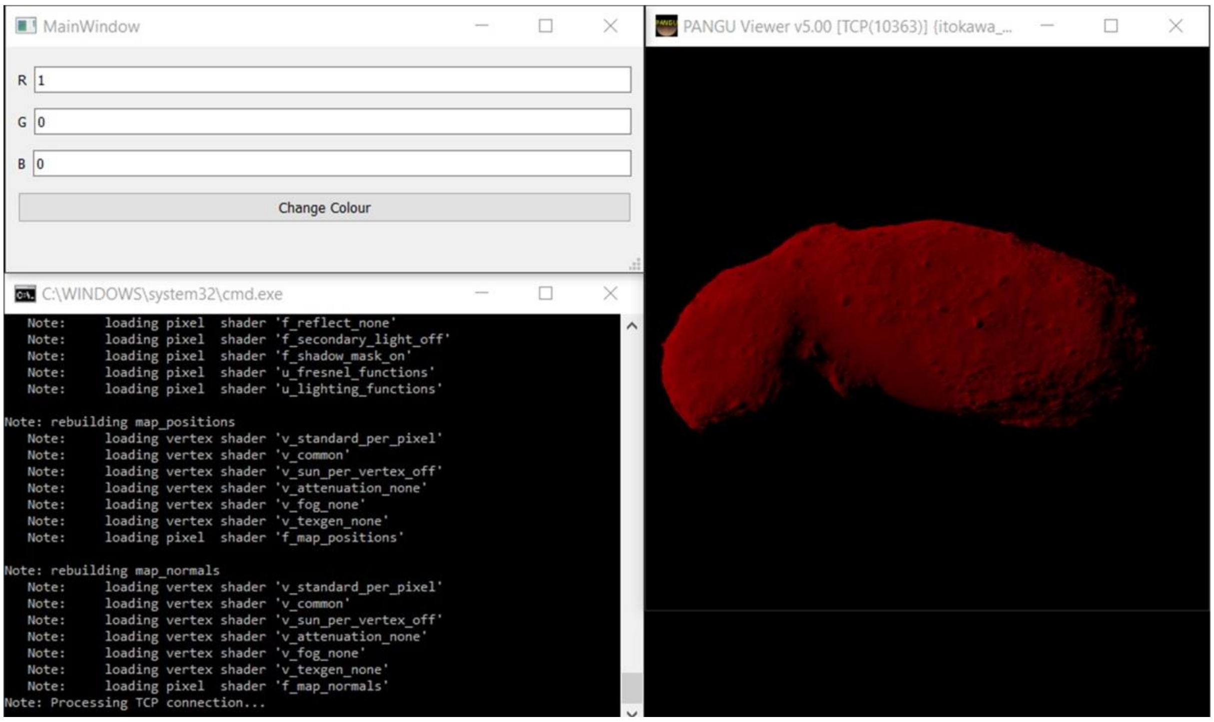


Figure 10 *– This GUI prototype project was a good learning experience in getting to grips with the drawing functionality in Qt. in this example a user can click buttons to generate different shapes, change the background and line colour or change the size of the shape itself.*

Developing this prototype was a good learning experience as it got me to grips with building my first GUI in Qt and taught me how I could use the different paint classes to generate shapes and objects on screen. During this project I also learned to apply functionality to the different buttons using the signals and slot mechanism. Each shape button on the screen is associated with a function and within that function is an equation used to describe the behaviour of the shape. The scroll boxes below can be used to scale the object and change the resolution of the object via the numbers of step calculations made.

A second prototype that I developed involved learning how to set up the socket connection between the PANGU server and a client GUI application. The socket connection is integral to interacting with PANGU at all, in addition to displaying images on screen. So, understanding how to set it up and use some of the built-in functions would be a big step towards beginning the final model. The GUI that I built can be seen in *figure 11*, it is a very simple interface with three line-edits where a user can change the value of the RGB colour values of the model and a button used to put the inputs into action. As is necessary PANGU is running the Itokawa asteroid model in server mode in the background. The function used is called *pan_protocol_set_sun_colour(this->getSock(), r, g, b);* which takes in the socket object and three floats which a user specifies on the interface to set the colour values of the model.


MainWindow
R 1
G 0
B 0
Change Colour
C:\WINDOWS\system32\cmd.exe
PANGU Viewer v5.00 [TCP(10363)] (itokawa_...

```
   Note:     loading pixel  shader 'f_reflect_none'
   Note:     loading pixel  shader 'f_secondary_light_off'
   Note:     loading pixel  shader 'f_shadow_mask_on'
   Note:     loading pixel  shader 'u_fresnel_functions'
   Note:     loading pixel  shader 'u_lighting_functions'

Note: rebuilding map_positions
   Note:     loading vertex shader 'v_standard_per_pixel'
   Note:     loading vertex shader 'v_common'
   Note:     loading vertex shader 'v_sun_per_vertex_off'
   Note:     loading vertex shader 'v_attenuation_none'
   Note:     loading vertex shader 'v_fog_none'
   Note:     loading vertex shader 'v_texgen_none'
   Note:     loading pixel  shader 'f_map_positions'

Note: rebuilding map_normals
   Note:     loading vertex shader 'v_standard_per_pixel'
   Note:     loading vertex shader 'v_common'
   Note:     loading vertex shader 'v_sun_per_vertex_off'
   Note:     loading vertex shader 'v_attenuation_none'
   Note:     loading vertex shader 'v_fog_none'
   Note:     loading vertex shader 'v_texgen_none'
   Note:     loading pixel  shader 'f_map_normals'
Note: Processing TCP connection...
```

Figure 11 *– On the top left of the image is the GUI window where a user can specify the input colour values of the model, to run the function the user presses the change colour button. On the right is PANGU running in server mode displaying the Itokawa asteroid model with the colour changed to red as specified by a user of the GUI. On the bottom left is the command prompt that shows the model being built and that the TCP connection is in place.*

6. Implementation and testing

This chapter will give an overview of the development of the final GUI and will aim to demonstrate that the requirements and use cases of the project have been satisfied. In addition, I will refer to some of the bugs that appeared in the code and the approach taken to deal with them.

6.1 The flight file editor component

As has been mentioned, the interface is to consist of three main components, the first is the text editor which I have decided should be the left-most component on the interface because it is what the user should first interact with. The purpose of this component of the GUI is to be able to upload and potentially edit a flight file, a user should also be able to save the updated file or save a copy with a new filename for use elsewhere. The GUI should also show which file has been selected and uploaded into the editor by the user.

The generally accepted method of best practice for testing an application is to apply test-driven development wherein software tests are prewritten before, or as the application is being developed. My approach has really been to deal with bugs as and when they have appeared in the code but I do have a set of use cases that lists the expected result from a series of actions that a user would want to accomplish. The use cases shall be used as test cases to make sure that the application achieves what it is intended to do.

6.1.1 The text editor algorithm for file upload

1) User clicks the upload file button

2) slot is activated, open a Dialog window for file selection (only flight files visible)

3) Call uploadfile function

4) Check if the filename is empty, if empty exit function

5) Check if the file is open, if not, exit function

6) If file not empty store file name in variable

7) Pass the file name into a QFile object

8) Check if the file is open, if not exit function

9) Create QTextStream to stream data from text file

10) Store each line of text in a QString

11) Filter out the file name from the full path and display it in the label below the editor

12) Clear the text edit to make sure there is no data already in it.

13) Display the contents of the text file in the editor.

14) Change status bar to "loaded flight file"

In the middle of this process, a filter is applied where the word "start" appears and the coordinate data is stored in a vector of vectors where each smaller vector stores one line of flight data. These numbers are then used to generate the model view in the next section but do not necessarily constitute part of the function of uploading a file.

6.1.2 Use case of uploading a file

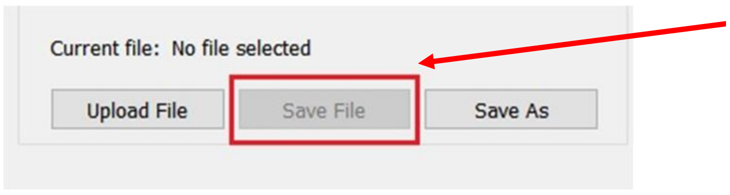
When a user clicks on the upload file button a new dialog window opens where a user is only allowed to select .fli files to upload into the editor. This is a preventative security method so that a user cannot upload any illegal file types that will cause the application to crash. Once the file is uploaded the file name appears in the text label and the data from the file appears in the editor. This is the expected outcome from these set of actions.

One of the bugs that I had to deal with was in the case that a user would upload a file into the GUI and the model view would be generated, but if they then tried to upload another flight file into the editor the data would not load correctly. This was because the vector storing the data from the file had not previously been cleared, the current file information did not update either. This problem could be solved by emptying the vector and resetting the current file information.

Other problems can arise however, when a user tries to complete operations for which the GUI was not intended. In the event that a user uploads a flight file that has bogus data that cannot be read, or read correctly by the application, the file will still upload into the editor but will fail to run as there is no valid data in the flight file. For a user to overcome this issue they will need to restart the application and upload a file which has good data in it.

6.1.3 Use cases of saving a file and saving as.

When the GUI application is first launched the '*Save*' button should not be accessible or functionable to a user as there is no data to save. Luckily, Qt has an option where you can disable a widget upon the launch of an application which is what I have done for the save button on the GUI. In the case that a user wants to write a flight file from the beginning, they can start by writing in the editor, but they must click on '*Save as'* first before the save button becomes actionable. This ensures that a file is in operation in the editor before it can be saved. Another security feature of the application is that when a user tries to save a file, they can only save the file as a .fli file rather some other type that is of no use to the GUI.



The save file button is greyed out upon initial launch of the GUI

Figure 12 *– Greyed out save button*

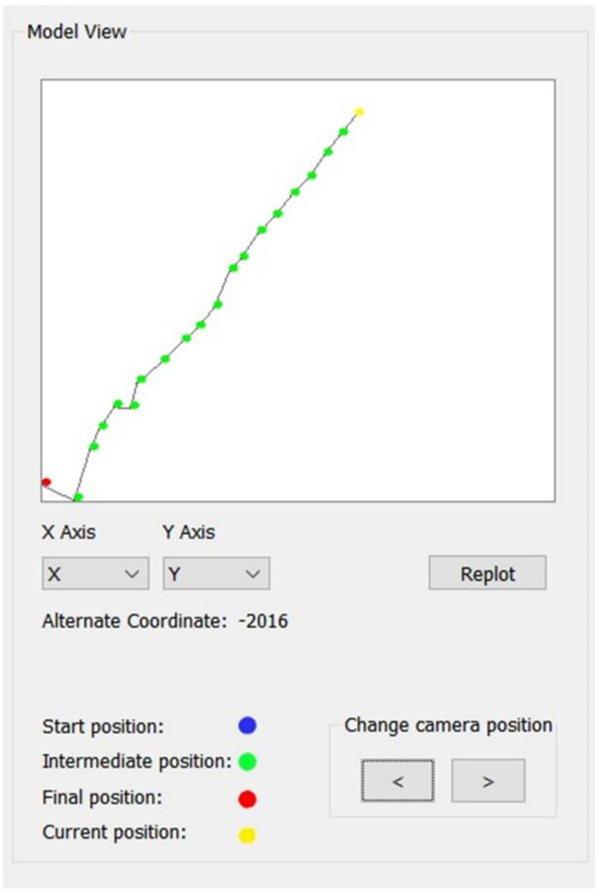
6.2 The model view component

The model view component of the GUI is probably the most complex part of the whole project and I think it is reasonably safe to say that this element is the one that could be most developed in future iterations.

The idea behind the model view is to extract the coordinate data taken from the flight file and express this information as a plot. Initially I was going to just run the simulation with a button where the image would be rendered and left for a 3 second period, but it made more sense for a user to be able to scroll through each coordinate position because of the save image functionality that we also want to incorporate. This meant that the '*Run*' button was removed and replaced with two scroll buttons so a user can move backwards and forwards through the simulation at their own leisure.

Another significant change that was made relatively late on in the project was the ability of the user to select which two of the three dimensions (*x*,*y*,*z*) they would like to plot on the screen. The reason for this is that sometimes one of the three dimensions may not change position at all through the whole simulation making the model look flat, a user may also just want to see a plot of two different coordinates. With this new and improved addition to the GUI a user can specify which of the coordinates they want to map and on which axes.

One of the suggestions from Dr Iain Martin was to introduce a key to the interface which makes it clear to a user which coordinates represent what position in the spacecraft's trajectory. In the beginning I was only going to use a cross to represent the final position and a circle to represent the starting position but given what is possible in Qt (as far as my understanding goes) representing the coordinates as an ellipse and using colours is just as effective and gives just as clear of a result.



Alternate Coordinate: -2016

Start position: ●
Intermediate position: ●
Final position: ●
Current position: ●

Figure 13 *– A display of the model view with a plot of some coordinate data taken from a flight file. The coordinates represent each of positions of the spacecraft on its trajectory as it generates images of the planetary surfaces. The red point indicates the final position of the spacecraft, the green intermediate points, and the yellow coordinate the current position.*

The alternate coordinate value corresponds to the X and Y value of the current position (yellow circle)

6.2.1 Use case of generating a plot from a flight file

Once a user has uploaded a flight file, the plot should immediately be generated in the model view, this happens as stated in section 6.1.1 where coordinates are filtered out and stored in a vector. A function called *drawtrajectory()* is called from the *uploadfile()* function which immediately adds a 10% margin to the drawing canvas whose scale depends upon the maximum values of the coordinates in the flight file, the coordinates are then drawn on the canvas and the model view is generated. The margin is necessary as otherwise coordinates are mapped right on the boundary points of the canvas and can be very difficult to see.

I have tested several different flight files, with little amounts of data, lots of data, negative numbers, small and large numbers. The alternate coordinate number is also displayed correctly on the interface in each case. In theory the plots are created in the correct fashion but could be improved by working in 3 dimensions.

6.2.2 Use case of changing the cameras position

With the assumption that a user has uploaded a flight file and the model view has been plotted, the current position of the model should align with the start position on the plot. When a user presses the right arrow key to change camera position forward, the current position of the model should change to the second coordinate and the view from the spacecraft should also change in alignment with the change in the model position. In addition, the alternate coordinate should display the coordinate that a user has chosen not to plot on screen. For example, if a user is plotting the X and Y coordinates, then the Z coordinate should appear on the GUI as a number. The user should be able to continue pressing the right camera button until they have reached the end of the simulation, where at that point nothing more should happen. The left camera button should give the same functionality except to move backwards through the simulation until they reach the starting position from the flight file.

In the event that a flight file has not been uploaded into the editor and a user presses either of the camera arrow keys, the status bar should change and display the message “*File not uploaded, cannot change camera position*” to notify the user that they have not yet uploaded a flight file to run the simulation. The application passes these tests with good results.

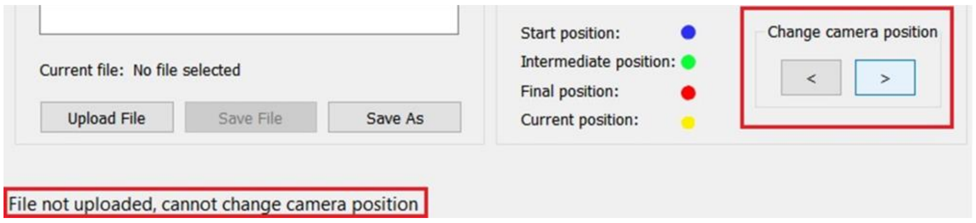


Figure 14 *– Demonstrates the change in the status bar when a user tries to change camera position without having uploaded a flight file into the editor*

6.2.3 Use case of changing plot coordinates

By default, the X coordinates are plotted along the X axis and the Y coordinates are plotted along the Y axis. A user is free to change which of the X, Y or Z coordinates they want to plot on what axis which they can do by clicking on the dropdown and selecting which coordinate they want to plot.

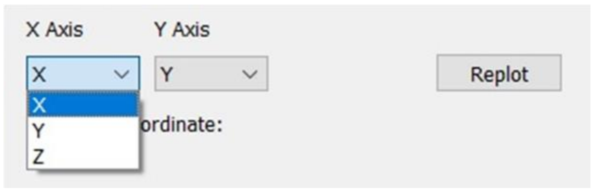


Figure 15 *– A display of the drop-down menu where a user can select which coordinates, they want to plot on which axes. The user must press the replot button for changes to take effect.*

The model view of the spacecraft trajectory should only be rendered once the replot button has been pressed. Once the replot button has been pressed the alternate coordinate should and does correctly display its value and the new plot is correctly shown on screen on the correct axes.

In the event that the replot button is pressed without there having been any flight data loaded into the GUI then then the status bar comes into action again displaying a message that no plot can be generated because no flight file has been uploaded. This gives a user useful feedback that their request cannot be completed.

6.3 The spacecraft view

On the surface the spacecraft view component of the GUI is the most straightforward, it consists of a QGraphicsView widget that displays the image sent back from the PANGU server and a save button. But it is also the only component of the GUI that is completely reliant upon the socket connection. Without which we would not be able to send or receive any information from PANGU.

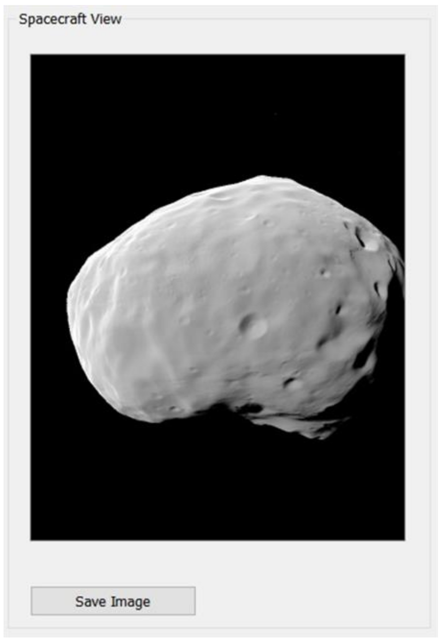
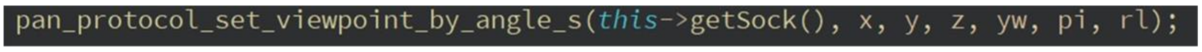


Figure 16 *– The spacecraft view component of the GUI with a generated image of the Phobos model. It consists of one simple button which is used to save the current image displayed on the screen.*

The image could not be incorporated into the GUI though without having a socket connection setup to send and receive messages to the PANGU server.

The socket operation is used to send and receive messages between PANGU and the GUI, wherein, each of these messages contains an ID that indicates which action is to take place. The ID's themselves are in the *pan_protocol_lib.h* header file and have a number associated with the given operation. In addition, there are several **_TX* (transmission) and **_RX* (receiver) helper functions that are responsible for building messages to be written to the socket. The transmission functions define a buffer which is a temporary memory storage space of a given size. An example of one of the functions used is seen in ***figure 17*** which is used to pass our coordinate data to the PANGU server.



```
pan_protocol_set_viewpoint_by_angle_s(this->getSock(), x, y, z, yw, pi, rl);
```

Figure 17 *– One of the transmission functions used to send the spacecraft coordinate data to the PANGU server*

This function begins by allocating the size of the buffer which holds the message ID (a long of 4 bytes) and then passes each of the six coordinate parameters which are floats (4 bytes). The message ID itself in this case is *MSG_SET_VIEWPORT_BY_DEGREES_S = 256* and is put onto the buffer via the helper function *pan_socket_poke_ulong()* in the *pan_protocol_lib.cpp* file. The six coordinate parameters are also added to the memory buffer via the function *pan_socket_poke_float().* Lastly the buffer is written to the socket by calling the function *pan_socket_write().*

Message ID	**x**	**y**	**z**	**yw**	**pi**	**rl**
4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes

***Table 1** – the message buffer data passed from the function pan_protocol_set_viewport_by_angle_s()*

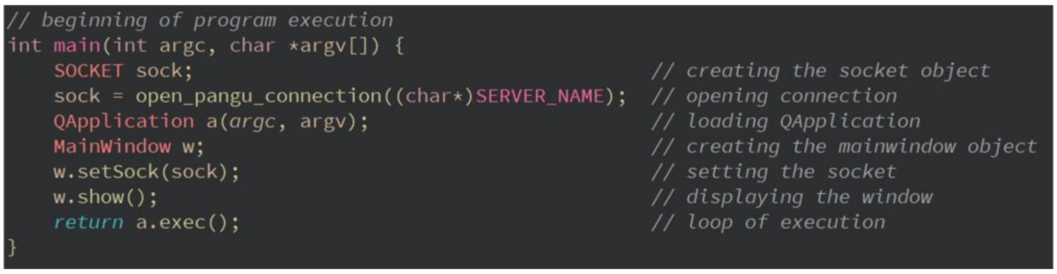
6.3.1 Use case of saving an image

To save an image a user must have already uploaded a flight file, this should automatically generate the model view in addition to immediately showing the first position of the spacecraft camera view. All a user needs to do is click the save button and the image is automatically saved to the homepath directory on the user's computer. The status bar will also display a message for 3 seconds, letting a user know that the image has been saved. The image is automatically saved as type *PNG*, unfortunately there is not yet the functionality in place for a user to choose what file type they want to save the image as, or of the name of the file.

In the case that a user clicks the save image button before the status bar will inform the user that they cannot save any images until they have uploaded a flight file.

6.4 Code structure

The design and structure of the code follows the Qt design principles with a *MainWindow* parent class which any other *QWidget* classes use as its parent. An object oriented approach was taken to using the socket connection, the socket was added as a private member and can be accessed or set by *getSock()* and *setSock()* functions respectively. The flow of the program is that the socket connection is opened using the *open_pangu_connection()* function which returns the socket. The *QApplication* is then created as is the *MainWindow* and the socket is added to the *MainWindow* object using the *setSock()* method. The *MainWindow* objects now have access to the socket and can use it as required to call various PANGU functions. Finally, the *show()* method of the *MainWindow* class is called to load the GUI.



```
// beginning of program execution
int main(int argc, char *argv[]) {
    SOCKET sock;                                        // creating the socket object
    sock = open_pangu_connection((char*)SERVER_NAME);   // opening connection
    QApplication a(argc, argv);                         // loading QApplication
    MainWindow w;                                       // creating the mainwindow object
    w.setSock(sock);                                    // setting the socket
    w.show();                                           // displaying the window
    return a.exec();                                    // loop of execution
}
```

Figure 18 *– The main function of the GUI from where operation of the program execution begins.*

7. Evaluation

PANGU is a software application that requires a license that is specific to a user's machine, this means that it is not software that is freely available to anyone online. Therefore, the scope of people who can test the new GUI application is limited. In addition to this, the country has been in a full lockdown situation since the end of March due to the COVID-19 pandemic meaning that the University and other public spaces have been closed. Because of these rare and complicated set of circumstances I have been limited to doing a user evaluation of the GUI with my partner who I share a household with.

The evaluation was conducted upon completion of the GUI and is therefore considered to be a final evaluation of the product. The participant was given an explanation of the purpose of the GUI, so they had some idea about what the tool is designed for. The beginning of the process involved letting the user have a look at the GUI and were then asked to complete a series of tasks that have been described by the use cases in the previous section.

Once the participant had completed the tasks, they were given a questionnaire to fill out so that they could give their thoughts about their experience of using the GUI.

The results of the questionnaire and the observations made by me during the process suggest that the user liked the overall design of the interface but found certain elements a bit confusing. The participant was able to upload a flight file successfully for example but did not understand what any of the coordinate data meant without an explanation, this of course is expected. An experienced user of PANGU would know what data existed in the flight file as it is probably something that they would have created themselves.

The most confusing part of the user's interaction was in understanding the model view and how to do a replot. They did not know which data aspect from the flight file represented what coordinate and had no understanding of what the alternate coordinate number represented on the screen.

The user was easily able to understand what the change camera operations would do as they clicked through the simulation and after a short while it became more clear to them what was happening in the model view component of the GUI.

The biggest problem identified in the process was that the user was not initially clear that an image had been saved when asked to complete this task. They did not immediately identify the change in the status bar that told the user the image had been saved and in what location and did not know whether the image had been saved or not.

The results suggest that there could perhaps be more clear feedback given to the user upon taking some action and a better explanation given as to how the model view works.

8. Description of functionality and Interfaces

Chapter 8 shall give an overview of the complete functionality of the user interface for the spacecraft flightpath planner GUI without going into the detail of applying tests as seen in chapter 6 of the report. The complete and final product, without any data being loaded into the editor is seen in ***figure 19*** below. A more detailed overview of each of the components can be seen in ***figures*** (20,21 and 22) along with an explanation of their functionality.

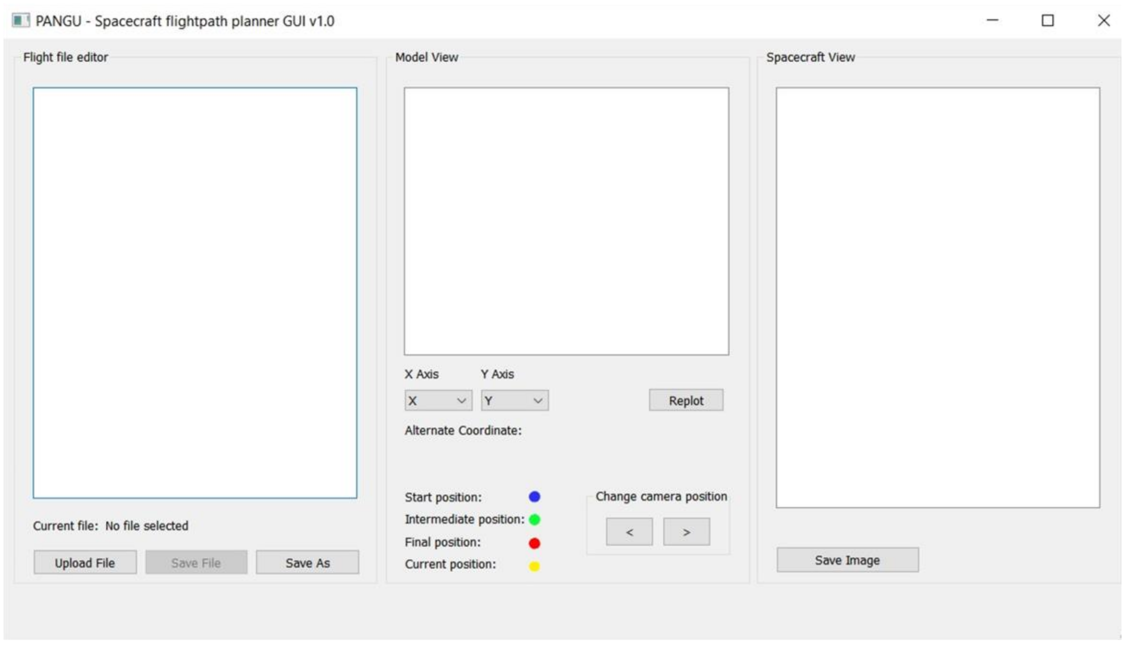


Figure 19 *– The final GUI for the spacecraft flightpath planner, consisting of the flight file editor, the model view, and the spacecraft view components.*

The GUI is given an appropriate title of "*PANGU – Spacecraft flightpath planner GUI v1.0*". Unfortunately, there is currently no application icon, but an appropriate addition would be to apply the PANGU logo.

It is designed in a logical fashion so that a user should move from the left side of the interface by first uploading the flight file, to adjusting the model view to suit their interests and lastly by viewing the rendered image and deciding whether they want to save that image or not, on the right.

8.1 Flight file editor interface

The flight file editor is the component where a user can upload a flight file into the interface, make any changes they want to the file and even create a new file which they can save on their hard disk. ***Figure 20*** below shows this part of the interface in some detail. There are three buttons on the interface that are used to upload a file, save a file and save as with clear labelling on each. As has been mentioned previously the save button has been greyed out or disabled until some file has been uploaded into the user interface. The last feature is the label which shows the name of the current file loaded into the editor, if no file is loaded then the label will state that no file has been selected.

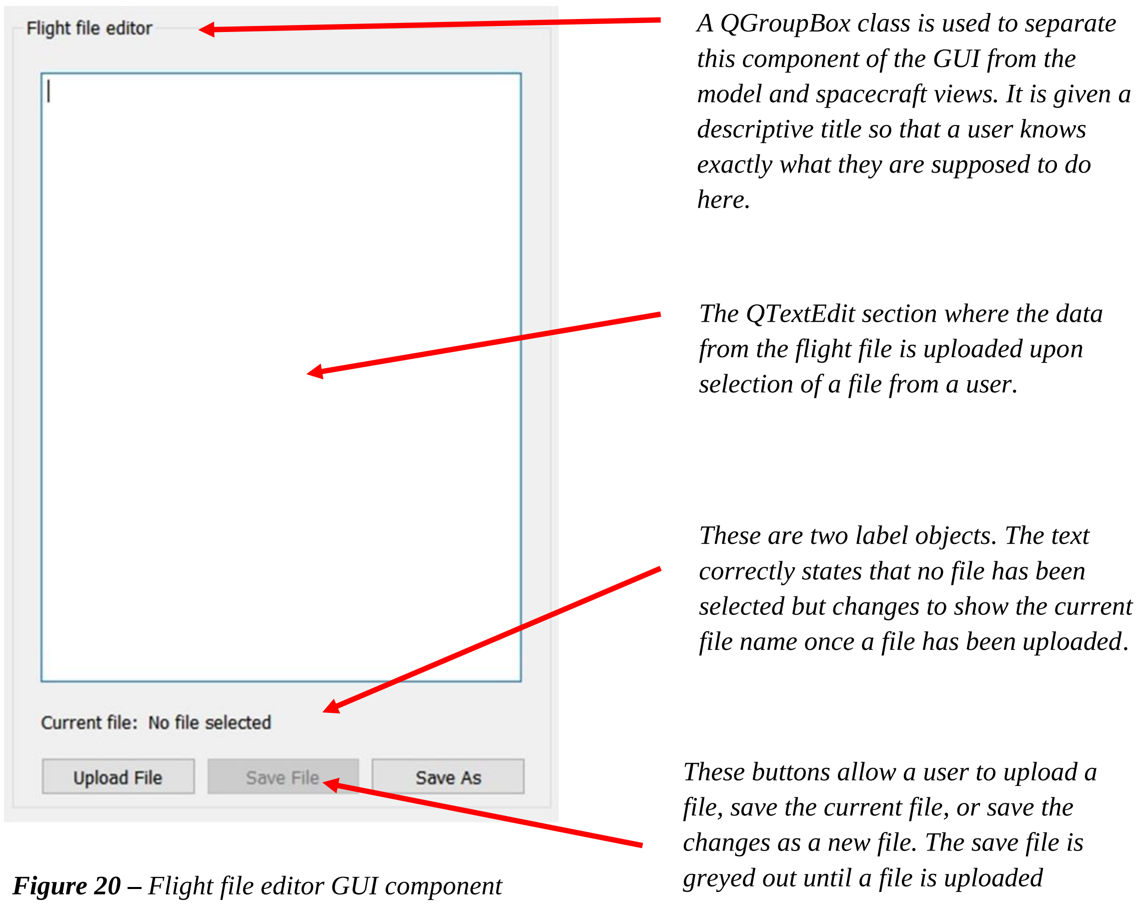
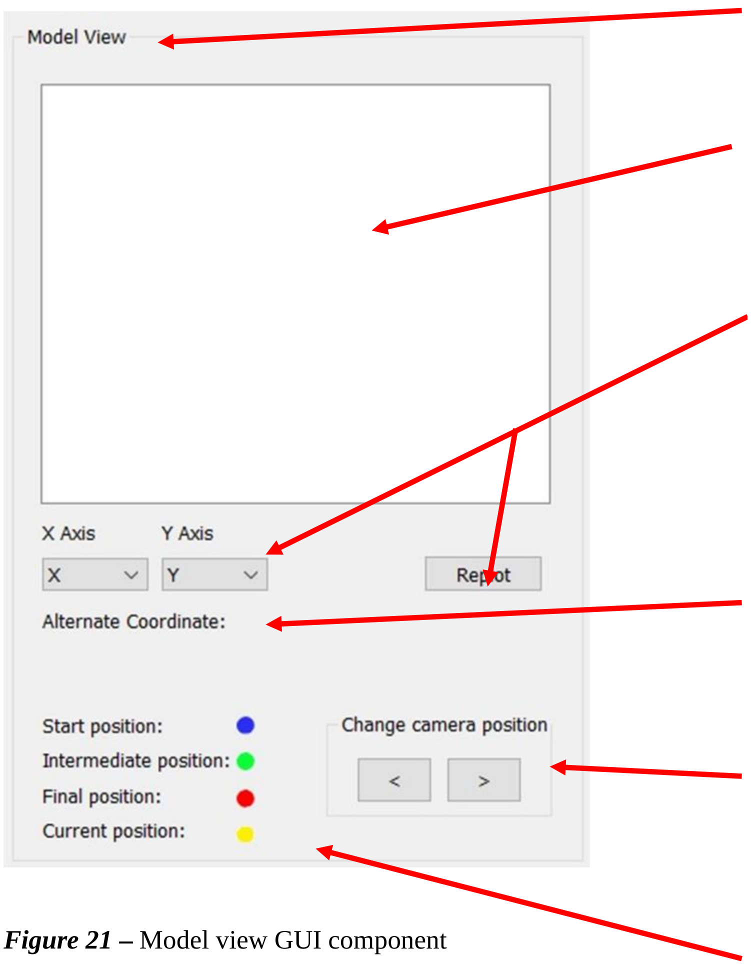

Flight file editor
Current file: No file selected
Upload File
Save File
Save As
A QGroupBox class is used to separate this component of the GUI from the model and spacecraft views. It is given a descriptive title so that a user knows exactly what they are supposed to do here.
The QTextEdit section where the data from the flight file is uploaded upon selection of a file from a user.
These are two label objects. The text correctly states that no file has been selected but changes to show the current file name once a file has been uploaded.
These buttons allow a user to upload a file, save the current file, or save the changes as a new file. The save file is greyed out until a file is uploaded

Figure 20 *– Flight file editor GUI component*

8.2 Model view interface

The model view interface is the part of the interface that shows the trajectory of the flightpath on its journey from beginning to end. The different coordinates are colour coded to show what part in the trajectory they represent. There is additional functionality in that a user can select which coordinates they want to plot on which axes, with the plot being re-rendered when the replot button is clicked. Since the plot can only be rendered in two dimensions, the third dimension is listed as the alternate coordinate and changes as and when the user changes camera position. The user can change camera position by pressing the arrow keys left or right to move through the simulation at their own speed.



A QGroupBox class for the model view

The Graphics view widget window which is where the plot is generated from the data uploaded from the flight file.

A user can choose which components they want to plot on either the x or y axis, as sometimes one of the dimensional positions of the spacecraft may not change leaving a flat plot.

The alternate coordinate label gives the value of the dimension not mapped onto the model view

The arrow keys allow a user to move from one frame to the next through the flight file, moving back and forward. The current position in the model also adjusts.

Figure 21 **–** Model view GUI component

This is a key that just makes it clear to a user what colours are used to represent different parts of the model trajectory.

8.3 Spacecraft view interface

The spacecraft view interface is the right most section of the GUI and it displays the rendered images that come from the PANGU server application, this means that for its functionality to be operable PANGU must be running in server mode in the background. Furthermore, for the spacecraft view to be of any use a flight file must be loaded into the editor for any request to be made to the server. The only other functionality of this component is the ability of a user to save an image they view on the interface which is executed by pressing the save button on the current image. Images are automatically saved in PNG format in the home directory.

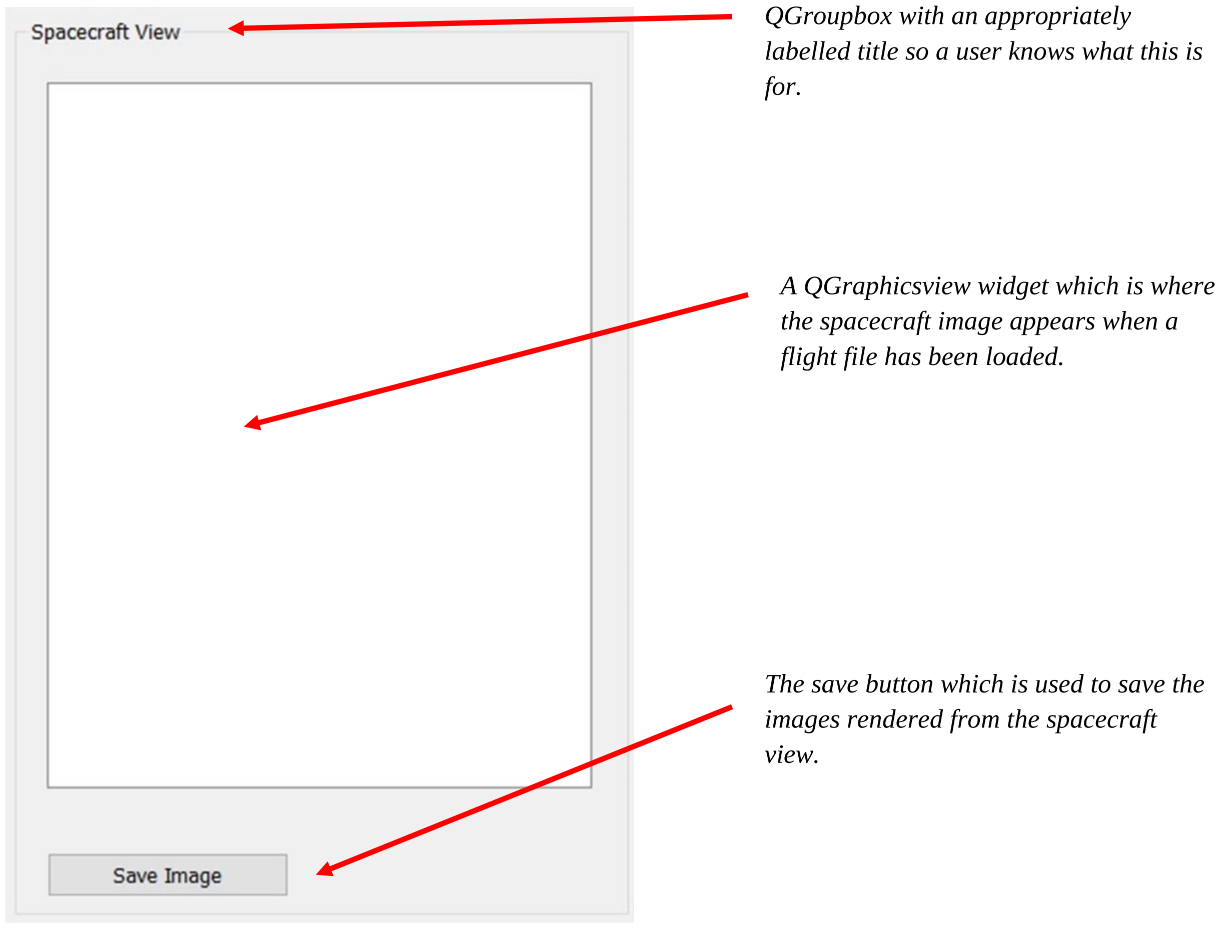

Spacecraft View
Save Image
QGroupbox with an appropriately labelled title so a user knows what this is for.
A QGraphicsview widget which is where the spacecraft image appears when a flight file has been loaded.
The save button which is used to save the images rendered from the spacecraft view.

Figure 22 **–** Spacecraft view GUI component

9. Appraisal

The project began under difficult circumstances and progress initially was quite slow, I think it probably took quite a long time to get a handle of the overall scope and what the end goal was. I found it difficult at first to visualise how the interface might look because I was not entirely sure what I was trying to create. For example, I did not know exactly what the model view was meant to be or what it represented, I had images in my head of a view of the planetary model at a distance and a curved trajectory line in 3D space which I had no idea how to create. I would say for sure that the process of understanding was made more difficult by the fact that the country has been in a lockdown situation for the last six months and I could only communicate to my supervisor via Microsoft teams meetings. I think in hindsight it would have been a good idea to ask more questions via email early in the project because written responses are always something you can go back to when you need information. Sometimes things are not always so easy to explain or understand via teams meeting and taking notes while trying to listen to someone talk is hard.

Although I found the Qt learning resources to be of varying quality, the documentation that comes with the framework and what is on their website is pretty good, but there does not seem to be much available in regard to textbooks which is why I made good use of the online courses to learn the platform. The Udemy courses could be a mixed bag, with course instructors not always fully explaining what and why they did what they did. With that said I’m still not sure what I would have done differently in learning how to use the application because there are not a large number of resources available, it took a while to find the right sources in learning what I needed to learn.

Overall, we managed to follow the schedule relatively closely. At first it was not clear whether there would be the time to learn and incorporate OpenGL and in hindsight it was the correct decision not to focus on it because we ended up not having the time. It would have been a nice addition to my skillset and would have no doubt improved the project overall but we would have ended up in more trouble had we decided to pursue using it to create the model view interface.

I think the overall design choice was good and the philosophy of keeping things clean and simple a good one. I am generally pleased with the overall layout of the interface and think it complies well with the principles listed in section 5.1. In terms of code organisation it might have been a better idea to make use of more classes for each of the components of the interface, this would have kept the internal organisation neater and easier to work with in future iterations. Otherwise I think I generally used the right approach, I would perhaps suggest that I might have been better trying to use some sort of plotting package in Qt for the model view and this could be a good intermediary step towards using OpenGL to create a 3D model view.

In regards to the wider context I would hope that the resulting application is of some use for those specialists who use PANGU to build simulations if they ever come across this tool. Perhaps it would even work as a basic framework towards building a more complex GUI that meets more complex requirements of the user. But since the application is so niche, I would not expect there to be huge commercial implications. I would just hope that it is a benefit to the Scientists who might use it and perhaps make their job slightly easier.

10. Summary and conclusions

The aim and objective of the project was to create a GUI tool in which a user of the PANGU software application could operate flight files. The current method of executing a flight file and generating the subsequent spacecraft images was done through the viewer tool which is not designed specifically to handle the flight file itself. A user would enter a command which the viewer tool would execute, displaying images on the screen and saving them to a folder very briefly before closing. The new spacecraft flightpath planner GUI is designed specifically to handle the operation of flight files so that a user has more control over the quality of the output.

The resulting product, the spacecraft flightpath planner GUI tool v1.0 is generally a success and I believe the result meets the basic specification. Users of PANGU can upload their flight files into an editor and make changes as they see fit. They can generate images from the spacecraft and view them in the GUI with the option of saving the image or not. Lastly a model view is also in place so that the user can see the flightpath trajectory in addition to all coordinate positions along the way. The GUI is simple and hopefully a bit improvement over the old process of having to use the viewer tool and hoping that the images meet the required resolution. user-friendly with a clear direction that the user should operate it from left to right.

11. Recommendations for future work

Future versions of the spacecraft flightpath planner GUI could probably include more text editing functionality, which you will find in most modern word processing packages such as Microsoft word. Features could include, the ability to change fonts, change the font size apply different colours to the text etc. Those changes might aid people with visual impairment issues who find it difficult to read small text. The GUI could also include a tool bar with clickable shortcuts for basic text editing functionalities such as cut, copy, and paste. It would also be useful to see the addition of flight file being highlighted in some colour to show which line in the text file corresponds to the model and spacecraft views.

The model view could be much more comprehensive with the addition of the OpenGL platform application to produce a 3D model of the spacecrafts trajectory. This would do away with the need to specify the alternate coordinate on the interface and the whole plotting of coordinates functionality. Additionally, it would also be a big improvement for the GUI to incorporate flight files that make use of the SPICE ethemeris system, or PANGU ephemerides files which specify the value that a property has at different times. Currently the GUI only works with basic flight files by filtering out the coordinate data that appears after the word “start”. For more advanced flight simulations, a user may require specifying the positions of solar system objects at specific times. With the added ability of a user to incorporate this data the GUI could accommodate more complex simulations.

In version 1 of the GUI images are saved to the home directory by default and the user is not given a choice of the file name that they want to specify for each image. An improvement would be to allow a user to specify both where they want to save their files and what the file name should be. Furthermore, the addition of the ability of a user to choose the file type that they want to save the image would be useful.

It may also be useful to include a help guide as part of a toolbar dropdown that would include more detailed instructions about how a user may complete operations if they were to get stuck with something. Finally, I might suggest that the incorporation of the GUI to display different languages could be a useful feature for users of PANGU who work in different countries and whose first language may not be English. Perhaps the addition of a feature like this would open the product to a wider market in terms of the commercial aspirations of the company.

12. References

[1] M. I. M. Dr SM Parkes, “Lunar surface Simulation - Opening the road to the Moon,” *Exploration and Utilisation of the Moon. Proceedings of the Fourth International Conference on Exploration and Utilisation of the Moon,* pp. 221-224, 2000.

[2] NASA, “NASA jet propulsion laboratory,” NASA, 1982. [Online]. Available: https://naif.jpl.nasa.gov/naif/spiceconcept.html. [Accessed 21 September 2020].

[3] I. M. M. D. D. M. S.M. Parkes, “Planet Surface Simulation with PANGU,” in *Eighth International Conference on Space Operations,* Montreal, 2004.

[4] M. Torrence, “NASA - The Mars Orbiter Laser Altimeter,” NASA, 19 1 2007. [Online]. Available: https://attic.gsfc.nasa.gov/mola/. [Accessed 2020 09 21].

[5] S. P. M. D. Iain Martin, “Modeling cratered surfaces with real and synthetic terrain for testing planetary landers,” *IEEE Transactions on Aerospace and Electronic Systems,* vol. 50, no. 4, pp. 2916-2928, 2014.

[6] W. O. Galitz, “The essential guide to user interface design second edition,” New York, Wiley Computer publishing, 2007, pp. 42-47.

[7] M. Dunstan, “Youtube,” Space technology centre, University of Dundee, 16 January 2017. [Online]. Available: https://www.youtube.com/watch?v=zJYlGI1n-qg&ab_channel=MartinDunstan. [Accessed 21/09/2020 September 2020].

[8] Qt. company, “Qt Coordinate systems,” Qt, 2016. [Online]. Available: https://doc.qt.io/archives/qt-4.8/coordsys.html. [Accessed 21 September 2020].