

Contents

1. Introduction	2
2. Getting Started.....	2
Requirements.....	2
Setting up the Project	2
Controller bindings.....	2
3. Folder Structure	3
4. RocketPy: Generating Simulation Data.....	3
5. Major Components	6
Getting RocketPy Data into Unity	6
Trajectory Visualisation/Scatterplot	7
Trajectory Animation	8
Trajectory Filtering.....	9
Bar Chart	9
Linked Highlighting.....	10
6. How-Tos	11
How To: Create a Trajectory Visualisation Object	11
How To: Setup VRTK Camera and Input Buttons	12
7. Known Limitations	12
8. Roadmap	13
9. Team	13

Rocket Trajectory Visualisation Tool

1. Introduction

The Rocket Trajectory Visualisation Tool is a Unity project to visualise rocket simulation data. It applies the model of the [Ladders of Abstraction](#) to visualise trajectories of rocket launches whilst introducing interactivity in moving between different levels of data abstraction. Along with the trajectory visualisation is the visualisation of the abstracted variables(s) as a 2D/3D bar chart. Linked highlighting between the two visualisations then allows researchers to observe correlations between input values and the impacts that changing them have on flight trajectories.

The project has been developed based on the rocket simulation data generated from the open-source project RocketPy. RocketPy simulates realistic trajectories of launches based on input environmental, launch and rocket variables. The RocketPy source code has then been refactored to output the results of the simulations into CSV files which are then fed into Unity and visualised using IATK (Immersive Analytics Toolkit) and the 3D Interactive Bar Chart package.

2. Getting Started


Requirements

- Project pulled from GitHub
- Unity 2020.3.32f
- OpenXR compatible headset, including Oculus Quest, Oculus Quest 2, Microsoft HoloLens, and Valve Index

Setting up the Project

1. Connect and setup your VR headset according to manufacturer's instructions.
2. Open the project folder and navigate to **Swordfish_IT_Project -> Assets -> Swordfish -> Main.unity**
3. Play the scene. The project should automatically detect and run the project on the VR headset.

Controller bindings

- Trigger button = selection
 - To show display for data point in trajectory visualisation
 - To show display for bar in bar chart
 - UI components
 - Grabbing the bar chart with 
- Primary button (A/X button) = select a data point to select its trajectory as the trajectory to play the animation over
- Left thumbstick = movement
- Right thumbstick up (hold) = activate pointer to interact with UI components

Note: These bindings can be changed by following the How-To guide in section 6.

3. Folder Structure

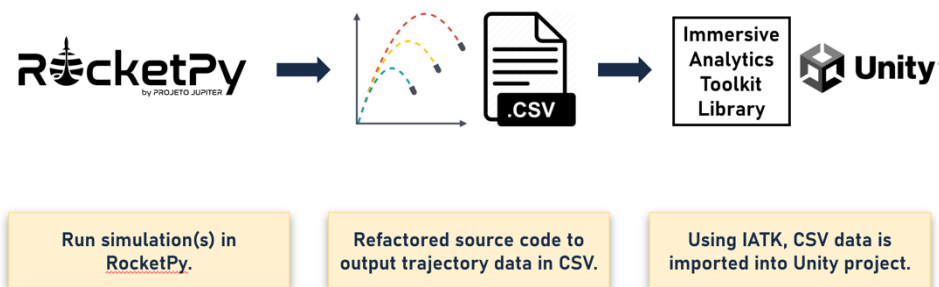
All the scripts created by the development team are located within the *Assets/Swordfish* folder, with any prefabs made by the team located in *Assets/Swordfish/Resources*. There are three primary packages that have been integrated into this project. All three of these can be accessed via the main *Assets* folder:

- IATK (Immersive Analytics Toolkit) - *Assets/IATK*
- 3D Interactive Bar Chart – *Assets/VC*
- VRTK (Virtual Reality Toolkit) - *Assets.VRTK.Tilia.Package.Importer*

All simulation data needs to be stored in the *Assets/Resources* folder. Refer to [section 4.5](#) for further details.

4. RocketPy: Generating Simulation Data

This section will outline RocketPy, the methods used for generating the rocket simulation data and importing the data into the Unity project. This is the pipeline used for generating simulation data and importing it into our Unity project:



Useful Links

- [RocketPy GitHub](#)
- [RocketPy v0.11.0 \(July 2022\)](#)
- [RocketPy Documentation](#)
- [RocketPy Technical Paper](#)

1. RocketPy Overview

Rocket simulation data is generated using open-source software RocketPy. Written in Python it allows you alter environment, launch and rocket variables and execute trajectory simulations. It is recommended to read through the official RocketPy documentation and attempt the 'Running Your First Simulation' section to gain an understanding of the main system classes (Solid Motor, Rocket, Environment, Flight).

2. Setting up the Development Environment

A customised rocket simulation file has been used for running rocket simulations: **ColabRocketPy.py**

It can be found within the project folder: **Swordfish_IT_Project -> Assets -> Resources**

The ColabRocketPy.py has been confirmed to function correctly with RocketPy v0.11.0 (See link above). It has not been tested with later versions.

Note: It is recommended to store the RocketPy files and ColabRocketPy.py on a Google Drive and run the simulation through Google Collab. This will avoid any errors with dependencies. The simulations were run locally on a computer until a bug was introduced in a RocketPy dependency (netCDF4), causing it to crash when accessing the weather data via API. Using Google Collab will avoid this error.

3. Running a Simulation

Simulations are executed through the ColabRocketPy.py file. Various simulation settings can be changed.

Number of Simulations & Altering Rocket Variables

The current setup of ColabRocketPy loops each *SolidMotor.thrustSource* within the *motors* array and runs *iterationsPerMotor* number of simulations.

For each simulation for a given rocket motor, the *Rocket.mass* is altered. The mass which each rocket motor begins at is set with the *startMass* variable and the step taken each loop for a motor is set with the *step* variable.

Therefore the total number of simulations conducted is $iterationsPerMotor * motors.length$

Setting the time steps (i.e. the number of data points)

The time step (in seconds) between each data point of a simulation can be set with the function parameter *timeStep* in the function *Flight.exportData()*. It is currently set to 0.6 seconds which will generate roughly 300-500 data points for a single simulation. It is recommended this value either be kept as is, or increased further to lower the number of data points generated. When visualising the data points in Unity it is best to keep the number of individual objects to a minimum to conserve performance.

Changing Flight Location & Rocket Parachutes

Two parachutes are currently included with the rocket created in the ColabRocketPy.py file, however they are not required to simulate a launch. If the location of the flight is altered (by changing *Environment.latitude* or *Environment.longitude*), the parachute class *mainTrigger(p, y)* will need to be updated to function correctly. Either remove the parachutes from the rocket or replace the '1471' in the *mainTrigger(p, y)* class with the altitude in metres of the new flight location.

Once all the script is executed, two types of csv files will be generated:

1. 'id'.csv (e.g. 3.csv) – A csv file will be generated for each trajectory simulation that is executed. This contains the datapoints and flight data for a single simulation. Each row of the data is a single point in time (set through the *timestep* function parameter), with data such as x,y,z, positions and velocity data.
2. all_inputs.csv – A csv file containing various input and output variables for each rocket flight. Each row is a single simulation, denoted by the 'id' column and corresponds to the 'id' of the csv file names. It includes data such as rocket mass, flight time and flight phases.

For a description of each variable included in the csv files please refer to the official RocketPy documentation.

4. Formatting of Simulation Data

The following formatting is required for the simulation csv data to ensure it functions with the project:

- 'id'.csv (e.g. 3.csv) - The filename of each simulation can be altered through the *idCount* variable. By convention this begins at '1' and increments for each simulation. However, it is recommended that this is not altered, as the project relies on the file name to link the trajectory data to the row in 'all_inputs.csv' file (i.e. 3.csv is linked with row where id=3)
- all_inputs.csv – The project requires this file to include the 'id' column of the trajectories as the first column of the file.

5. Unity Folder Structure for Simulation Data

The csv files containing the simulation data can be placed anywhere within the Unity project, however the file path must be added to any DataFiles and LoadInputVariables scripts within the Unity Editor. For consistency, we have been placing the csv data using the following structure:

Swordfish_IT_Project -> Assets -> Resources -> MainData -> All trajectory csv files

-> inputData -> input variables csv

6. Further Help

For any further enquiries into the RocketPy software, it is worth reaching out to the official RocketPy development team. The best way to do this is through the official Discord server (Link can be found on front page of [RocketPy documentation](#)). The development team are helpful and respond promptly to all questions.

5. Major Components

Getting RocketPy Data into Unity

Related files:

- DataFiles.cs
- LoadInputVariables.cs

DataFiles.cs and LoadInputVariables.cs are the two scripts which handle the loading of csv data at runtime. DataFiles.cs is primarily used for the trajectory visualisation and LoadInputVariables for the bar chart.

DataFiles.cs

This is the main script for loading the csv trajectory data and creating the trajectory objects. The script should be added to an object placed within the trajectory visualisation object. The following variables need to be set within the Unity Editor before runtime:

- Visualisation (*Visualisation.cs*) – The IATK trajectory *Visualisation* object.
- Data Point Prefab (*DataPoint.prefab*) – The *DataPoint* prefab object for creating each individual data point.
- Rocket (*RocketAnimation.cs*) – The rocket model object which should contain the *RocketAnimation* script.
- Legend Item Prefab (*Legend.prefab*) – The *VRTK4_UI Canvas* object for adding the latitude and longitude data to the visualisation.
- Folder (*String*) – The folder within ‘Assets/Resources’ containing the trajectory csv data. This is currently ‘MainData’.

The process DataFiles.cs uses to create the trajectory visualisation objects is as follows:

1. For each trajectory csv file, create a *CSVDataSource* object and add the trajectory data.
2. Find the minimum and maximum value for each column of every *CSVDataSource* object data. This is used for normalising the data so that all trajectories can be accurately rendered based on dimensions.
3. For each *CSVDataSource*, normalise the data and create a trajectory object (*VisualisationLine* & *VisualisationPoints*).
4. Update the graph axis values.
5. Update the colour coding legend.

LoadInputVariables.cs

Loads the *all_inputs.csv* csv file and creates the bar chart. This script is added to the *BarGraphGenerator* object within the Unity Editor. The following variables need to be set within the Unity Editor before runtime:

- Trajectory Files (*DataFiles.cs*) – The *DataFiles* object used for the trajectory visualisation.
- Id Col (*int*) – The column index in *all_inputs.csv* which corresponds to the simulation id.
- Axis X (*String*) – Sets the bar chart x-axis. Name of the column in *all_inputs.csv*.
- Axis Y (*String*) – Sets the bar chart y-axis. Name of the column in *all_inputs.csv*.
- Axis Z (*String*) – Sets the bar chart z-axis. Name of the column in *all_inputs.csv*.
NOTE If a 2D bar chart is desired, leave this variable empty.
- Graph Factor (*int*) – Size multiplier for the size of the bar chart. Default is 1.
- Graph Size (*Float*) – In-game size of the bar chart. Default is 0.164.

- Bar Color (*Unity.Color*) – Colour for first z-axis of bars. Each consequent z-axis of bars slightly alters the colour to simulate a gradient.

The process of *LoadInputVariables.cs* for creating a bar chart is as follows:

1. Create a *CSVDataSource* object for the data from *all_inputs.csv*.
2. Using the *IBC* package, create either a 2D or 3D bar chart.

Trajectory Visualisation/Scatterplot

Related files:

- *DataPoint.cs*
 - *BillboardBehaviour.cs*
 - *ConnectorLink.cs*
- *VisualisationPoints.cs*
- *VisualisationLine.cs*

DataPoint.cs

This script is on the *DataPoint* prefab in Assets > Swordfish > Resources > Prefabs. *DataPoint.cs* represents one single point in time within the trajectory, or one single row within the CSV data. It reads in the single row of data as a string, which is then shown on the UI display that is instantiated only when the user selects the data point (for performance). The data point changes colour based on the hover and selected state. The following variable must be set before runtime in the Unity Editor:

- Hover Material – The material to set the *DataPoint* as to indicate a hover state when the *SelectionCollider* on the controller cones enter the *DataPoint* collider
- Selected Material – The material to set the *DataPoint* as to indicate a selected state when *Select()* is called from *SelectionCollider.cs* to show the data display for the *DataPoint*

BillboardBehaviour.cs on the UI display ensures that the display is always facing the VR headset. *ConnectorLink.cs* controls the *LineRenderer* to always connect the *DataPoint* GameObject to the bar of the UI display as it rotates. Future work could be done to make the bar grabbable so that the UI display can be moved. *ConnectorLink.cs* would then ensure the *LineRenderer* is always connected to the *DataPoint* and UI display.

VisualisationPoints.cs

This script is added to a new GameObject created at runtime within *DataFiles.cs* when creating a new trajectory in *CreateTrajectory()*. *VisualisationPoints.cs* is then responsible for actually creating all the *DataPoint* Gameobjects. It creates the whole trajectory as singular GameObjects so that each point is interactable. IATK visualises the data as a singular mesh, which is not the desired form for interactivity with individual data points. Thus, the generated mesh from IATK is passed into *VisualisationPoints.cs* from *DataFiles.cs*, to which data points are then created based on each vertex in the mesh. It also accounts for the different flight phases and colour codes the data points accordingly. When *DataFiles.cs* creates the new GameObject with the *VisualisationPoints* component, it sets the following variables,

- *.setVisualisationMesh(BigMesh mesh)* – The *BigMesh* component that IATK automatically generates which visualises the data as a singular mesh.
- *.setDataPointPrefab(GameObject dataPointPrefab)* – The *DataPoint* prefab object for creating each individual data point.

- `.setPointMaterial(Material mat)` – The Material to use for the *DataPoint*.

VisualisationLine.cs

VisualisationLine.cs represents a trajectory as a line using *LineRenderer*. In a similar manner, it is added to a new *GameObject* created at runtime within *DataFiles.cs* when creating a new trajectory in *CreateTrajectory()*. *DataFiles.cs* sets the generated mesh from IATK which is then used as the vertex positions for the *LineRenderer*.

- `.setVisualisationMesh(BigMesh mesh)` – The *BigMesh* component that IATK automatically generates which visualises the data as a singular mesh.
- `.setLineMaterial(Material mat)` – The Material to use for the *LineRenderer*.

Trajectory Animation

Related files:

- *RocketAnimation.cs*
- *RocketAnimationUI.cs*

An animation feature has also been implemented to play an animated rocket across the trajectories in the trajectory visualisation. Users can control the playback using the UI buttons and slider, as well as select which trajectory to play the animation on by using the UI dropdown or walking up to the visualisation and selecting a *DataPoint* using the Primary button (A/X button) – to which the animation will play across the corresponding trajectory.

RocketAnimation.cs

This script on the Rocket *GameObject* controls the animation process. It essentially moves the rocket along the trajectory by following and using the positions of the data points in the trajectory whilst considering appropriate rotation to point towards the next *DataPoint*. The following variable must be set before runtime in the Unity Editor:

- Data Objects – The *DataFiles* object containing the trajectory data.

RocketAnimationUI.cs

This script also on the Rocket *GameObject* is responsible for connecting the UI components to the *RocketAnimation* functionalities. This includes playback controls and slider for the animation, populating the UI dropdown list of selectable trajectories and showing the values of the *DataPoint* that the rocket is currently at.

- Trajectory Source UI Dropdown – UI Dropdown component to populate with trajectories to select from.
- Trajectory UI Slider – UI Slider component to control the animation.
- Data Displays – *GameObjects* containing a child *Text* component to display the values of the *DataPoint* that the rocket is currently at.

Trajectory Filtering

Trajectories can be filtered (hidden/unhidden) based upon input variables. The *Filtering.cs* contains the logic to achieve this. It is attached to the scatterplot visualisation object (i.e. *Visualisation.cs*). The following variable must be set before runtime in the Unity Editor:

- Files (*DataFiles.cs*) – The *DataFiles* object containing the trajectory data.

The following variables are used for trajectory filtering:

- Filter Variable - The rocket variable to filter upon. This is a dropdown which is updated at runtime based upon the variables in the *'id'.csv* files.
- Filter Type – How to filter the trajectories:
 - Hide Above – Hides all the trajectories where the Filter Variable is above a given value.
 - Hide Below – Hides all the trajectories where the Filter Variable is below a given value.
 - Between – Hides all the trajectories where the Filter Variable is outside the given range (min-max).
- Filter Value– The value to filter upon.

One potential area for future work would be allowing users to filter based upon all simulation variables. This could be implemented by loading the *all_inputs.csv* file within *Filtering.cs*

FilteringUI.cs

Also on the visualisation object is *FilteringUI.cs* which connects the UI components to the logic in *Filtering.cs*. The following variable must be set before runtime in the Unity Editor:

- Filter Var Dropdown – UI Dropdown component to populate with all the output variables that can be filtered on (i.e. possible values of Filter Variable)
- Filter Type Dropdown – UI Dropdown component of possible Filter Types
- Single Filter Input Field – UI Input Field component to input a single value to filter on. This component will only be active when the Filter Type Dropdown is on “Hide Above” or “Hide Below”.
- Min Filter Input Field – UI Input Field component to input the minimum value to filter on. This component will only be active when the Filter Type Dropdown is on “Between”.
- Max Filter Input Field – UI Input Field component to input the maximum value to filter on. This component will only be active when the Filter Type Dropdown is on “Between”.
- Keypad – GameObject with *Keypad.cs* and child UI Buttons to represent the keypad. Each UI button has the *OnClick()* set to call *Keypad.appendValue(int value)* with the value.

Bar Chart

Useful Links

- [3D Interactive Barchart GitHub](#)
- [3D Interactive Barchart Unity Storepage](#)
- [Creating a barchart guide](#)

The 2D and 3D bar chart is implemented using the 3D Interactive Barchart Unity package. It is worth following the ‘Creating a barchart guide’ pdf linked above to learn how to generate a bar object within Unity. We are using a customised version of the package to create a bar chart and link the data to the

trajectory visualisation. To gain an understanding of how bar charts are generated, it is worth looking through the following package scripts: *BarGraphGenerator.cs*, *BarGraphManager.cs* & *GraphBox.cs*

To create a bar chart Unity object, create an empty game object and add two scripts:

- *BarGraphGenerator.cs*
- *LoadInputVariables.cs*

For a guide on the setting the variables for *LoadInputVariables.cs* in the Unity Editor, refer to section '[Getting RocketPy Data into Unity](#)'.

The black plane on the bar chart is also grabbable using the cones on the controllers and the trigger button. It is constrained on the Y-axis to only allow movement up and down within the bar chart. This is achieved through the Assets/VC/3D Interactive Bar Chart/3.Prefabs/BarGraph/GraphBox.prefab which has a BottomStopper and TopStopper that the plane collides with and is blocked by. The grabbable plane GraphBox > XZ Plane > Interactions.Interactable_Plane has a Rigidbody component that freezes the rotation and X, Z axis movement.

Linked Highlighting

Related files:

- *ChartLinkingManager.cs*
- *ThirdLevelChartLinkingManager.cs*

The trajectory visualisation and bar chart can also be linked so that highlighting a specific input value from the bar chart will highlight its corresponding output trajectory in the trajectory visualisation.

ChartLinkingManager.cs

This script is found on the topmost parent GameObject that contains the trajectory visualisation and **2D** bar chart as its children. It handles highlighting bars and trajectories according to their IDs based on the UI value of the slider component. For example, if the UI slider is set to the value of 22 to represent 22kg of the input variable of rocket mass, the script will search the input variable CSV file to find 22 as the rocket mass and the corresponding ID for the run. It then highlights the bars and trajectory based on the ID found. The bars store their IDs in *BarProperty.cs* and the trajectories get their ID based on the CSV file name they are generated from – which is set to the ID. The following variable must be set before runtime in the Unity Editor:

- Slider – UI Slider component to control the highlighting. It must contain a *UISliderStep.cs* component that forces the slider to be a discrete slider. The script automatically then sets the possible discrete values to correspond to the possible values of the input variable based on the input CSV file.

ThirdLevelChartLinkingManager.cs

This script is used instead of *ChartLinkingManager.cs* if the bar chart is **3D**. It inherits from *ChartLinkingManager.cs* and is also responsible for highlighting but accounts for the addition of the second abstracted variable. It highlights along each axis of the bar chart individually or together or a single bar at a time. The following variable must be set before runtime in the Unity Editor:

- Slider – Leave empty/None

- X Slider – UI Slider component to control highlighting for the first abstracted variable/X axis variable on the bar chart.
- Z Slider – UI Slider component to control highlighting for the second abstracted variable/Z axis variable on the bar chart.
- Single Slider – UI Slider component to control highlighting for a single bar at a time.

6. How-Tos

How To: Create a Trajectory Visualisation Object

1. Create empty game object and attach following scripts:

- *Visualisation*
- *ScatterplotVisualisation*

2. Drag a 'Legend' prefab into the visualisation object.

3. Create an empty child object within the Visualisation object and attach the *View* script.

4. Create an empty child object within the View object and attach the following components/scripts:

- Mesh Filter
- Mesh Renderer: Set a basic material to the Materials panel.
- *Big Mesh*: Set the same material as above.

5. Create a new game object and add *CSVDataSource* script. Add a single csv file to the 'data' section.

6. Within the Visualisation object, set the *CSVDataSource* object to the 'Data Source' option. Choose 'SCATTERPLOT' in 'Visualisation Type'. Set the three axis. Set 'geometry' to 'spheres'. At this point three axis objects should be created as children within the Visualisation object. You should also see a basic mesh rendering of the data in the scene.

7. In the Visualisation object, select 'Data Source' and change it to 'none'.

8. Hide/delete the *CsvDataSource* object.

9. Create an empty child object within the Visualisation object and attach the *DataFiles* script. Set the following within the DataFiles editor:


- Visualisation – The visualisation object
- Data Point Prefab – The data point prefab named 'DataPoint'
- Legend Item Prefab – The Legend object

10. Run the project, you should now see all data files being rendered within the visualisation.

How To: Setup VRTK Camera and Input Buttons

1. The VRMainCamera object in the *Main.unity* scene has been setup according to the VRTK guides [here](#) and [here](#), with additions made such as the UI Pointer and custom selection cone. Thus, it is recommended that if a new scene is created, the VRMainCamera object from the *Main.unity* scene is made into a prefab and reused in the new scene to ensure that all functionalities remain.
2. To change the *DataPoint* and *Bar* selection action used to show its display:
 - a. VRMainCamera > CameraRigs.TrackedAlias > Aliases > LeftControllerAlias > Interactions.Interactor > AvatarContainer > ExampleAvater > Cone > SelectionPoint > SelectionCollider.cs > Data Point Selection Action
 - b. Drag in desired input action from VRMainCamera > UnityInputSystem.Mappings.GenericXR > InputActions > LeftController
 - c. Repeat with RightControllerAlias, choosing desired input action from the RightController components
3. To change the action used to select a *DataPoint* for selecting its trajectory to animate over:
 - a. VRMainCamera > CameraRigs.TrackedAlias > Aliases > LeftControllerAlias > Interactions.Interactor > AvatarContainer > ExampleAvater > Cone > SelectionPoint > SelectionCollider.cs > Trajectory Animation Selection Action
 - b. Drag in desired input action from VRMainCamera > UnityInputSystem.Mappings.GenericXR > InputActions > LeftController/RightController
4. To change the actions used for the UI pointer interactions:
 - a. VRMainCamera > CameraRigs.TrackedAlias > Aliases > LeftControllerAlias > Interactions.Interactor > [L_R]_ UI Pointer on Interactor > VRTK4_UI Pointer.cs
 - b. Change the Activation Button and/or Selection Button to the desired Boolean Action from VRMainCamera > UnityInputSystem.Mappings.GenericXR > InputActions.
 - c. Mirror the change on VRMainCamera > CameraRigs.TrackedAlias > Aliases > LeftControllerAlias > Interactions.Interactor > Indicators.ObjectPointers.Straight > PointerFacade.cs with the same Activation Action and/or Selection Action
 - d. For further information, refer to this [guide](#) for the project used to implement the UI pointer and how it is setup.
5. To change the action used to grab interactable objects:
 - a. VRMainCamera > CameraRigs.TrackedAlias > Aliases > LeftControllerAlias > Interactions.Interactor > InteractorFacade.cs > Grab Action
 - b. Repeat with RightControllerAlias
 - c. For further information, refer to this [guide](#) on how the controller interactor is set up to interact with other interactable objects.
6. The left joystick movement on the controller cannot be changed from the left controller as it has been explicitly defined/implemented by VRTK. Refer to this [guide](#) on how it is set up.

7. Known Limitations

- Currently, the bar chart can be grabbed via the trigger button and . This functionality works by the GrabBarGraph.cs on Third-Level > Interactions.Interactable_BarGraph > MeshContainer > Cube GameObject. On grab of this Cube object, it parents the BarGraph object as a child of the Cube. On release, it unchilds the BarGraph object. Further work should look into whether this is

the best way to implement the grab feature in terms of user experience. It also has not been implemented for the trajectory visualisation.

- It can be slightly difficult to grab onto the black plane on the bar chart. Further work needs to investigate appropriate collider sizes. In addition, if the user grabs the bar chart and rotates it, grabbing and moving the black plane results in incorrect positioning and rotation of the plane. This may be due to the Rigidbody imposing the constraints on a world level, and should instead be on a local level within the bar chart.
- Filtering is on filters the trajectories on the trajectory visualisation. Future implementation may investigate applying the filtering to the bar chart so that unrelated bars are hidden as well.
- Creating a new Visualisation object requires the user to set the axis variables in the Unity Editor by creating a *CSVDataSource* object (As discussed in [this section](#)). This is a carryover method from the IATK software package and the creation process should be streamlined.

8. Roadmap

Here are some ideas for future system features:

- **Exporting the Unity visualisations as 3D models.** In a presentation to DSTG researchers this was highlighted as an important feature they would like to see.
- **User validation and testing.** In a presentation to Swordfish Computing employees it was suggested that it would be valuable to demo and interview users of the system (i.e. DSTG researchers). It is important to get their feedback.
- **Improve the movement and usability of the system.** We found while developing that it was often easier to just use the keyboard/mouse in Unity Editor to move around the scene rather than putting on the headset.
- **Performance Optimisations.** Investigate rendering fewer individual data points. Many data points in the second half of a trajectory are bunched together, some of these could be removed. Additionally, each data point has it's own box collider. It is worth investigating disabling the box colliders when they are not needed (e.g. when the user is a certain distance from the data points. The system is currently CPU bound, so any improvements in this area will drastically improve FPS.
- **Further rocket visualisations.** RocketPy has large amounts of data for each simulation. Could look into creating further visualisations of these such as velocity, thrust etc.
- **Weather visualisations.** Investigate how weather can be integrated into the visualisations. This would be very valuable for researchers to understand the impact weather/environment plays on rocket launches. RocketPy calls an external API to get current weather data. Investigate how this can be exported and imported in the Unity project.
- **Streamline the creation of creating an IATK visualisation in Unity Editor.** This will require refactoring of IATK source code.

9. Team

This project is developed by:

- Luke Gerschwitz (luke.gerschwitz@gmail.com)
- Elyssa Yeo (elyssayeo16@gmail.com)