DEGREE PROJECT IN ENGINEERING PHYSICS,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2017

Real-Time Visualization of Flight Data for Suborbital Missions

JOHAN BERGLUND

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING

Real-Time Visualization of Flight Data for Suborbital Missions

Johan Berglund
johanbe8@kth.se

A thesis presented for the degree of
Master of Science

KTH Electrical Engineering
KTH Royal Institute of Technology
Sweden

June 26, 2017

Supervisors	*Examiner*
Marcus Lindh	Tomas Karlsson
Nickolay Ivchenko	

Abstract

The aim of this thesis was to systematically develop a tool used for the visualization of real-time flight data during suborbital mission, on behalf of the Swedish Space Corporation. By focusing on requirements, restrictions, and trade-offs regarding functionality, communications technologies, geospatial visualization platforms, and many other aspects, an interactive product visualizing the position, attitude, speed, G-loads and angular rates of the rockets could be developed, ensuring a visualization as close to real-time as possible. The resulting product was tested during the flight of the MAXUS 9 rocket, Europe's largest sounding rocket, and was subsequently refined based on the results of that flight.

Sammanfattning

Målet med detta examensarbete vara att utveckla ett verktyg för visualisering av flygdata i realtid under suborbitala uppdrag, på uppdrag av Swedish Space Corporation. Genom att fokusera på krav, restriktioner, och avvägningar gällande funktionalitet, kommunikationstekniker, geospatiell utveckling, och många andra aspekter, kunde en interaktiv produkt som visualiserar raketens position, attityd, hastighet, G-krafter och vinkelhastigheter utvecklas, som även säkerställer att visualiseringen sker så nära realtid som möjligt. Den resulterande produkten testades under uppskjutningen av MAXUS 9, vilket är Europas största sondraket, och förfinades sedan baserat på resultaten av den flygningen.

Acknowledgements

I would like to thank the Swedish Space Corporation for giving me the opportunity to work on this thesis, and also to allow me to join them for the launch of the MAXUS 9 rocket, which was an experience worth remembering. An especially big thank you goes to my supervisor at SSC, Marcus Lindh, for his help, support, and unwavering enthusiasm, which has been invaluable for this project. Furthermore, I would like to thank my supervisor at KTH, Dr. Nickolay Ivchenko, who have guided me in the process of writing this thesis.

My gratitude also goes to my parents, Monica and Bosse Berglund, for supporting and believing in me throughout my childhood, and for nurturing my interest in technology.

Contents

List of Figures

1 Introduction

The use of microgravity, which is a state obtained when an object is in free fall, with no external forces acting on it, has proven to be very important as a tool used to understand phenomena within physics, material science, biology, and many other areas. It can be achieved using a number of different platforms, depending on the time of microgravity needed. Satellites and space stations are used when the duration of continuous microgravity needed is high, whereas experiments where only a limited time is needed can make use of for example drop towers, parabolic flights, or sounding rockets.

Sounding rockets can achieve microgravity on the timescale of a few minutes. During this time, telemetry data such as gyro, accelerometer, GPS position and velocity data is transmitted to the ground station, where it is displayed numerically. However, it is crucial to be able to intuitively gain an understanding of the conditions and dynamics of the rocket in real-time, in order to have the possibility to adjust the control systems in case of non-optimal performance. The display of numerical values does often not fill this function, but would need to be complemented with some sort of data visualization in order obtain a full picture of how the rocket is behaving.

1.1 The Swedish Space Corporation

The Swedish Space Corporation, or SSC, was founded in 1972, and has since then developed 8 scientific satellites, more than 60 rocket vehicles, and 60 scientific payloads, as well as launched 550 rockets and 520 balloons [5]. Today, SSC offers expertise in the development of satellite subsystems as well as in satellite communication and operations. They also develop rocket systems and experiment payloads, and launch sounding rockets and high-altitude balloons from their launch site Esrange, located in northern Sweden.

1.2 Previous work

Most companies launching rockets use some kind of data visualization. The earliest kind of real-time data visualization for rockets mostly concerned the position data, and one of the companies that were first with showing that kind of visualization, which is shown in figure 1, to the public was the United Launch Alliance, or ULA. In recent years, the public display of data has become more common. SpaceX is one of the companies that uses live data visualization during their launches for the purpose of publicity. They use a combination between live cameras and real-time data overlays, as well as a 3D globe showing the trajectory when cameras are unable to follow the rocket anymore, as shown in figure 2. Another company that utilizes real-time data visualization is Arianespace. In figure 3, a visualization of the Soyuz launch from May 18 2017 is presented, showing both a 3D model above Earth, trajectory and altitude curve, as well as a dashboard displaying some numerical data.

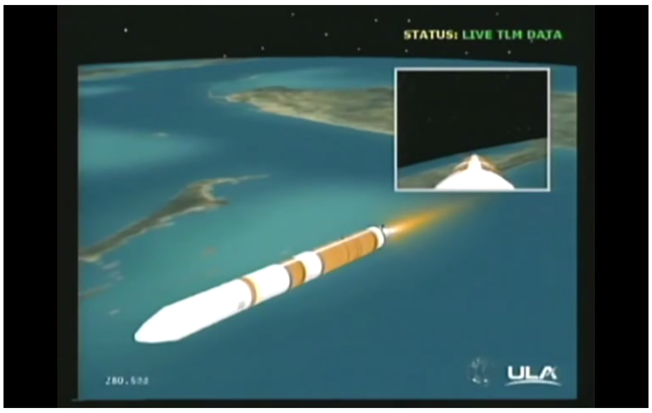
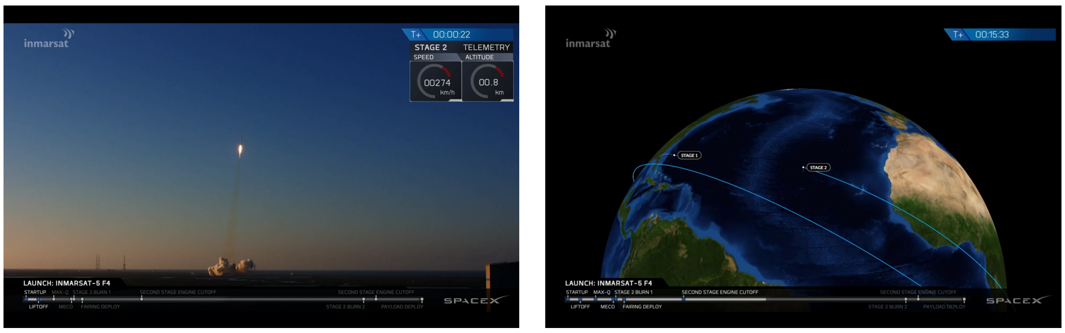


Figure 1: Early real-time data visualization performed by ULA.



(a) Live camera with data overlay

(b) Rocket trajectory

Figure 2: Data visualization performed by SpaceX

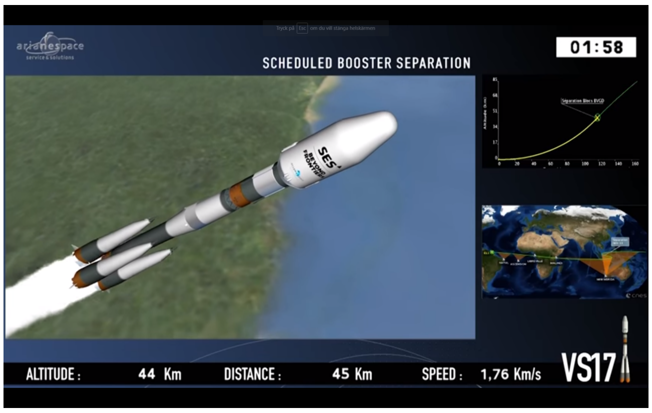


Figure 3: Data visualization performed by Arianespace.

1.3 Objective

To provide an intuitive way of interpreting the behavior of the sounding rockets launched by SSC, this thesis aims to develop a tool for the graphical visualization of flight data. Data of interest includes the GPS position, gyro and accelerometer data, attitude, and velocity, which would be extracted from a local network, and subsequently processed before being visualized.

A working prototype needed to be finished before the launch of the MAXUS 9 rocket, which is Europe's largest sounding rocket, in the beginning of April 2017, at which point the product was supposed to be thoroughly tested. The results of this test would then serve as a base for improving and tuning the final product.

2 Requirement specifications

In order to ensure that the product of this project is satisfactory, a requirement analysis has been done. The functional requirements stated by SSC were combined with operational and performance requirements, which are stated in the sections below. Each requirement is coupled with a requirement code, with which the specific requirement can be referred to with in later sections.

2.1 Functional requirements

The functional requirements describe what the product shall do in order to function as indented. Since the product of this project is a visualization application, all functional requirements are related to what the product shall visualize, and were identified as:

F.1 The product shall visualize the GPS position of the rocket.

F.2 The product shall visualize the attitude of the rocket.

F.3 The product shall visualize the gyro data received.

F.4 The product shall visualize the accelerometer data received.

F.5 The product shall visualize the speed of the rocket in a dashboard style.

F.6 The product shall support a countdown clock overlay.

F.7 The product shall support the display of events at correct times.

F.8 The product shall plot the trajectory of the rocket.

F.9 The product shall visualize the ground track of the rocket.

F.10 The product shall visualize the instantaneous impact point of the rocket.

The verification process of each of these requirements involves two main parts. First, it must be verified that the product is capable of visualizing the specific property, and then it must be verified that the value of that property is true. Only then can the requirement related to that property be considered as verified.

2.2 Operational requirements

The operational requirements state the essential capabilities of the product, which mostly corresponds to the processing of data needed. Stated below are the operational requirements identified.

O.1 Data shall be extracted by a network sniffer.

O.2 The speed of the rocket shall be calculated based on the GPS positions received.

O.3 The data shall be processed to match the requirements on the data structure.

O.4 The product shall handle the absence of new data by sending the last known data points.

O.5 The product shall notify the user whether the data is being received from the network sniffer, and if that data contains telemetry or GPS data.

For the operational requirements, the verification process includes checking so that certain scenarios are handled as expected. For example, for requirement O.4 to be verified, it is necessary to simulate interruptions in the data stream received and evaluate the response of the product.

2.3 Performance requirements

The performance requirements state any requirements related to how the product must perform in order to be satisfactory. In this project, these requirements were identified as:

P.1 The received data shall be collected, and subsequently sent to the product with a frequency not higher than 2 Hz, and not lower than 1 Hz.

P.2 The product shall not deviate more than 4 seconds from true real-time.

These requirements can only be verified by testing the product live, since the performance must be measured from the reception of data to the final visualization.

3 Development

3.1 Development environment

In order to find a suitable environment in which to develop the visualization application, a number of popular libraries and graphics engines have been investigated. Advantages and drawbacks were considered, and are summarized in table 1.

Google Earth API
In 2008, Google introduced the Google Earth API [10], which was built on a technology called Netscape Plugin Application Programming Interface, or NPAPI, and enabled developers to use JavaScript in order to modify and interact with the Google Earth browser. This library has been very popular for developing apps that utilizes geospatial data. Unfortunately, the Google Earth API has been deprecated due to a combination of a decrease in cross-platform support and the fact that Chrome and Firefox removed the support for NPAPI due to security reasons [11]. Therefor, this library was not considered an option.

Cesium
Cesium is an open source, JavaScript-based library, commonly used for visualizing time-dynamic geospatial data, and was founded by AGI, which is a company that has developed numerous programs for air, space and defense applications [2], in 2011, and was initially intended for dynamic data visualization in the space and defense industry. However, since the launch, Cesium has grown to be used by a range of different industries [4]. Like with most open source libraries, Cesium is dependent on the community, which has not been very extensive so far. However, with the deprecation of the Google Earth API, many developers migrated to Cesium, resulting in an increase in activity within the Cesium community.

ArcGIS
ArcGIS is a software specialized in the visualization of geospatial data, without the need of extensive programming. It comes with pre-built tools for mapping, visualizing and analyzing both static and dynamic data [7]. ArcGIS is not open source, but it does come with a free version for developers, although with limited capabilities.

Unity3D
As a graphics engine commonly used for games, Unity3D is a very powerful tool for creating eye-catching visualizations. It has a very large user base, and the active community can be very helpful. However, it does not have any inbuilt support for importing maps, and the visualization of geospatial data would require extra development time, since the API used in Unity3D is not built for that purpose. Like ArcGIS, Unity3D is not open source, and it can be free to use, but with limited capabilities [21].

XNA Game Studio
XNA Game Studio, which is developed by Microsoft, is quite similar to Unity3D in terms of visualizaton capabilities. They are both powerful tools for working with graphics, and have basically the same advantages and drawback compared to the tools that are developed specif-

ically for geospatial data visualization. The main differences are that XNA Game Studio are completely free to use, but has a much smaller user base, and hence user community, than Unity3D.

Table 1: Properties of the different development environments

Environment	Open source	Map provider	Free	Deprecated
Google Earth API	No	Google	Yes	Yes
Cesium	Yes	Multiple	Yes	No
ArcGIS	No	Multiple	Yes, limited functionality	No
Unity3D	No	None	Yes, limited functionality	No
XNA Game Studio	No	None	Yes	No

It was finally decided that the development environment that were to be used for this project was going to be Cesium. The fact that it is open source, along with the focus on time-dynamic data visualization was the deciding factor. Both the use of Unity3D and XNA Game Studio would prolong the development time, and the already tight time-line before the launch of the MAXUS 9 rocket would not allow that. Furthermore, while the ArcGIS software is quite similar to Cesium in terms of the intended use, Cesium has the economic advantage and is more flexible since it is open source.

3.2 Data management

3.2.1 Time stamps

Since Cesium is a library focused on temporal geospatial data, time stamping is crucial to have a smooth visualization. The GPS data already have a time stamp corresponding to each data point, and can be extracted along with the data itself. However, since no other data set has its own time stamps, the different data sets have to be synced together in order to ensure that the data being visualized are all defined at the same time. This can be done in several ways, one possibility is to use the local system time of the computer being used for the program for all data sets, including the GPS data. That way, they would all be defined with respect to the same time system. Another way is to use the time stamp on the first GPS data received as a base, and convert the time stamps for the other sets to that same time system as

$$t = t_{base,GPS} + (t_{now,system} - t_{base,system}) \tag{1}$$

where the two base times, $t_{base,GPS}$ and $t_{base,system}$, are the base time set from the GPS data and the corresponding base time in the local system time, defined as the system time at the moment the GPS data base time was defined. Finally, $t_{now,system}$ is the local system time of the reception of the data point.

The time received from the GPS data also needs some processing before use. The format it is received in is hhmmss.sss, that is, it contains hours, minutes, seconds and milliseconds. Since sounding rocket missions span over a time of a few minutes, that information is all that is needed. However, Cesium also requires the year, month and day in order to interpret the time stamps. To solve this, the time stamps for the GPS data was modified to contain the year, month and day from the local system time, while the hours, minutes, seconds, and milliseconds remained unchanged. The resulting GPS time stamp could look like the following:

$$t = t_{year,system} + t_{month,system} + t_{day,system} + t_{GPS} \tag{2}$$

3.2.2 Pre-processing

The format used by the Cesium in order to stream data is called CZML, which is a subset of the JSON format [1]. Each CZML document contains a single JSON array, where each object in the array is a CZML packet. Every packet has an id property, used to identify the object described by the packet. By referring to this id, it is possible update the properties of the object when e.g., streaming the CZML packages. There are many standard properties defined for CZML, however, only a few are of interest during this project.

Several properties that are to be visualized need to be time stamped using the ISO8601 time format [8], written as a string. The time stamping can either be done for each sample, or, if several samples are being sent in the same packet, by defining an epoch and having the time of each sample being measured in milliseconds since that point of time. Time stamps are not necessary if the data is to updated at a lower frequency, so that only one value per data push is visualized.

Position

The longitude, latitude, and altitude can all be described by the ”position” property in the CZML format, in which the cartographic values are specified in an array as [Time, Longitude, Latitude, Altitude], where ”Time” is the time stamp. Below is an example of the position property of a packet containing two samples, with a sampling time of 100 ms.

```
"position": {
  "interpolationAlgorithm":"LAGRANGE",
  "interpolationDegree":1,
  "epoch":"2017-04-02T16:00:00Z",
    "cartographicDegrees":[
     0,21.1068944441480,67.8932469830635,362813,
     100,21.1068945441480,67.8932469830635,362913
  ]
}
```

In the above example, the type of interpolation algorithm is also included, and it is even possible to include extrapolation, if desired.

Orientation
The orientation property describes the attitude of the rocket, and is constructed the same way as the position property, only using quaternions instead of cartographic values. The attitude data has to be processed according to section 4.2 before added to the CZML packet. Using the same example scenario as before:

```
"orientation": {
  "interpolationAlgorithm":"LINEAR",
  "interpolationDegree":1,
  "epoch":"2017-04-02T16:00:00Z",
     "unitQuaternion":[
     0,0.456521883683,-0.0495800359952,-0.88193443594,0.106401317853,
     100,0.3096885260,-0.0592870464532,-0.94528388650,0.083764179757
  ]
}
```

Model URL
It is also necessary to send the model which is to be used by the visualization program. This can be done by attaching an URL to a 3D-model to the model packet, as shown below:

```
"model": {
  "gltf":modelURL,
  "scale":1
}
```

Speed
The speed will be calculated according to section 4.1.1, and its visualization will be separated from the 3D-visualization, for which reason it was desirable to send it in a separate packet. In the current version of CZML, no property exists for solely pass on a numeric value, so to avoid the trouble of changing the Cesium source code to create a custom property, an invisible point can be created, with its pixel size property set to the speed value. This value can then be extracted to be used for the visualization.

```
"point": {
  "show":"false",
    "pixelSize": {
     "number":350
  }
}
```

G-loads and angular rates
Both the G-loads and angular rates are 3-component properties, and can be passed on with CZML by defining them as the Cartesian position of an invisible point.

```
"point": {
  "show":"false",
      "position":[100,100,100]
  }
```

Instantaneous impact point

The instantaneous impact point, or IIP, is the point of impact estimated for the rocket at every time step, if the propulsion would not be a factor. This property is a position, and can therefor be sent with the position property, attached to an abstract point.

```
   "point": {
     "show":"false",
       "position":[21.1068944441480,67.8932469830635,362813,0]
  }
```

Mission time and events

The mission time and events will be sent as strings, which can be done by updating the packet names of the packets every time any of these strings changes.

```
   "point": {
     "name": "eventString/TimeString",
     "show":"false",
     "pixelSize": {
        "number":350
     }
   }
```

Example data structure

An example of a complete CZML array is shown below. Note that the first packet in a document, or in a new stream, must be a document packet, stating the CZML version and name of the package. The second packet contains the data used for updating the state of the 3D model, i.e., the position, orientation, and the model URL, and the third packet contains the speed value, represented as the pixel size of an invisible point, and the G-loads in the form of the points position. To this packet, the name property has also been applied in order to extract information about events that might occur. The final data packet contains the time string as well as the angular rates, added in the same way as the G-loads.

```
   [{
     "id": "document",
     "name": "czmlExample",
     "version": "1.0"
  },{
     "id": "rocket",
     "position": {
        "interpolationAlgorithm":"LAGRANGE",
        "interpolationDegree":1,
        "epoch":"2017-04-02T16:00:00Z",
        "cartographicDegrees":[
           0.0,21.1068944441480,67.8932469830635,362813,
            100.0,21.1068945441480,67.8932469830635,362913
         ]
      },
      "orientation": {
         "interpolationAlgorithm":"LINEAR",
         "interpolationDegree":1,
         "epoch":"2017-04-02T16:00:00Z",
         "unitQuaternion":[
            0.0,0.456521883683,-0.0495800359952,-0.88193443594,0.106401317853,
            100.0,0.3096885260,-0.0592870464532,-0.94528388650,0.083764179757
         ]
      },
      "model": {
         "gltf":modelURL,
         "scale":1
      }
   },{
      "id": "event_speed_gLoads",
      "name": "eventString",
      "point": {
         "show":"false",
         "pixelSize": {
            "number":350
         }
         "position":[100,100,100]
      }
   },{
      "id": "time_angularRates",
      "name": "+00;00;00",
      "point": {
         "show":"false",
         "position":[100,100,100]
      }
    },{
      "id": "IIP_coordinates",
      "point": {
         "show":"false",
         "position":[21.1068944441480,67.8932469830635,362813,0]
      }
   }]
```

Note that it is very important that the packets are sent in the order stated in the example above, since the server will sort and store some of the properties.

3.3 Communication protocol

The data that is to be visualized is sent down from the rocket to the ground station, where it it stream out on a local network. That data is then extracted using a network sniffer, which then verifies requirement O.1, and sent to an internet based server, which in turn will send that data to the clients for the visualization to take place. Cesium is a WebGL based JavaScript library, which means that the clients in this case are the browsers being used by the users. In order to send data back and forth through various networks, it is necessary to follow some communication protocol. Today, numerous protocols meant for network data transfers are available, and presented below is a few protocols that show promise for real-time applications.

Hypertext Transfer Protocol

The Hypertext Transfer Protocol, which is more commonly know as HTTP, is a request/response protocol that operates by exchanging data over a reliable connection [13]. It works by having a client sending a request to either send, receive, delete or change data to the server, which will then send a response back to the client. HTTP is a fundamental protocol, on which many other protocols build on, and is supported by all commercial browsers, making it very important. The drawback of using HTTP includes that, for some browsers, the number of supported simultaneous connections is very limited. Furthermore, it does not support bi-directional communication without having to make heavy modifications to the protocol [13].

WebSockets

WebSockets are full-duplex, bi-directional, single socket connections [16]. The advantage of using WebSockets is that, apart from it being bi-directional, it can reuse the same first connection for subsequent data transfers. It is based on HTML5, which means that is supported by all modern browsers.

Server Sent Events

Server Sent Events, or SSE, are based on the same principles as HTTP streaming [18]. Similar to the WebSockets, it only needs to perform one connection to use for all the subsequent data transfers. It is commonly used within the Cesium Community, meaning that many examples exist to study, and it provides the possibility to automatically try to reconnect if needed. However, it is not bidirectional and can not send data upstream [16], and it is not come with native support for Internet Explorer or Microsoft Edge.

3.3.1 Local network to server

Once the flight data is extracted from the local network, it will be processed according to section 3.2.2, and then sent to the server. The application that will perform these actions was written in C#, which comes with many solutions for using HTTP requests in order to send data. That, in combination with the fact that this part of the data transfer only needs an unidirectional connection made HTTP a suitable choice.

3.3.2 Server to client

The server will have to send data through a separate connection for each user. For that reason, HTTP was excluded as an option. If bi-directional data transfers would be a requirement here, WebSocket would be a good choice, however, since the data transfer, once again, only needs to be sent in one direction, and SSE requires less implementation, it was decided to go with Server Sent Events instead. By using a so called polyfill, which is a piece of code that provides the technology lacking from certain browsers, as a fall back, that problem could be disregarded.

When the data is received at the server, it is desirable to directly route the data to the client, in order to keep as close to real-time as possible. For that reason, a custom routing method was written for the server. This method work by taking all the connected client requests and store them in an array. Whenever a request is detected from the data processor to push data, it will take that data, check if it is valid, and directly route it to all clients connected. This also comes with the advantage that it will never be necessary to store large amounts of data on the server, minimizing the risks of memory leaks, and it enables the server to quickly validate if the data source is secure or not.

In figure 4, a flow diagram is presented to visualize how the data is transferred to the rocket. By the method of trial-and-error, a push frequency of 2 Hz was found to be the sweet spot, giving some margin for spikes in processing time while still maintaining a low latency between the reception of data and its visualization. This also meant that requirement P.1 could be considered to be verified.

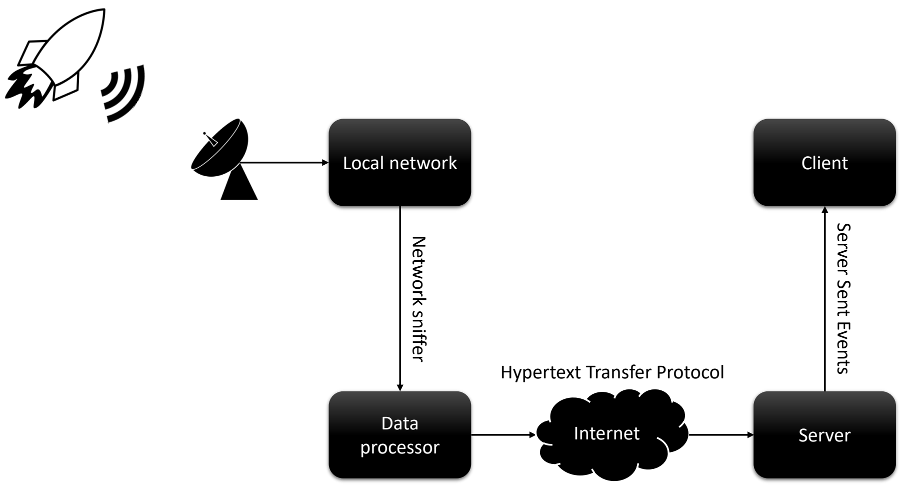


Figure 4: Flow diagram of the data transfer system

3.4 User interface

3.4.1 User controls

In order to provide a better experience for the user, it was decided to implement a set of controls that would allow the user to manipulate the view around the rocket. Cesium uses a virtual camera, which has a position and an orientation, to modify the view of the application. Cesium already provides user controls for this purpose, however, while those were functional, they did not behave in a smooth matter, as they translated the mouse movement directly to camera position. To enable the user to perform smooth control of the camera, a new system was implemented, translating the mouse movement to camera velocity, enabling the user to maintain a steady movement for a period however long desired.

A function from the Cesium API allows the virtual camera to always look towards the rocket, which means that the orientation did not have to be controlled. To control the position of the camera around the rocket, three basic user inputs were implemented. First, a zooming function was needed to move the camera towards or away from the rocket. It was decided to enable the user to zoom in and out both using the scroll wheel and the right mouse button. The zoom function using the scroll wheel was developed as a simpler zoom, similar to that of the original Cesium function, where scrolling translates directly to the amount the camera is zoomed in or out. The right mouse button, however, was configured so that, when holding it down, the scroll speed is decided as:

$$v_{zoom} = F_{zoom}(y_{mouse\_position} - y_{mouse\_start\_position}) \tag{3}$$

where $y_{mouse\_start\_position}$ is the vertical position of the mouse on the screen at the time of initiating the zoom, $y_{mouse\_position}$ its current position. Furthermore, F_{zoom} is a scale factor which is dependent on the distance d the camera has moved with respect to its initial position, and was calculated as:

$$F_{zoom} = 0.1(1 + d) \tag{4}$$

The reason for having this scale factor was to ensure a faster zoom when the camera was on a greater distance from the rocket. Finally, the actual zoom amount was calculated by integrating the zoom speed as:

$$D_{zoom} = v_{zoom}\Delta t \tag{5}$$

where Δt is the time between the rendered frames. Once the zoom distance had been determined, d could be updated as:

$$d = d + D_{zoom} \tag{6}$$

The second and third user controls needed were the two camera rotation directions around the rocket. The circulating movements were designed to be controlled using the left mouse button, using the same principle for the zoom to decide the angular velocity with which to change the cameras position:

$$\omega = F_{rotation}((x, y)_{mouse\_position} - (x, y)_{mouse\_start\_position}) \tag{7}$$

where, similar to the case for the zoom, $F_{rotation}$ is a constant scale factor that was determined empirically in order to provide a pleasant rotational speed. The actual rotation of the virtual camera around the rocket, γ, was then decided by integrating ω as:

$$\gamma = \omega \Delta t \tag{8}$$

3.4.2 GUI

The graphical user interface has to be intuitive, and visualize the data in a way that it is easy to find what the user is looking for. A number of iterations were worked through before the final design was decided on. Figure 5 shows the first concept developed, where the top of the screen consisted of numerical representations of the angular rates, G-loads, speed and mission time, as well as status diodes. The ground track and altitude curve of the rocket were placed at the bottom corners. Figure 6 shows a more developed concept, where the the speed and altitude of the rocket were visualized with two gauges, and the placement of the ground track and altitude curve had been moved in order to obtain a better overview. The angular rates and G-loads would still only be displayed as numerical values in a dashboard at the bottom of the screen, which also would contain a set of events. In figure 7, the final concept is presented, where the screen had been divided into three parts. The leftmost part contained the mission time, ground track, altitude curve, speed and G-loads, whereas the right part contained the angular rates, events, and status diodes. Instead of just being showed as numerical values, the G-loads and angular rates were now to be visualized using bars as well. Furthermore, a window dedicated to the attitude exclusively was added, which would always show the rocket from west to east, in order to enable the user to have a consistent view of the attitude. Finally, figure 8 shows the final developed GUI. In order to further enhance usability, most elements of the GUI were developed so that they could be enlarged by hovering above them with the mouse.

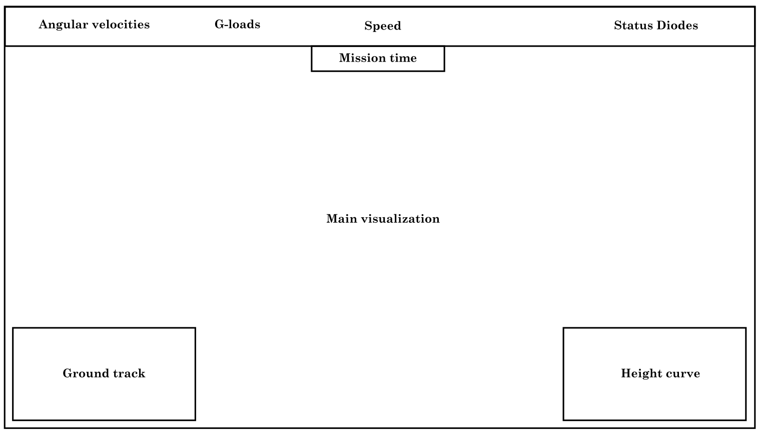


Figure 5: First iteration of GUI concepts.

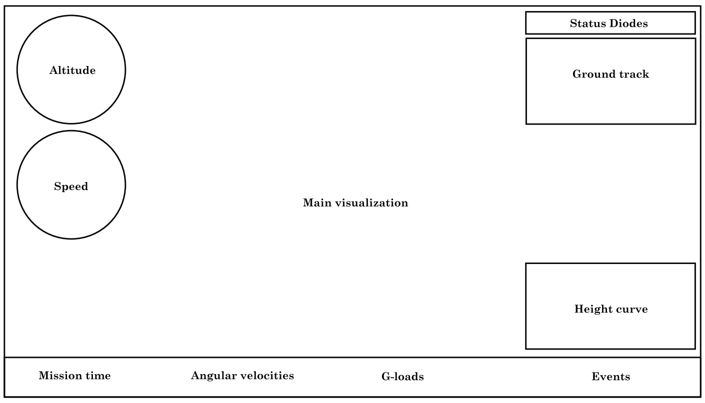


Figure 6: A sample of the GUI concepts in between the first and final concept.

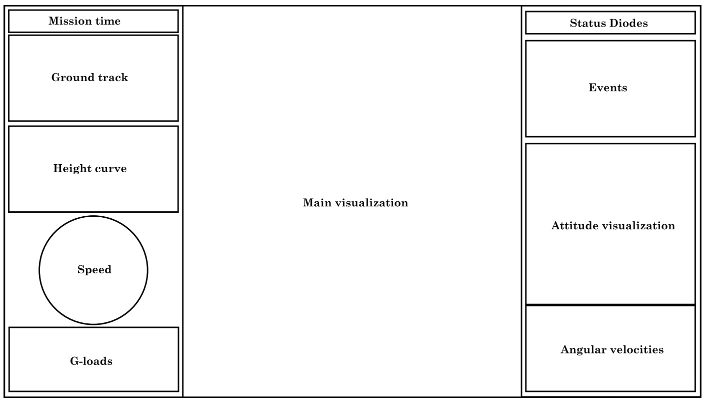
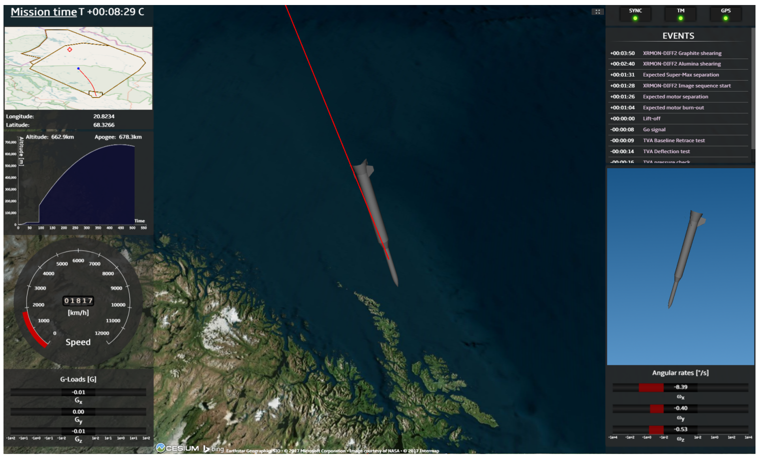


Figure 7: Final iteration of GUI concepts.



Mission time T +00:08:29 C

Longitude: 20.8234
Latitude: 68.3266

Altitude: 662.9km Apogee: 678.3km

```
+00:03:50   XRMON-DIFF2 Graphite shearing
+00:02:40   XRMON-DIFF2 Alumina shearing
+00:01:31   Expected Super-Max separation
+00:01:28   XRMON-DIFF2 Image sequence start
+00:01:26   Expected motor separation
+00:01:04   Expected motor burn-out
+00:00:00   Lift-off
-00:00:08   Go signal
-00:00:09   TVA Baseline Retrace test
-00:00:14   TVA Deflection test
```

01817 [km/h]

G_x -0.01, G_y 0.00, G_z -0.01

ω_x -8.39, ω_y -0.40, ω_z -0.53

Figure 8: Final developed GUI.

4 Theory

4.1 Geospatial data

Geospatial data is data containing explicit geographic positioning information [14]. A very common source of geospatial data is the Global Position System, or GPS, which gives information about the geodetic coordinates, that is longitude, latitude and altitude, of the object tracked. The GPS coordinates are often defined with respect to the WGS84, which is a model that describes the Earth as an ellipsoid [17].

Apart from geodetic coordinates, there are numerous standards for coordinate systems that use Cartesian coordinates. Two of these that will be used in this project are the Earth-Centered, Earth-Fixed system, denoted as ECEF, and the Topocentric Horizon system, denoted as TH. The ECEF system is a coordinate system that is centered at the center of Earth, and rotates along with it. The axes are aligned so that the Z-axis points along the axis of Earth's rotation, the X-axis points towards the prime meridian in the equatorial plande, and the Y-axis completes a right-handed system [9].

The TH system, is a system with its origin located on the surface of the WGS84 ellipsoid, having its x-axis pointing in the direction normal to the surface, the z-axis towards north, and the y-axis points so that it completes a right-handed system [22]. Figure 9 illustrates how the two coordinate systems defined.

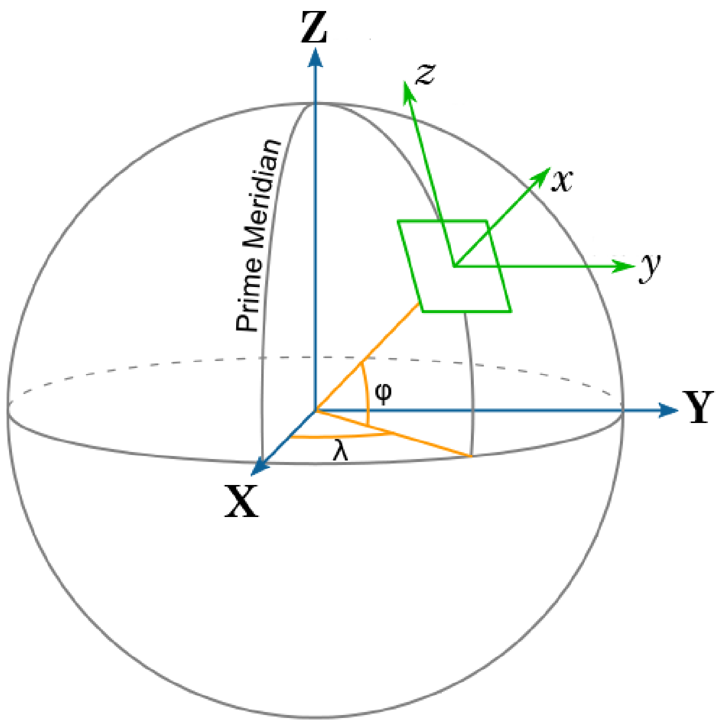


Figure 9: Rotation of the Topocentric-Horizon system with repect to the ECEF system.

4.1.1 Speed from GPS coordinates

In order to decrease the number of data types needed to extract using the network sniffer, hence increasing robustness, the speed, v, will be calculated using the position data. The speed calculation makes use of the time stamps, t, attached to the position data, and is performed as:

$$v = \left| \frac{\bar{x}_2 - \bar{x}_1}{t_2 - t_1} \right| \tag{9}$$

where the positions $\bar{x}_i$ must be given in Cartesian coordinates in the ECEF system, that is, $\bar{x} = [X,\ Y,\ Z]$. The conversion between GPS coordinates and ECEF coordinates can be done by using the following equations [3]:

$$X = (N + h)\cos\varphi\cos\lambda \tag{10a}$$

$$Y = (N + h)\cos\varphi\sin\lambda \tag{10b}$$

$$Z = [(1 - e^2)N + h]\sin\varphi \tag{10c}$$

where λ, φ, and h are the geodetic coordinates, that is, longitude, latitude, and altitude over the ellipsoid, respectively. Furthermore, e is the eccentricity of the ellipsoid, and N is the radius of curvature in the prime vertical, which is given by

$$N = \frac{a}{1 - e^2\sin^2\varphi} \tag{11}$$

where a is the semi-major axis of the ellipsoid. For WGS84, the values of the eccentricity and semi-major axis are [17]:

$$e = 8.1819190842622 \cdot 10^{-2}$$
$$a = 6378137\ \text{m}$$

4.2 Rotational kinematics

In CesiumJS, the rotation of a body is represented by quaternions in the ECEF system [24]. The data that can be extracted from the gyros on the rocket, however, is defined with respect to an inertial frame, which will coincide with the TH system, which has its origin on the launchpad. Hence, to find the rotations of the rocket with respect to the ECEF system, it is necessary to find the rotation of the body axis with respect to the TH system, and subsequently the rotation of the TH system with respect to the ECEF system. This can be done by finding the direction cosine matrix, or DCM, of the two rotations, and multiply them so that the resulting DCM gives the transformation from the body axis to the ECEF system. The order of operations done to construct this transformation matrix will in this project be done like the following, where $\{X\}$ denotes the coordinate system X, and $\left[C_X^Y\right]$ denotes the matrix that describes the rotation of the $\{X\}$ system with respect to the $\{Y\}$ system.

$$\{B\} = \left[C^{B}_{TH}\right]\{TH\} \tag{12}$$
$$\{TH\} = \left[C^{TH}_{ECEF}\right]\{ECEF\} \Rightarrow \tag{13}$$
$$\{B\} = \left[C^{B}_{TH}\right]\left[C^{TH}_{ECEF}\right]\{ECEF\} \tag{14}$$

The attitude data received from the rocket is given with respect to the TH system, and is most commonly represented by quaternions. The DCM $\left[C^{B}_{TH}\right]$ can be, according to [12], constructed from these quaternions as:

$$\left[C^{B}_{TH}\right] = \begin{bmatrix} \beta_0^2+\beta_1^2-\beta_2^2-\beta_3^2 & 2(\beta_1\beta_2+\beta_0\beta_3) & 2(\beta_1\beta_3-\beta_0\beta_2) \\ 2(\beta_1\beta_2-\beta_0\beta_3) & \beta_0^2-\beta_1^2+\beta_2^2-\beta_3^2 & 2(\beta_2\beta_3+\beta_0\beta_1) \\ 2(\beta_1\beta_3+\beta_0\beta_2) & 2(\beta_2\beta_3-\beta_0\beta_1) & \beta_0^2-\beta_1^2-\beta_2^2+\beta_3^2 \end{bmatrix} \tag{15}$$

In some cases, however, the attitude data could be given as Euler angles, which are defined in a yaw-pitch-roll, or (3-2-1), configuration. the DCM $\left[C^{B}_{TH}\right]$ can in this case be constructed by multiplying the three DCM's corresponding to a rotation around each of the body axis. Here, roll is defined around the body x-axis, pitch around the y-axis, and yaw around the z-axis.

$$\text{Roll}: \quad C_1(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \tag{16a}$$
$$\text{Pitch}: \quad C_2(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{16b}$$
$$\text{Yaw}: \quad C_3(\psi) = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{16c}$$

By using the (3-2-1) sequence, the complete DCM from the TH system, to the body axis system can be described by: [12].

$$\left[C^{B}_{TH}\right] = C_1(\phi)C_2(\theta)C_3(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{17}$$

which gives the complete DCM as:

$$\left[C^{B}_{TH}\right] = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi-\cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi+\cos\phi\cos\psi & \sin\phi\sin\theta \\ \cos\phi\sin\theta\cos\psi+\sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi-\sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix} \tag{18}$$

Now, to get the quaternions with respect to the ECEF system, it is necessary to find the DCM that describes the rotation of the Topocentric-Horizon system with respect to the

ECEF system. By analyzing figure 9, it could be determined that this can be done by first rotating an amount equivalent to the longitude around the Z-axis, and subsequently an amount equivalent to the latitude around the negative Y-axis. This corresponds to a (3-2) rotation, and the DCM can be constructed by multiplying matrix 16b with matrix 16c:

$$\left[C^{TH}_{ECEF}\right] = \left[C_2(-\varphi)\right]\left[C_3(\lambda)\right] = \begin{bmatrix} \cos\lambda\cos\varphi & \sin\lambda\cos\varphi & \sin\varphi \\ -\sin\lambda & \cos\lambda & 0 \\ -\cos\lambda\sin\varphi & -\sin\lambda\sin\varphi & \cos\varphi \end{bmatrix} \tag{19}$$

Finally, to provide CesiumJS with the quaternions, it is necessary to extract them from the DCM describing the rotation of the rocket with respect to the ECEF system. The simplest way of doing so is to directly identify terms in matrix (15):

$$\beta_0 = \pm\frac{1}{2}\sqrt{C_{11} + C_{22} + C_{33} + 1} \tag{20a}$$

$$\beta_1 = \frac{C_{23} - C_{32}}{4\beta_0} \tag{20b}$$

$$\beta_2 = \frac{C_{31} - C_{31}}{4\beta_0} \tag{20c}$$

$$\beta_3 = \frac{C_{12} - C_{21}}{4\beta_0} \tag{20d}$$

This method, however, gives rise to a mathematical risk. If β_0 reaches zero, the equations (20b)-(20d) breaks down. In order to avoid this, a method described by Markley [15], which is a modification of Sheppard's Algorithm [19], can be used. Sheppards Algortihm is based on the assumption that the DCM used is orthogonal, and could possibly break down if numerical errors are introduced, whereas the method developed by Markley works even if the DCM is only approximately orthogonal. The method works by first constructing the four quaternions in four different ways:

$$\boldsymbol{q}^{(1)} = \begin{bmatrix} 1 + trace(C) \\ C_{23} - C32 \\ C_{31} - C13 \\ C_{12} - C_{21} \end{bmatrix} \tag{21a}$$

$$\boldsymbol{q}^{(2)} = \begin{bmatrix} C_{23} - C_{32} \\ 1 + 2C_{11} - trace(C) \\ C_{12} + C_{21} \\ C_{13} + C_{31} \end{bmatrix} \tag{21b}$$

$$\boldsymbol{q}^{(3)} = \begin{bmatrix} C_{31} - C_{13} \\ C_{12} + C_{21} \\ 1 + 2C_{22} - trace(C) \\ C_{23} + C_{32} \end{bmatrix} \tag{21c}$$

$$\boldsymbol{q}^{(4)} = \begin{bmatrix} C_{12} - C_{21} \\ C_{31} + C_{13} \\ C_{23} + C_{32} \\ 1 + 2C_{33} - trace(C) \end{bmatrix} \tag{21d}$$

By comparing terms with matrix (15), it can be seen that equations (21a)-(21d) corresponds to

$$\boldsymbol{q}^{(i)} = 4\beta_i\boldsymbol{\beta} \tag{22}$$

which means that all vectors $\boldsymbol{q}^{(i)}$ are just linear combinations of $\boldsymbol{\beta}$, and so $\boldsymbol{\beta}$ can be calculated as

$$\boldsymbol{\beta} = \frac{\boldsymbol{q}^{(i)}}{||\boldsymbol{q}^{(i)}||} \tag{23}$$

Furthermore, by choosing the vector $\boldsymbol{q}^{(i)}$ of greatest magnitude, it can be guaranteed that equation (23) never breaks down.

In the case of complete absence of attitude data, an attitude aligned with the direction of flight will be calculated instead. If no rotation has occurred, the rockets x-axis $\boldsymbol{x}_b$ will be aligned with the x-axis of the TH system, $\boldsymbol{x}_{TH}$. Now, assume that the rocket should be aligned with a vector $\boldsymbol{v}$ instead. To achieve that attitude, a rotation from $\boldsymbol{x}_{TH}$ to $\boldsymbol{v}$ has to be performed. The principal rotation axis $\boldsymbol{e}$ of such a rotation, which is the axis around which a single rotation can be performed to achieve the desired attitude [12], is the axis that is perpendicular to both $\boldsymbol{x}_{TH}$ and $\boldsymbol{v}$, and is illustrated in figure 10. A quaternion describing a rotation Φ around the principal rotation axis can now be constructed as [12]:

$$\beta_0 = \cos\frac{\Phi}{2} \tag{24a}$$
$$\beta_1 = e_1\sin\frac{\Phi}{2} \tag{24b}$$
$$\beta_2 = e_2\sin\frac{\Phi}{2} \tag{24c}$$
$$\beta_3 = e_3\sin\frac{\Phi}{2} \tag{24d}$$

where e_1, e_2, and e_3 are the three components of the principal rotation vector. Since a dot product and cross product between two normalized vectors are defined as

$$\boldsymbol{x}\cdot\boldsymbol{y} = \cos\alpha \tag{25}$$
$$\boldsymbol{x}\times\boldsymbol{y} = \perp(\boldsymbol{x},\boldsymbol{y})\sin\alpha \tag{26}$$

where α is the angle between the two vectors, equation 24a-24d can instead be expressed as the dot product and crossproduct between a vector $\boldsymbol{u}$, defined by

$$\boldsymbol{u} = \frac{\boldsymbol{x}_{TH}+\boldsymbol{v}}{||\boldsymbol{x}_{TH}+\boldsymbol{v}||} \tag{27}$$

and $\boldsymbol{v}$, as stated below:

$$\beta_0 = \boldsymbol{u}\cdot\boldsymbol{v} \tag{28a}$$
$$\boldsymbol{\beta}_{1,2,3} = \boldsymbol{u}\times\boldsymbol{v} \tag{28b}$$

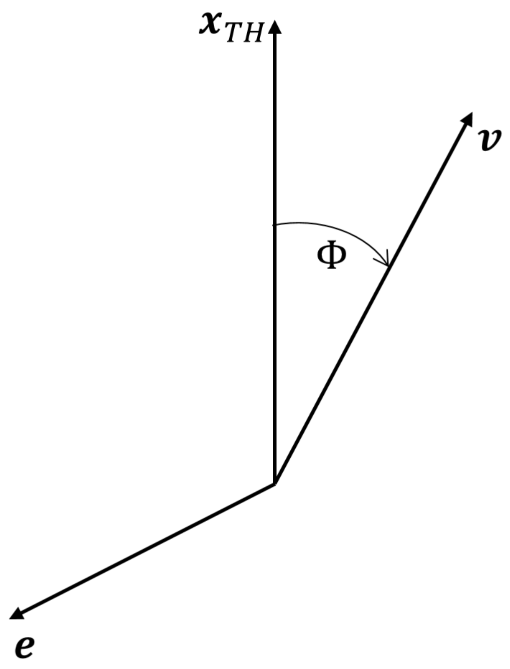


Figure 10: Illustration of the rotation from one vector to another around the principal rotation vector.

4.3 Data filtering

The data extracted from a sensor is never perfect. Even the most sophisticated sensors are subject to noise and uncertainties. However, these undesired traits can potentially be mitigated using different kinds of filters. When using filter for real-time applications, however, some constraint arise with regards to computational efficiency and data latency. For that reason, the data should not be filtered more than necessary. In the following sections, two very common filters are discussed.

4.3.1 Moving average filter

A moving average filter is a very popular filter type used to mitigate the effects of random noise [20]. It works by averaging a number of values in the data set, as according to:

$$y(k) = \frac{x(k) + x(k-1) + x(k-2) + \ldots + x(k-N-1)}{N} \tag{29}$$

where k is the current data point, and N the averaging window. To demonstrate the effects of a moving average filter, a sine curve with added white Gaussian noise with a signal-to-noise ratio of 30 was created, as shown in figure 11.

A moving average filter with an averaging window of 20 samples was then applied to the data set, giving a smoothing effect. Figure 12) shows the comparison between the original

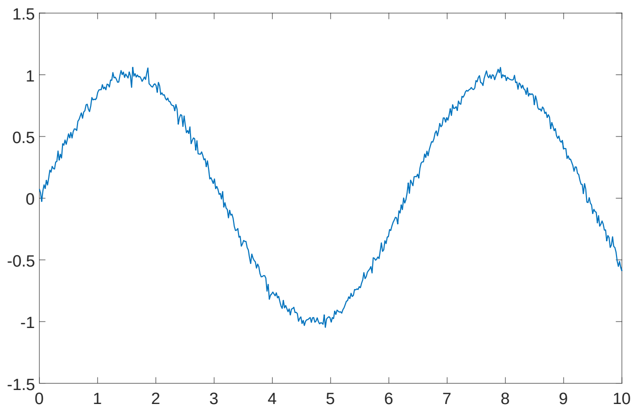


Figure 11: Sine signal with added noise

signal, before adding noise, and the filtered signal. It is evident that the moving average filter does remove the noise to a certain degree, however, shown here is also the signal delay that this kind of filter creates. This delay is caused by the fact that each data point has a corresponding time point, and when combining the data points into one average, the same has to be done with the time. Thus, the time delay caused by an average filter is given by

$$\Delta t = \frac{(N-1)T}{2} \tag{30}$$

where T is the time between each sample.

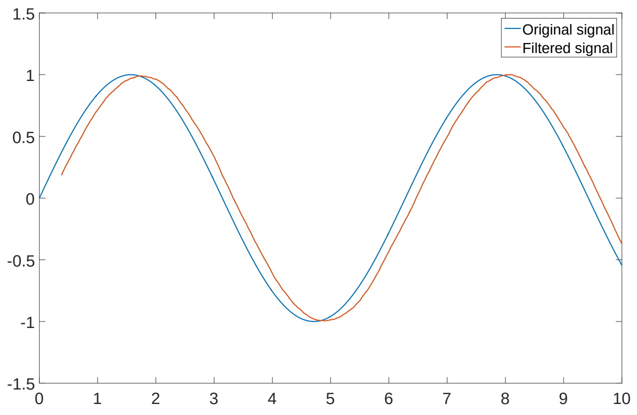


Figure 12: Original signal, before adding noise, in comparison to the signal containing white Gaussian noise that is filtered using a moving average filter.

4.3.2 Median filter

Some sensors can experience impulse noise in the data output, which more commonly is known as data spikes. A simple filter that is frequently used for this kind of behavior is the median filter. The advantage of using a median filter for impulse noise is that the median of a set of numbers is a non-linear statistics of of the set, with the property that it is insensitive to to possible outliers within the set [23]. The median filter samples a number of data points, and then chooses the median of those points. In case of the sampling window being an even number of points, the median can instead chosen as the mean of the two points in the middle of the sorted data set. The filter can be mathematically described as

$$y(k) = \text{median}\{x(k),\ x(k-1),\ x(k-2),\ \dots\ x(k-N-1)\}, \quad N \text{ odd} \tag{31}$$

$$y(k) = \text{mean}\{x_{sorted}(k-N/2),\ x_{sorted}(k-N/2-1)\}, \quad N \text{ even} \tag{32}$$

In figure 13, the same sinusoidal signal as used for the moving mean filter is showed, but with a random distribution of spikes added instead of white Gaussian noise. In figure 14, a median filter of window size 3, which is the minimum reasonable size for a median filter, was applied. Almost all of the data spikes was removed from the signal, leaving only those where the impulse noise lasted for two samples or more.

The downside of using median filters are that they can not remove spikes with a length of more than half the window size. Furthermore, increasing the window size increases both the computational complexity, as well as the risk of losing some of the characteristics of the data.

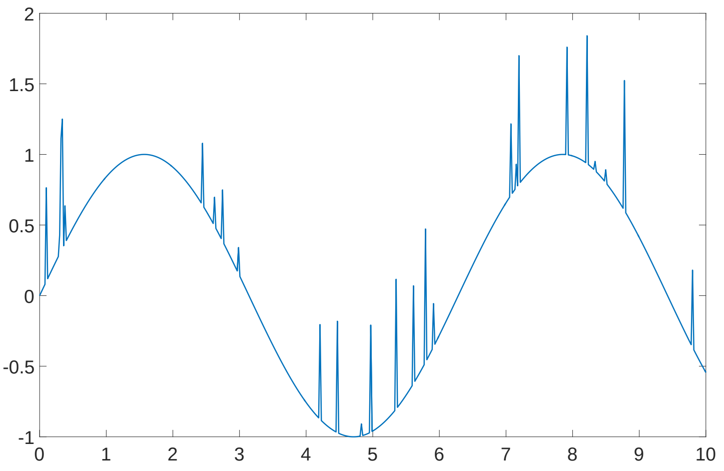


Figure 13: Sine signal with added noise

The temporal displacement of the median filter is limited to the time range of the filter window. That is, the maximum time delay that can occur for a sample is:

$$\Delta t = (N-1)T \tag{33}$$

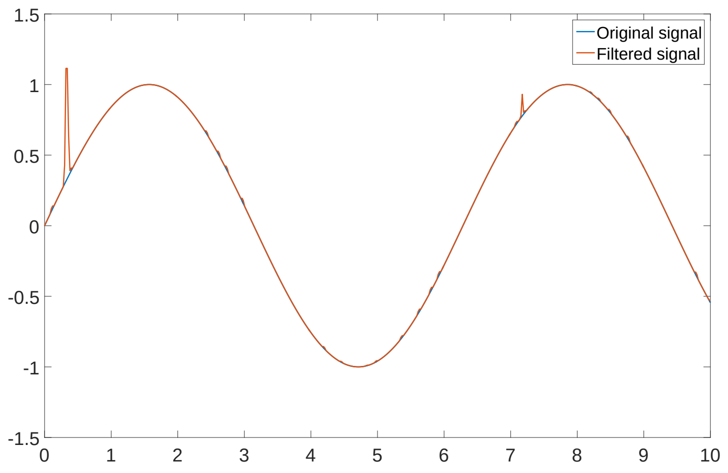


Figure 14: Sine signal with added noise

However, since the window size of a median filter is preferably kept to a minimum, the temporal displacement of a median filter can often be disregarded.

5 Testing

5.1 Pre-flight test cases

Since no live data was available, a number of test cases were designed in order to simulate some of the behavior that could occur during a real launch. The tests were set up along with conditions that had to be verified before the test could be considered successful. The data used during these tests was predicted flight data, as well as some network data that was collected during the stationary tests of the MAXUS 9 rocket.

Test case 1 - The full set of data is being updated:

Position, attitude, G-loads, and angular rates are all being updated. In this case, the visualization should be fully operational. In the main window as well as in the attitude view, the 3D model shall update its position and attitude. Furthermore, all numerical displays and their respective visualization elements shall be updated.

Test case 2 - Full data set $\rightarrow$ GPS signal lost $\rightarrow$ GPS signal recovered:

If the GPS signal is suddenly lost, the data processor shall keep sending the last known position, although with a new time stamp, constructed using equation 1. During this time, the 3D-model shall not move, the ground track and altitude curve shall not be updated, the speed shall retain its last value, and the GPS status diode shall turn red. If the signal then is recovered, the diode shall turn green, and the visualization components shall resume the updating of data.

Test case 3 - Full data set except attitude data available:

In the complete absence of attitude data, the attitude shall follow the direction of movement, as described in section 4.2.

Test case 4 - Full data set $\rightarrow$ Attitude data lost $\rightarrow$ Attitude data recovered:

If the attitude data is not being updated, but old attitude data exists, the data processor shall push the latest known data point instead of falling back to the scenario stated in test case 3. If the attitude would fall back to following the direction of movement, it would risk the attitude switch back and forth between the two modes, which is why it was deemed better to maintain the last received attitude.

Test case 5 - Full data set $\rightarrow$ G-load data lost $\rightarrow$ G-load data recovered:

In the case of not receiving new G-load data, the data processor will keep pushing the old data point for a predetermined time. If the time since last new data point was received exceeds that, the visualizer shall place an overlay on top of the G-load component, stating that no new G-load data is available.

Test case 6 - Full data set → Angular rate data lost → Angular rate data recovered:

The handling of this test case is identical to test case 5, except it will be with respect to the angular rate data.

Test case 7 - No data is being updated → each data set added separately:

This test case test is mainly in place in order to test the status diodes. When no data is being updated, the data processor shall send empty data packets. The visualizer shall detect these data packets, and state that a connection is established between the data processor and the visualizer by setting the SYNC diode to green. If any of the data sets are updated, the TM diode shall turn green, and if the GPS signal is detected, the GPS diode shall turn green.

Once all of these test cases could be run without any remarks, requirement O.3, O.4, and O.5 could be considered to be verified,

5.2 Live flight test

The product was tested during the launch of the MAXUS 9 rocket, which was a sounding rocket designed for research in micro-gravity, and was developed through the combined effort of the Swedish Space Corporation and Airbus [6]. The payload of MAXUS 9 consisted of five different experiment modules, containing a total of 10 separate experiments, as well as the various rocket service systems, such as the guiding and communication system. The engine was of the type Castor IVB, which uses about 10 tonnes of solid fuel to propel the rocket into space. After burn-out, the rocket engine was separated from the payload, and after about 100 seconds from launch, micro-gravity was achieved and was then maintained for about 12 minutes. Finally, the rocket landed using parachutes to bring the speed down, and the retrieved by helicopter.

During the launch, data was collected and visualized in real time, and network data was recorded. The visual performance of each of the programs components were evaluated for later improvements, and the comments are shown below:

Main 3D window

Worked mostly correct, the trajectory looked good, which meant that requirement F.8 could be considered verified, and the data handling over transition between negative to positive time worked perfectly. However, the angle of the visualization view sometimes glitched, as shown in figure 16. Furthermore, there was a slight time shift between the countdown clock and the simulated time, which made it look like the rocket did not launch precisely when the countdown hit zero.

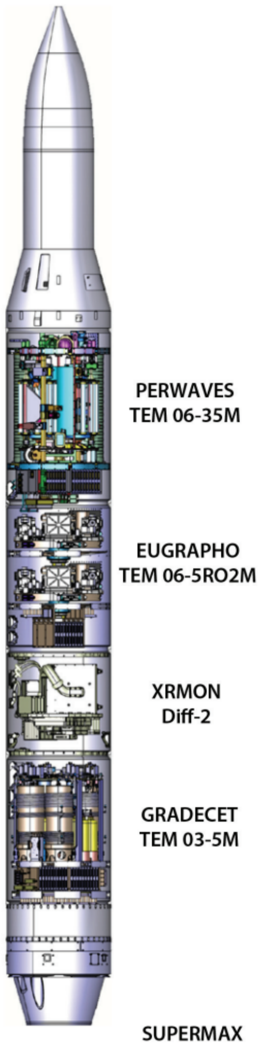


Figure 15: Illustration of the MAXUS 9 payload and placement of the different experiments.

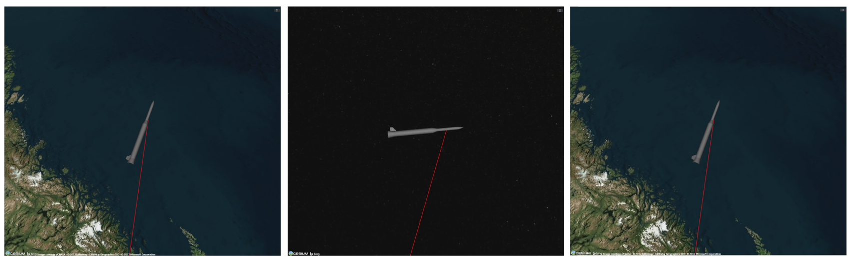


Figure 16: Visualization program showed over a time span of three frames showing the camera glitch that sometimes occured during the MAXUS 9 flight. In the first frame, the camera was in its nominal position. In the second frame it flipped to show the rocket from underneath, only to change back the subsequent frame.

Attitude
The quaternions received were visualized correctly, however, the attitude data were riddled with spikes and strange behaviors, resulting in a large amount of glitching, as show in figure 17. Furthermore, in the middle of the flight, the attitude could suddenly change and then be maintained in that new state, giving rise to an uncertainty about whether the attitude visualized was correct or not.

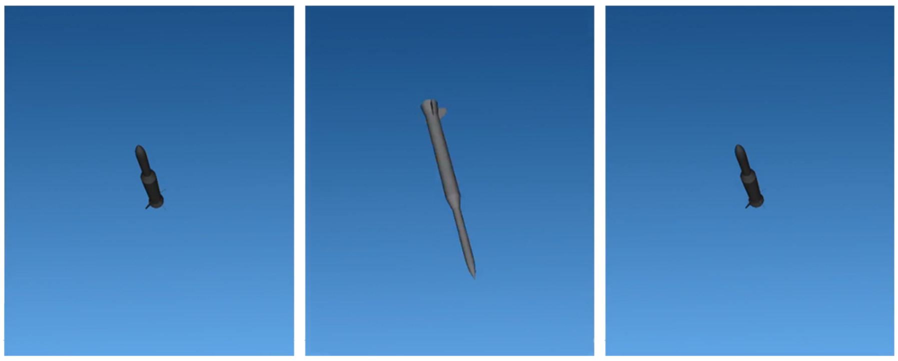


Figure 17: A sequence of three frames, showing the attitude glitch. In the first frame, the nominal attitude is visualized. The second frame shows how the rocket abruptly changed attitude, and in the third frame, it was back to normal.

Ground track
The ground track component worked as intended, visualizing both the ground track as a line and the IIP, which meant that requirement F.9 and F.10 could be considered verified.

Altitude curve
The altitude curve component worked as intended, and handled the loss of GPS signal in a good way. The only remark was that the unit for the altitude should be changed from meters to kilometers.

Speed gauge
The speed gauge visualized the speed that was received as intended, however, the value of the speed was incorrectly calculated, giving rise to a glitching behavior, with values jumping from very large to very small values in an unreasonable way, as shown in figure 18.

G-loads
The G-Loads component worked as intended.

Angular rates
The angular rate data was handles as intended, however, the data received was not scaled to the unit showed in the product. The problem was noticeable when comparing the rotation of the rocket model and the values displayed in this component.

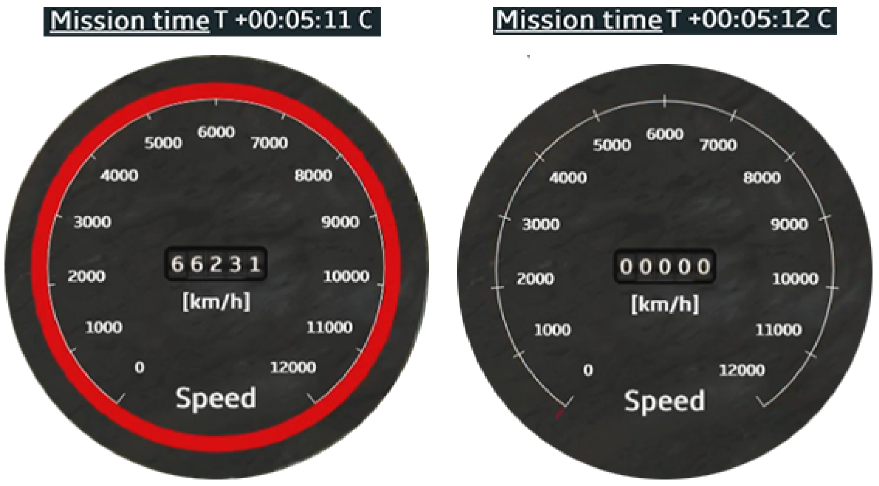


Figure 18: Two samples of the speed gauge with a time difference of one second. In the left sample, the speed calculated from the GPS data was 66231 km/h, which in itself is unreasonable, and one second later it was calculated to 0 km/h.

Status diodes
The status diodes worked as intended.

Event display
The mission events were displayed as intended, and requirement F.7 could be noted as verified.

Mission time
The mission time was updated and displayed as intended, for which reason requirement F.6 could be noted as verified.

5.2.1 Flight data analysis

The network traffic was recorded during the launch of the MAXUS 9 rocket, enabling future playbacks to be made in order to simulate a live launch at will. The server was set up to record and save the data as .csv files, enabling the possibility of thoroughly analyze the data improve performance.

Position
Since the visual performance of the components using the position data was fairly good, it was expected that no drastic improvements had to be made on that particular data set. Looking at figure 19, it can be seen that the data makes out a smooth curve, making out the trajectory of the rocket. The spiral behavior at the end of the curve is the result of the rocket parachuting down, making it drift along with the wind. Since the behavior corresponded to what was expected, requirement F.1 could be marked as verified

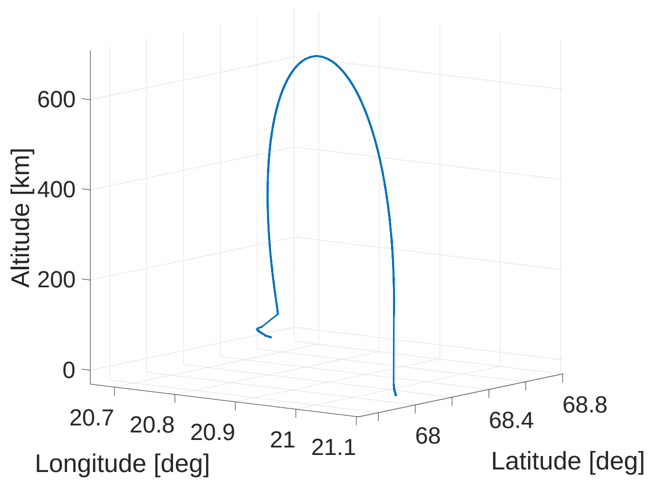


Figure 19: The flight path of the MAXUS 9 rocket.

The spiraling movement is also visible when plotting the latitude versus longitude, as show in figure 20, which makes up the ground track of the rocket.

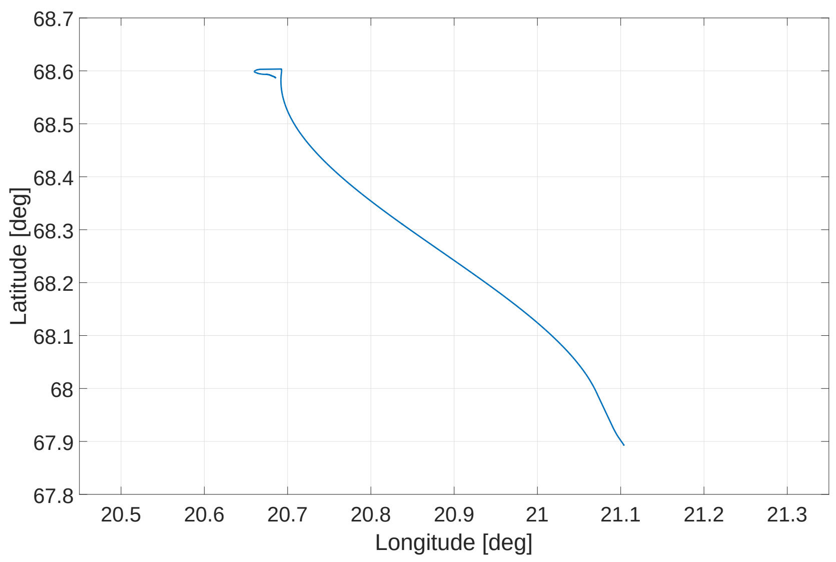


Figure 20: The ground track of the MAXUS 9 rocket.

The GPS signal was not always perfect, and longer times where no GPS signal was received occurred at two points, first during the engine burn, and then during the reentry. This loss of signal was expected to show up in the position plots, however, in figure 19 and 20, it is not really noticeable. The reason for that is that the distances are so large, and at the points of signal loss the horizontal velocity was small in comparison to the vertical velocity. If, instead, figure 21, 22, and 23, which shows the time dependence of the three position properties of the rocket, are studied, it is obvious that there in fact was at least two longer periods of GPS signal loss. The signal loss was handled by maintaining the last known position of the rocket, as intended.

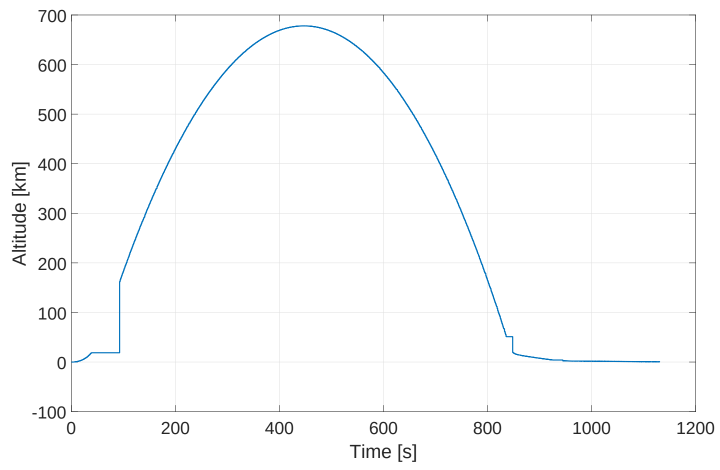


Figure 21: Altitude curve for the MAXUS 9 rocket.

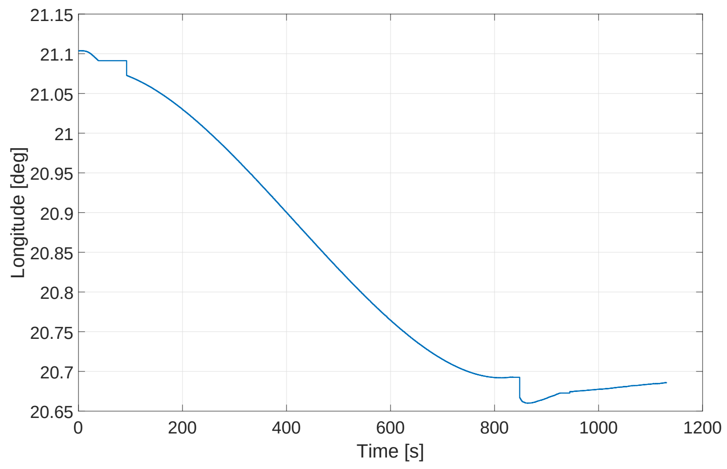


Figure 22: Longitude plotted against time for the MAXUS 9 rocket.

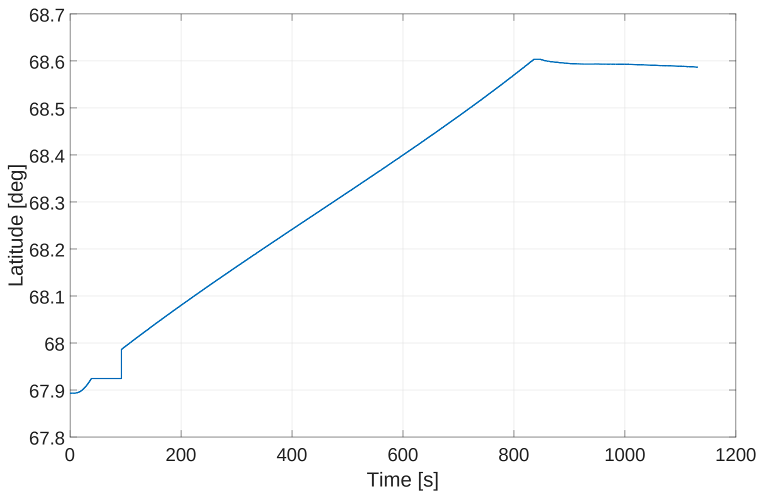


Figure 23: Latitude plotted against time for the MAXUS 9 rocket.

Attitude

The attitude visualization showed many unwanted properties, including glitching and ambiguous behavior. The fact that the attitude could suddenly change was tracked down to a sign error in how matrix 15 was constructed from the quaternions. The first element in the matrix was calculated as

$$\beta_0^2 + \beta_1^2 + \beta_2^2 + \beta_3^2 \tag{34}$$

instead of

$$\beta_0^2 + \beta_1^2 - \beta_2^2 - \beta_3^2 \tag{35}$$

as it should. This in itself would not be a big problem, and the attitude should at least have shown consistent behavior, however, since the matrix was no longer a real rotation matrix, the Markley algorithm broke down as the vectors 21a - 21d no longer were linear combinations of equation 23. This, in turn, meant that the selection process where the quaternion was constructed from the vector $\mathbf{q^{(i)}}$ of greatest magnitude resulted in different quaternions depending on which $\mathbf{q^{(i)}}$ was chosen. The reason why this issue did not show up in any of the tests being done before is that the test data consisted of predicted euler angles, which means that this part of the code was not used. Furthermore, the test network data that was obtained only contained quaternions corresponding to the rocket being aligned with the first axis of the coordinate system, which implies that $\beta_2 = \beta_3 = 0$, resulting in that equation 34 being equal to equation 35.

Once it was ensured that the quaternions were constructed in a reliable way, the data could be plotted and evaluated. In figure 24, the quaternions describing the rockets attitude are plotted versus time, and figure 25 shows the corresponding Euler angles. It is evident that the data was experiencing heavy impulse noise, which is the reason for the glitching behavior in the attitude visualization.

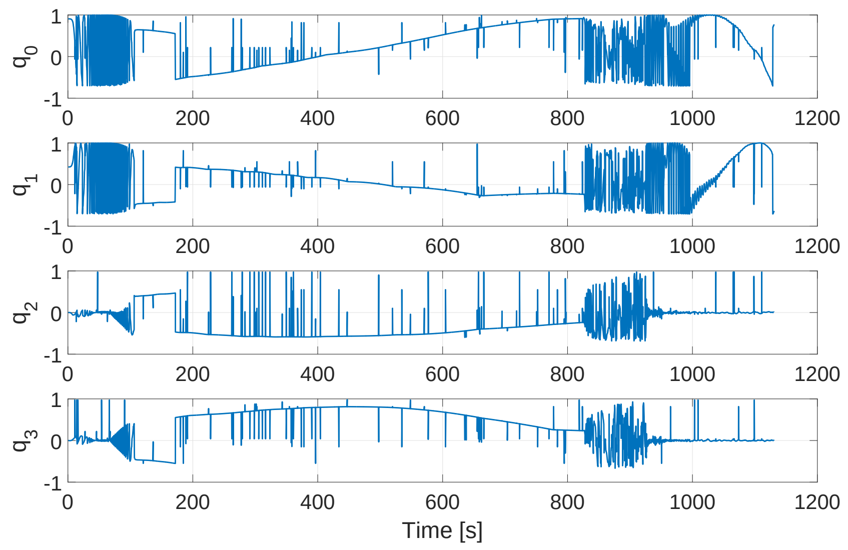


Figure 24: The quaternions received during the MAXUS 9 flight.

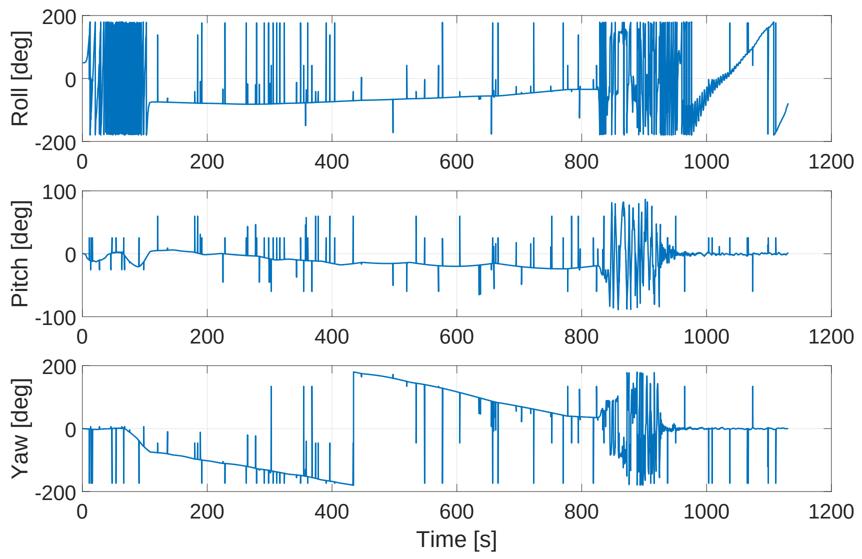


Figure 25: The Euler angles calculated from the quaternions.

G-loads

The G-loads were visualized without any remarks, for which reason requirement F.4 could be considered verified. Figure 26 shows the time development of the G-loads over time, and it can be noted that no extreme noise was present. Initially, the x-component, which is defined along the rocket, increased gradually to about 8.4 G, only to quickly drop at about 64 s, which was when the rocket reached burn-out. After about 100 s, micro-gravity was reached and maintained for about 10 min. When re-entering the atmosphere, it seems like the rocket experienced a chock of about 23 G, as seen in figure 27. However, by zooming in on the time of re-entry, as done in figure 28, it can actually be seen that the accelerometers in the y and z direction are maxing out at about 16 G, which implies that the load on the rocket was even higher.

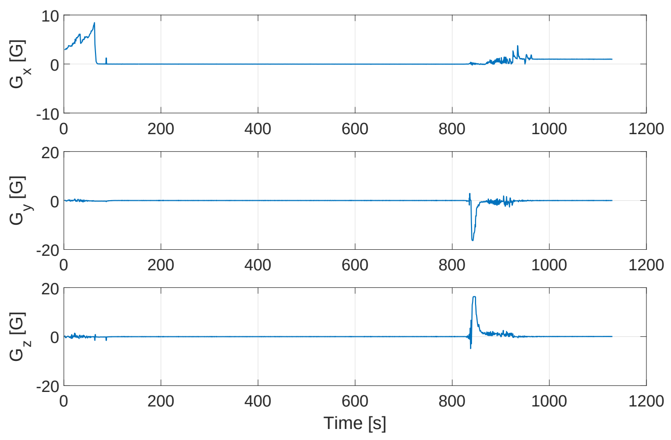


Figure 26: The G-loads experienced by the rocket during its flight.

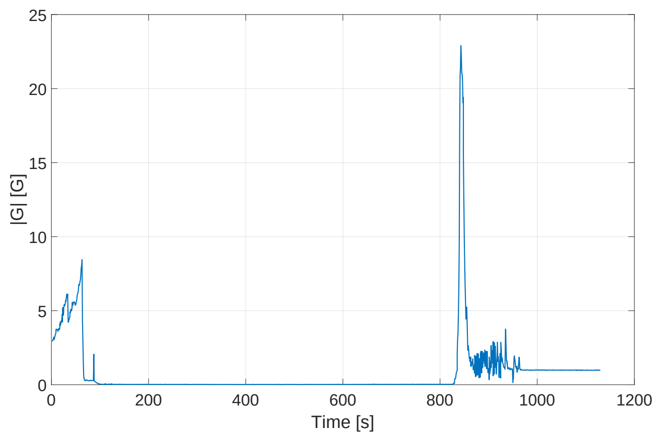


Figure 27: The absolute value of the G-loads experienced by the rocket during its flight.

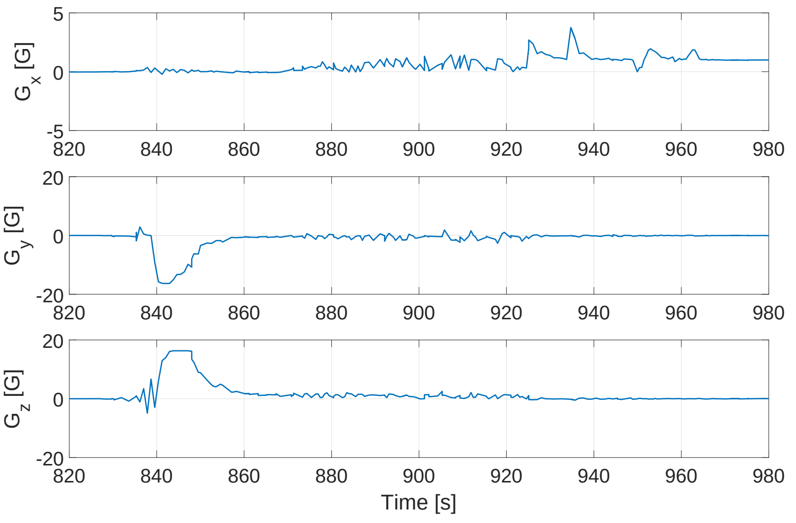


Figure 28: G-loads during re-entry.

Angular rates

Once the scaling of the angular rates was correct, they were plotted versus time. During the burn, the rocket started to spin up, and was then stabilized during the micro-gravity phase. It was then stable until right before re-entering the atmosphere, where it started to spin up again. It started to tumble during its fall through the atmosphere, only to oscillate to a stop during the parachuting phase. From figure 29, it is evident that the data set was subject to a great deal of noise, often in the form of spikes with constant magnitude.

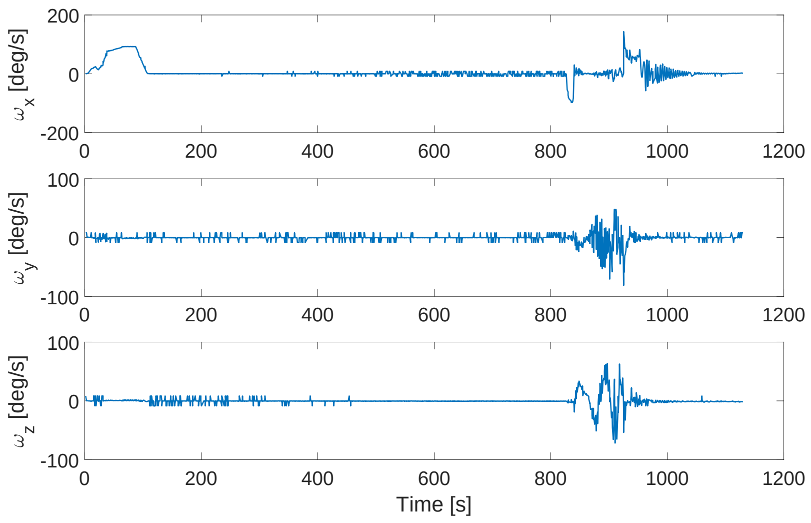


Figure 29: The Angular velocity experienced by the rocket during its flight.

Speed
The visualization of the speed calculated from the GPS data was very jittery, and looking at figure 30, it is obvious that something went very wrong. The data experienced data spikes of several order of magnitudes greater compared to what was expected.

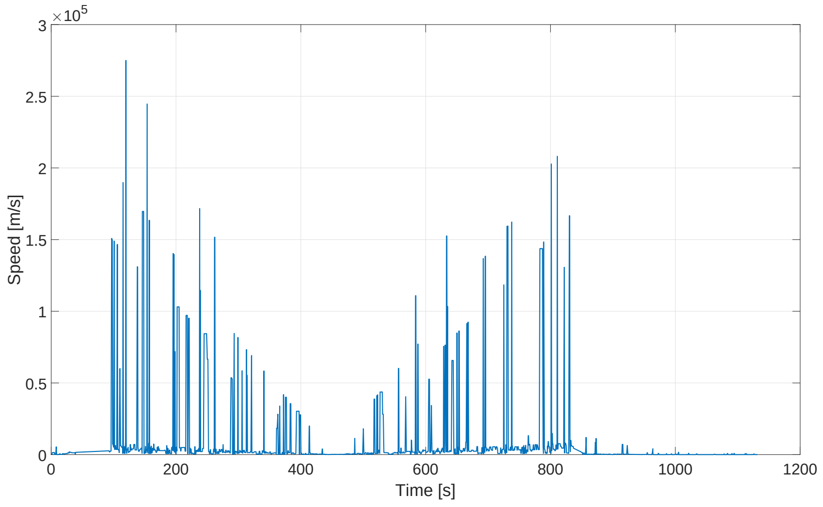


Figure 30: Speed values calculated during the MAXUS 9 flight.

5.2.2 Improvements

Looking at the data plots in section 5.2.1, it is evident that some data sets were in need of some kind of improvement. The attitude data suffered from heavy impulse noise, as seen in figure 24 and 25, with most of the spikes having a width of a single sample. As discussed in section 4.3.2, a median filter does a good job of removing such spikes, and for that reason a median filter with a window size of three samples was implemented on the attitude data set. The resulting output can be seen in figure 31 and 32. With the median filter implemented, almost all of the data spikes were gone, giving a smooth attitude description. One of the spikes, which is visible at about 500 s had a sample length of two, which made the filter ineffective at that particular spike. It would be possible to remove such spikes by increasing the filter window size to five samples, however, heavier filtering increases the risk of removing real behavior from the data, and for that reason the filter window size was kept at three samples. The requirement corresponding to the visualization of the attitude data, F.2, could thereby be seen as verified.

The gyro data also experienced frequent impulse noise, as seen in figure 29, which implies that the use of a median filter could result in some improvement. Figure 33 shows the output obtained using a median filter with a filter size of three samples. The result is a signal which still experience noise, but to a degree that was deemed satisfactory, and so requirement F.3 could now be considered verified. A heavier filter would be able to remove the remaining spikes, but that would risk losing oscillating behaviors.

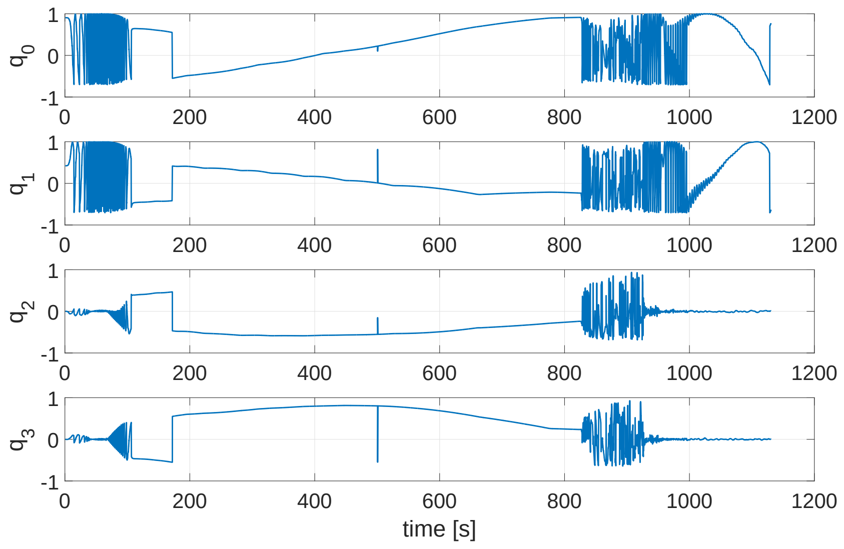


Figure 31: The quaternions received during the MAXUS 9 flight, filtered using a median filter.

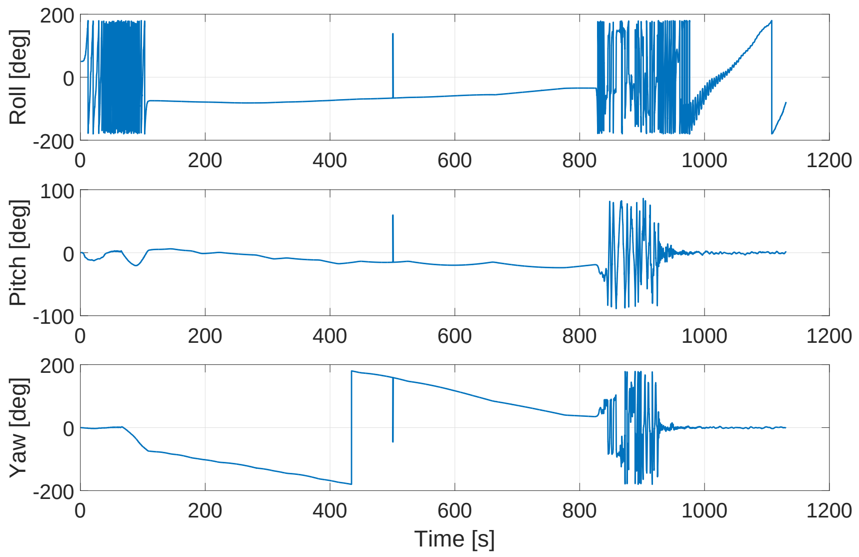


Figure 32: The euler angles received during the MAXUS 9 flight, filtered using a median filter.

At a first glance of the speed data in figure 30, it seems like the data was subject to heavy impulse noise, however, since the data set was calculated using the GPS data, which did not have have any problems with noise, it was suspected that the error was in the calculation

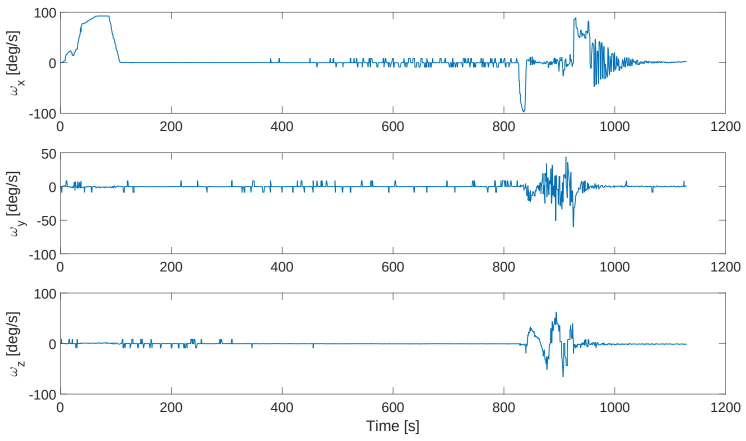


Figure 33: The Angular velocities received during the MAXUS 9 flight, filtered using a median filter.

itself. A heavy median filter was applied to the data set in order to find out if at least the base data was correct, or if even the behavior was wrong. In figure 34, a median filter with a window size of 17 samples was applied, showing a slightly more reasonable behavior, although the data shows a very high variation that seems to be centered around some mean line. A heavy moving average filter, as described in section 4.3.1, was applied in order to further investigate the underlying data, and the result were then plotted in figure 35. The result shows that the underlying data is most likely correct, as it shows a behavior that was expected, that is, the speed was increasing during the burn, only to fall approximately linearly to the point in time which corresponds to the apogee.

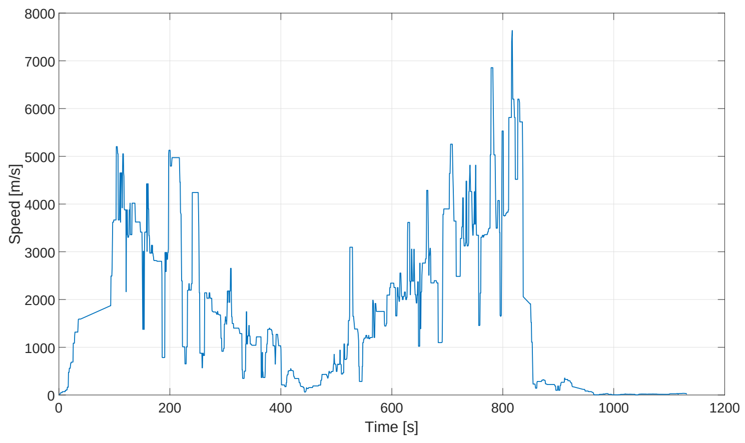


Figure 34: Speed values using a median filter with a filter window of 17 samples.

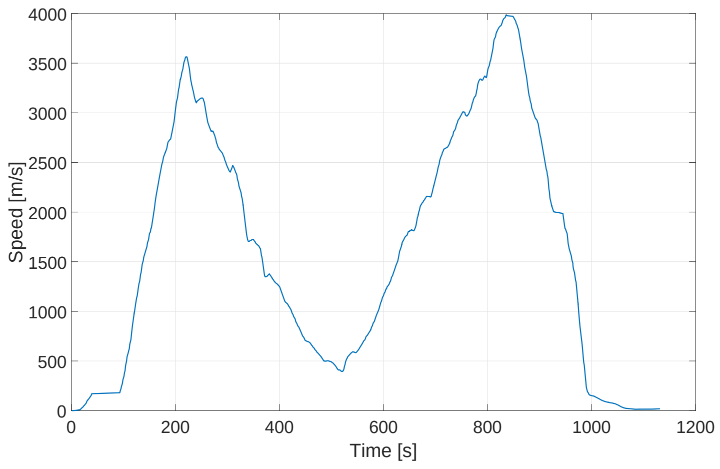


Figure 35: Speed values using a moving average filter with a filter window of 200 samples on top of the median filter.

Since the underlying data showed a feasible behavior, the method of calculating the speed was most likely correct, which implied that the errors were caused by the data used in the calculation having large uncertainties. The issue was finally tracked down to the method of time stamping the data. The speed calculation used the time stamps defined in system time as the data point was detected. This meant that, if the time it took from the rocket generating the data point to the time of detecting that point with the network sniffer was not consistent, the data used in the numerator in equation 9 would not correspond to the data used in its denominator. This was reworked to use the real time stamps that are generated as the same time as the positions. Figure 36 shows the resulting behavior, which is a smooth line corresponding to the expected output. At this point, both requirement F.5 and O.2 could be noted as verified.

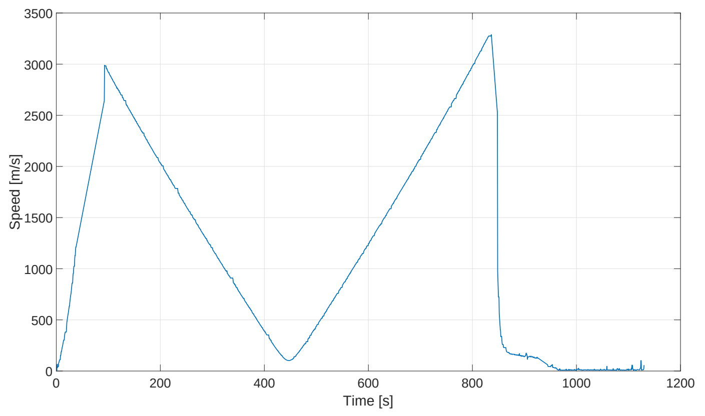


Figure 36: Speed values calculated using the correct time stamps.

6 Results and Discussion

During the course of this project, a product aiming to visualize flight data during suborbital missions has been developed. The resulting product consists of three main parts, a data processor, a server, and a visualizer. The data processor works by extracting the raw flight data from a local network using a network sniffer. It then processes it before sending it to the server in the JSON-based format CZML. The server, in turn, sorts the data, and stores necessary backlogs while sending the necessary data for visualization to the visualizer. The Visualizer consists of a browser application that makes use of the JavaScript library Cesium, which is specialized in the visualization of dynamic geospatial data.

The trade-offs and choices made in order to arrive at the resulting program have all been made with the goal of creating a product in line with the requirements stated by SSC. A continuous stream of feedback has been received, which in combination with the discussions held with the employees at SSC have helped to make the product as good as possible.

A first prototype of the product was first tested live during the launch of the MAXUS 9 rocket, which occurred on April 7, 2017. The performance of the product during this test was imperfect, but it was possible to collect a large amount of valuable feedback and data, which served as a base for heavy improvements. Furthermore, during this test, the network data was recorded, making it possible to simulate a real-time launch at will, which was a great help during the continuing development.

The resulting product satisfies the requirements stated in section 2. It visualizes the position, attitude, speed, angular rates, accelerometer, and gyro data. Furthermore, it displays he mission time and events that occur during the flight. The performance of the product was also satisfactory, and only differed a couple of seconds from true real-time, which verifies requirement P.2. The visualizer, however, is quite demanding regarding processor power, which makes the experience on low-end devices limited.

6.1 Societal impact

The space industry is currently in a state of change, as more and more private companies are trying to establish their presence. With this change, the industry is becoming more open than ever, giving the public an insight in the current development of new technologies that has not been possible before. The introduction of real-time visualization applications, such as the one developed during this project, will hopefully help to further increase the public interest in space technology, which will hopefully inspire more people and businesses to get involved within the space industry.

Numerous technologies that initially were developed for the space industry has been adopted for applications on Earth. These technologies includes, for example, medical equipment, environmental surveillance, manufacturing techniques, and much more. This shows that the space industry is very important for future technological advancements. By increasing the public interest, the rate of development can hopefully be increased in the long run, which would be beneficial for both the public and the industry.

6.2 Future work

This is a product with great potential for future improvements, including the implementation of new functions, as well as extending already existing functionality. A function that would have a positive impact from the users perspective would be the possibility to record launches directly to a CZML file, which would then be available for playback. As the number of rockets launched increases, a library containing these could be built, enabling the users to watch a variety of rocket launches at will. Such a function could also include the possibility to fast forward or backward, in order to locate and watch interesting parts of the launch without having to watch it all from the beginning.

In the data processor, some work could be spent on making it more user friendly. For example, some of the code that has to be changed for every missions, such as events lists, rocket model, etc., could be exchanged for a system that uses an initializing file that is loaded as the program is launched.

It is very important for a product like this to be appealing to the users, which means that there will always be room for improvements regarding the visual aspect of the program. The model in this project has been a static rocket model without textures. By changing the model to something more visually appealing, and by implementing effects such as exhaust flames and motor separation, the visual experience could be greatly enhanced. Furthermore, by implementing a method that predicts the rockets trajectory, it would maybe be possible to extrapolate the data points for when the GPS signal is lost, making the visualization look smoother

Finally, the library Cesium is constantly being updated by its developers, and by keeping track on that development, both visual and computational performance could possibly be improved in the future.

References

[1] AGI. Czml guide, 2016. https://github.com/AnalyticalGraphicsInc/czml-writer/wiki/CZML-Guide.

[2] AGI. About agi, 2017. http://www.agi.com/about.

[3] Marcin Ligas; Piotr Banasik. Conversion between cartesian and geodetic coordinates on a rotational ellipsoid by solving a system of nonlinear equations. *Geodesy and Cartography*, 60(2):145, 2000.

[4] Cesium. About cesium. https://cesiumjs.org/about.html.

[5] Swedish Space Corporation. Our experience. http://sscspace.com/about-the-ssc-group/history.

[6] Swedish Space Corporation. Maxus 9 - europas största forskningsraket. 2017. http://sscspace.com/file/documents/science-services-1/rocket/informationsmaterial-maxus-9.pdf.

[7] esri. About arcgis. http://www.esri.com/arcgis/about-arcgis.

[8] International Organization for Standardization. Date and time format - iso 8601. https://www.iso.org/iso-8601-date-and-time-format.html.

[9] Aboelmagd Noureldin; Tashfeen B. Karamat; Jacques Georgy. *Fundamentals of Inertial Navigation, Satellite-based Positioning and their Integration.* 2012.

[10] Google. Earth and javascript, together at last, 2008. https://maps-apis.googleblog.com/2008/06/earth-and-javascript-together-at-last.html.

[11] Google. Announcing deprecation of the google earth api, 2014. https://maps-apis.googleblog.com/2014/12/announcing-deprecation-of-google-earth.html.

[12] Hanspeter Schaub; John L. Junkins. *Analytical Mechanics of Space Systems.* 2 edition, 2009.

[13] Jason Lengstorf; Phil Leggetter. *Realtime Web Apps.* 2013.

[14] York University Libraries. Geospatial data, 2017. http://researchguides.library.yorku.ca/content.php?pid=245987&sid=2176375.

[15] F. Landis Markley. Unit quaternion from rotation matrix. *Journal of Guidance, Control, and Dynamics*, 31(2), 2008.

[16] Vanessa Wang; Frank Salim; Peter Moskovits. *The Difinite Guide to HTML5 WebSockets.* 2013.

[17] NIMA. Dod world geodetic system. *NIMA Technical Report*, 2000.

[18] Jonathan Reid. *HTML5 Programmer's Reference.* 2015.

[19] S. W Shepperd. Quaternion from rotation matrix. *Journal of Guidance and Control*, 1(3):223, 1978.

[20] Steven W. Smith. *Digital Signal Processing, A Practical Guide for Engineers and Scientists*. 2003.

[21] Unity3D. Create games, connect with your audience, and achieve success. `https://unity3d.com/unity`.

[22] K. H. Bhavnani; R. P. Vancour. Coordinate systems for space and geophysical applications. 1991.

[23] Saeed V. Vaseghi. *Advanced Digital Signal Processing and Noise Reduction*. 4 edition, 2008.

[24] Simon Werner. 3d visualisation, 2016. `http://blog.anemoi.nz/3d-visualisation/`.

TRITA EE 2017:083
ISSN 1653-5146