



Operation Silent Giant

Project report

Daniil Galakhov
Chiang Mai, Thailand
2025

Contents

Contents.....	1
Summary.....	3
Project team.....	3
Team mentors.....	3
Motivation.....	4
Prerequisites.....	4
Previous experience.....	4
Key objectives of the project.....	4
Simulation setup.....	5
Rules.....	5
Choice of the target.....	5
Plot development.....	6
Simulated space mission objectives.....	7
Technical assets.....	7
Hardware.....	7
Software.....	7
Simulated launch vehicles and spacecrafts.....	9
Energia heavy load launch vehicle.....	9
DeltaGlider spaceplane.....	11
Envisat heavy Earth-observing satellite.....	13
Simulated launch sites and airports.....	14
SRC Polygon.....	14
Singapore - Changi airport.....	15
Chiang Mai International Airport.....	16
Software development.....	16
Mission planning and analysis software.....	16
Flight control software.....	19
Auxiliary software.....	31
Development methods used.....	32
Final time costs assessment.....	33
Peculiarities of software development for space applications.....	33
Mission simulation.....	34
Preparation.....	34
Legal questions of space debris operations.....	34
Envisat engineering analysis.....	35
Envisat motion analysis.....	36
Envisat deorbiting risks assessment.....	37
Envisat deorbiting method development.....	37
EVA planning.....	41

Mission planning.....	42
Flight.....	46
Briefing.....	46
Mission objectives.....	47
Mission crew.....	47
Launch.....	47
First Obstacles.....	47
Miss or hit?.....	49
Hard work in the darkness.....	51
Astrogation.....	52
Long way home.....	53
Final.....	53
Debriefing.....	54
Insights and conclusions.....	55
About software for space applications.....	55
About algorithms for flight control and mission planning.....	55
About astrogator work during spaceflight.....	55
Psychological aspects.....	56
Philosophy of spaceflight.....	56
Acknowledgments.....	57
Legal information.....	57
Legal Disclaimer.....	57
Use of Software and Intellectual Property.....	57
List of literature.....	59

Summary

Project "Operation Silent Giant" is a part of an ongoing educational and technical development program, designed to improve multidisciplinary competence in space operations. This project purposed to improve skills in software development for space applications, sharpening practical understanding astrodynamics and orbital mechanics, and training participants to manage complex technological systems under stressful, time-constrained conditions.

The project considered simulation of a real space mission, so practically it included two phases: preparation and actual simulation.

During the first phase, the project team developed the plot and objectives of simulation, formulated simulation rules, built hardware and software setup for simulation and developed proprietary software for space mission planning and automated control of simulated spacecraft.

During the second phase, using precise, high-realistic computer simulation, the project team carried-out an analog space mission. Main objectives of the mission was to intercept and prepare to deorbit a massive, non-cooperative satellite (ESA Envisat). Tasks for the team included development of the method for heavy satellite deorbit, mission planning and simulated space flight (including launch, intercept, EVA, reentry, atmospheric flight and landing).

Despite multiple difficulties, the analog space mission was finished successfully.

Project team

1. [Daniil Galakhov](#) - Project leader, lead programmer, astrodynamacist, orbital mechanics and space mission planning expert. Space enthusiast. Tech and aerospace entrepreneur, Unity3D developer (AR/VR and game AI expert).
2. [Anna Apalkova](#) - Computer simulation enthusiast, project artist, visual designer. Space enthusiast. Digital artist, UI designer, sci-fi illustrator.
3. **Rebecca W. Lee** - Operational planning specialist. Hazard response and legal expert. Space enthusiast. Marine logistics and operational security specialist (retired).
4. [ChatGPT](#) - programmer, data analysis specialist, script writer, space flight psychology expert. Generative artificial intelligence chatbot.

Team mentors

- [David A. Vallado](#) - Senior Research Astrodynamacist at Commercial Space Operations Center (CSSI), author of "Fundamentals of Astrodynamics and Applications", algorithms from which were implemented in software used in the project. Additionally, Mr. Vallado provided great help with questions of spacecraft attitude determination.
- [Aleksandr Khokhlov](#) - Head of Small Spacecraft Projects at Geoscan Group. Number of times provided great help with consultations and educational examples.
- [Christophe Bonnal](#) - IAA Chairman of the Space Debris Committee, CNES Senior Expert, topmost expert in space debris. Provided important feedback about assessment of risks of deorbiting extra-heavy satellites.

Motivation

Prerequisites

Studying astrodynamics and related disciplines, in my opinion, requires regular practical application. Astrodynamics (same as marine navigation) is a much wider discipline than just studying the laws of spacecraft movements, and including itself a big practical component.

Thus, during the studies you must participate in practical tasks.

I never participated in any space projects as a full-fledged astrodynamacist (hopefully, just yet?). My real space experience is limited with volunteering in one test-flight space mission ("Edelweis" by Geoscan Group) and interaction with unsuccessful space startups.

Because of all of this, I perform realistic astrodynamic and navigation tasks using simulation software (like space- and flight simulators). This helps me better understand problems I am studying.

Previous experience

As I said earlier, I already performed simulated flight- and space missions, including circumpolar flight around North Pole (using *Microsoft Flight Simulator 2004*), space mission "Rendezvous with RAMU" (using *Orbiter 2016*) and rescue cargo mission to Antarctica after failure of "Rendezvous with RAMU" (using *Microsoft Flight Simulator 2004*).

I used these simulations for practical testing my knowledge of navigation and astrodynamics. And these really gave me useful experience, including understanding of methods of aeronavigation and astrogation and operation experience of different navigational and mission planning tools.

Most important thing is that this simulated missions and my real space experience gave me understanding that:

- Astrogation studies really requires practical application
- Simulation is a good way for improving your skills in astrogation
- For this improvement, you need tools allowing you properly control your spacecraft
- These tools must provide automatic control of key aspects of the navigation process, giving astrogator time to think and decide. They also must have simple controls but be agile in usage.

Key objectives of the project

Summarizing above, **the objectives of this projects were:**

1. Testing software tools, developed as a part of studying the disciplines related to astronautics, astrodynamics and control of spacecraft and atmospheric aircraft.
2. Studying advanced methods of maneuvering in space.
3. Studying methods of planning and analysis of space missions.
4. Improvement of skills of controlling complex technical systems in stressful situations with incomplete data and limited time.

Simulation setup

Rules

To achieve project objectives, next rules had been chosen:

1. Objectives of planned analog space mission must be achieved with realistic, technically and economically founded solutions.
2. Mission must be planned with a realistic level of details, including documenting engineering solutions, detailed mission plan (launch, maneuvers, EVA and reentry planning), assessment of risks, studying of legal questions, etc.
3. Software for mission planning and flight control must be developed using the most precise but most reliable methods and algorithms (like for real space missions).
4. This software must use as few third-party modules as possible (best way - implementation without "off-the-shelf" packages).
5. Simulation software must provide the highest possible level of realism.
6. Simulation must include all required phases of the flight (pre-start procedures, launch, orbital cruising, intercept, close approach, EVA, reentry, atmospheric flight, landing, etc.).
7. Simulation must be done in persistent real-time, without breaks, time skips and changes of simulation rate. Only short pauses, caused by technical reasons (grabbing screenshots, saving simulation process etc.) are acceptable.
8. Saving of the process can be used only for recovery of simulation caused by environmental reasons (like glitches of simulator, power failures, cats falling on a keyboard, earthquakes etc.)
9. Mission can be declared as successful only in case if all mission objectives are completed, all crew members safe and healthy returned, and hardware damage and side effects (damage for third-party objects, environmental pollution) would be within planned boundaries.
10. In case any of the mission objectives would not be completed, or if in simulation a critical situation (which in real life could cause heavy injuries or death of a crew or other people, critical malfunction or destruction of hardware, and other side effects outside planned boundaries) would appeared, the mission must be declared as a failure.
11. In case of emergency landing outside the planned area, participants of the mission must in real life walk the distance equal to the distance from actual landing site to closest inhabited area.

Choice of the target

Process of testing autopilot apps and mission planning software required a task, based on solving targeting problems through the solution of the Lambert problem. So, from the very beginning it was obvious that this should be an intercept of a space object of some sort.

Same time, simulated flight required for such a test, should not be too long - to be convenient for simulation without time skips or simplification of calculations (which would minimise risk of errors and miscalculations during simulation).

Within these constraints, we needed to choose a target for our mission. And that had to be a real, well-known, valuable target - because in other cases that would not be interesting. And that had to be difficult! Because in space it's the right way to choose goals - "not because they are easy, but because they are hard".

We asked ChatGPT, and within offered options - flight to ISS as emergency or support mission, another repair of Hubble, calibration of scientific instruments of Terra (EOS AM-1) after hacker invasion, reconnaissance mission in Fengyun-1C Debris Field, etc. - we suddenly found a mission to Envisat. It was a well-known flagship of ESA scientific spacefleet, it was huge, it was dangerous, it was mysteriously silent since 2012 - and I had a very good add-on with its model for our simulation software.

This solved the doubts - Envisat became the target for our mission.

Plot development

Proper simulation always requires realistic and at the same time interesting plot. The plot must explain to participants the value of the mission, and "ground" it - link it with real life for a proper accounting of the consequences in case of success or failure.

So, for our case, we needed a plot, which would fulfill our project objectives, and same time

- Mean an intercept of Envisat
- Required a rapid response
- Been realistic
- Match the limits of our simulation hardware and software (like available add-ons etc.).

With these limits we, again, turned to ChatGPT to generate a story. Within a few minutes, AI generated a backstory for us, from which we developed the optimal plot for simulation.

So, the story so far...

In early spring 2025 an extremely close approach between Envisat and other space debris had happened (caused by unstable space weather etc.). Space surveillance centers, because of lack of coordination caused by global political instability, predicted this within just a few weeks. Happily, Envisat and piece of space debris passed by each one within hundreds of meters. But this incident raised a question to deorbit Envisat ASAP to prevent possible Kessler syndrome start. Major space agencies did not find a way to respond in time due to preexisting commitments and political reasons. So, a second tier South-East Asia space company "Ultradyn", utilizing old Soviet and US technologies, stepped in with a solution for the problem.

They designed a device suitable for deorbiting extra-heavy satellite with acceptably low risks, and planned a manned mission to Envisat, utilizing experimental heavy launch vehicle (based on Soviet HLLV "Energia") and experimental winged spacecraft (based on US projects of spaceplanes, founded at 1980s - 1990s).

Funded by ISRO, GISTDA and topmost commercial companies of South-East Asia, with support from ESA, the project was accepted and initiated...

Simulated space mission objectives

Follow developed backstory, next objectives for analog space mission was declared:

1. **Design a concept of hardware for reasonably priced and safe deorbiting of Envisat.**
2. **Design a space mission, including rendezvous with Envisat and deployment of deorbiting hardware.**
3. **Execute designed mission using designed hardware and available spacecrafts and launch vehicles.**

With these objectives in mind, we dived into setting up the technical part of simulation.

Technical assets

Hardware

List

1. Laptop Asus ROG Strix G15 G513RW - 2 pcs.
2. Single-board PC OrangePi PC3 - 2 pcs.
3. Smartphone TECHNO POVA 6 Neo - 2 pcs.
4. Network hardware (LAN switches, Wi-Fi access point, etc.)

Limitations

Because me and Anna Apalkova have no access to part of simulation-related equipment, we were not able to use proper controllers (joysticks) for flight simulation. Thus, I had to develop a simple application, allow to use a common computer mouse as a basic joystick for controlling ailerons and elevators of simulated spaceplane.

Software

Preliminary notes

Basic "stack" used for this project was Python + Orbiter 2016. This solution was used because these systems are freeware, and both of them, actually, have no analogs in their functionality and versatility.

Despite a number of limitations, Python provides simplicity and flexibility, highly exceeds those of any programming language, and the same of systems like Matlab or Wolfram Mathematica.

Orbiter 2016, from the other side, is ***the only existing space flight simulator***, providing industrial-grade simulation precision in "off-the-shelf" configuration.

Other existing "space simulators" are nothing but just computer games. They do not match realism and precision requirements of the project.

List

1. Orbiter 2016 v.160828
 - 1.1. Add-on ENERGY project (by Yuri Kulchitsky)
 - 1.2. Add-on ENVISAT (by Brian Jones)
 - 1.3. Add-on PAYLOAD MANAGER (by Yuri Kulchitsky)
 - 1.4. Add-on ReFuelMFD (by Roman Starkov)
 - 1.5. Add-on Scenic 500 (author unknown)
 - 1.6. Add-on Chiang Mai Launch Base (author unknown)
 - 1.7. Service add-ons (Spacecraft4, FlyByWire v.0.91, etc.)
 - 1.8. Metamods (OrbiterSound, etc.)
 - 1.9. Modified DLL **LuaInterpreter.dll (by Daniil Galakhov)**
 - 1.10. Add-on **OrbitalAP MFD (by Daniil Galakhov, supported by ChatGPT)**
 - 1.11. Add-on **Aerobraking MFD (by Daniil Galakhov)**
 - 1.12. Add-on **ReentryAP MFD (by Daniil Galakhov)**
 - 1.13. Add-on **AtmosphericAP MFD (by Daniil Galakhov)**
 - 1.14. Add-on **Telemetry MFD (by Daniil Galakhov)**
 - 1.15. Add-on **Drag Augmentation Device (by Daniil Galakhov, Anna Apalkova and Rebecca W. Lee)**
2. **manage_bases (by ChatGPT, managed by Daniil Galakhov)**
3. **input (by Daniil Galakhov)**
4. **telemetry_server (by ChatGPT, managed by Daniil Galakhov)**
5. LUA 5.1.2 + Eclipse IDE
6. Python 3.9.13 + IDLE IDE
 - 6.1. numpy
 - 6.2. matplotlib
 - 6.3. sgp4
 - 6.4. flask
 - 6.5. http.server
 - 6.6. pynput
 - 6.7. vgamepad
7. CesiumJS 1.129
8. Git 2.42
9. Blender 4.1
10. Msh_Maker
11. SAI version 1.2.5
12. Different freeware apps (for work with vector images, reading .pdf files, text/video communication etc.)
13. ChatGPT (gpt-4-turbo, March 2025).

Settings

To provide required level of realism, Orbiter 2016 was set up to emulate:

- Complex flight model (including wave drag, compressibility drag and effects of boundary layer at transonic, super- and hypersonic speeds)

- Damage and failure simulation
- Limited fuel
- Non-spherical gravity sources
- Radiation pressure
- Gravity-gradient torque
- Atmospheric wind effects

Simulation of Earth's gravity was realized with Legendre polynomial series with coefficients J2-J5. For Earth atmosphere Jacchia71-Gill model was used.

Limitations

Because Orbiter 2016 is already a very old application, it suffers from incompatibility with modern hardware and operating systems, incompatibility of add-ons within the simulator itself and lack of documentation.

There is a newer, community-maintained version of Orbiter simulator (Orbiter 2024), but it is unstable and suffers from add-on incompatibility even more than Orbiter 2016. Because of this, I decided to use Orbiter 2016 v.160828.

Orbiter 2016, because of its complexity, has almost no autopilot software. So, all flight control software for the project was developed "from scratch".

Add-ons for Orbiter 2016 can be developed using C++ or LUA. Because of insufficient documentation for C++ compiler configuration needed for building add-on .dll-s, I decided to use LUA as the main programming language for development flight control software. LUA appeared to be a good and relatively comfortable programming language. But despite this, in the end I faced a need to work with C++. Using my previous C++ experience and reverse-engineering methods, compiler configuration was found, and part of the project was written with C++.

Because of compatibility issues, LUA version was limited to 5.1.2

Python is suffering from compatibility problems between versions, so for development I decide to use version 3.9.13 only.

CesiumJS is working unstable with weak internet connection. Also, sometimes developers of Cesium adding changes without documenting them. So for development purposes was chosen offline (local) CesiumJS 1.129.

As version control system was used Git 2.42

A payware version of ChatGPT was used.

Simulated launch vehicles and spacecrafts

Energia heavy load launch vehicle

Description

Energia HLLV is a super-heavy launch vehicle, developed in the USSR in the late 1980s and proposed for launch to LEO payloads with a mass up to 110 tons.

Energia was originally used for the Buran spaceplane, and has a unique design, allowing it to lift payloads of almost any shape (the only limit is payload mass).

Launch vehicle consists of 2 stages.

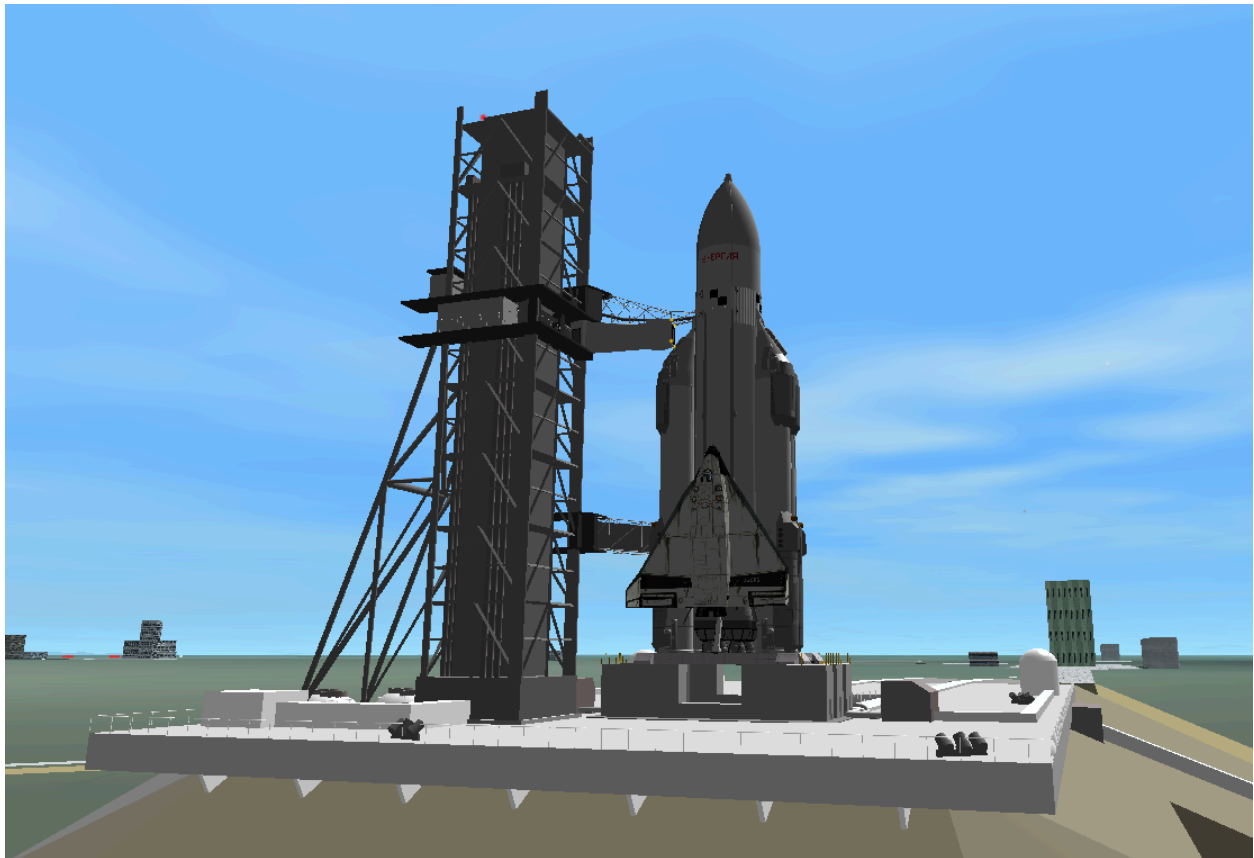
First stage includes 4 reusable booster modules (so called "Block A"), placed in pairs around the central module of the second stage. After decoupling from the launch vehicle, they execute a return maneuver, entering the atmosphere, and after aerobraking, landing via parachutes in the planned area (on water or on ground), from where they delivered to the launch facility.

Second stage includes one big expendable booster (so called "Block C").

Both stages use RP+LOX propellant composition.

Important! Energia HLLV does not deliver payload to circular orbit (for prevention of increasing space debris amount). So, the payload must have its own propulsion system to circularize its orbit.

Following the simulation plot... *design and technologies of Energia were transferred to international partners by the Russian Space Agency in the early 2000s within the "SRC Polygon" project. After Russia left the project in the late 2000s, project documentation stayed under control of the project. Using this data and reverse-engineering of samples of Energia LV provided by Ukraine and Republic of Kazakhstan, Asian commercial companies found small-scale production of improved versions of old soviet launch vehicles. Modern control systems improved usability of the launcher, and, despite its high cost and long manufacturing duration, it still can be a competitor for SpaceX Falcon Heavy.*



Pic. 1: Energia HLLV with attached DeltaGlider CPS spaceplane on the launchpad at SRC Polygon, Indonesia.

Energia HLLV tech specs

Full launch mass:	2419 t
Payload mass (LEO, nominal):	105 t
Length (max):	60 m
Diameter (by fins of blocks "A"):	18 m
Full thrust (sea level):	34270 kN
Max overload on launch:	2.95 G

Justification of choice

Energia and Falcon Heavy are the only LV add-ons for Orbiter 2016, which are able to launch to space payload heavier than 25 tons, yet still realistic and working stable in simulation. Using SpaceX LV would focus simulation on the SpaceX spacecraft family (like Crew Dragon etc.). It was obvious that using the resources of SpaceX it's definitely possible to perform the mission we are trying to simulate. Because of this, the simulation would not be challenging enough. That's why Energia was chosen.

DeltaGlider spaceplane

Description

DeltaGlider is a fictional, medium-sized, high delta-v spaceplane, utilizing a chemical propulsion system using Aerozine50/N₂O₄ propellant composition.

Following the simulation plot... *DeltaGlider was designed for the "SRC Polygon" project by United Space Alliance, as a successor of projects of spaceplanes, started by Boeing and Lockheed Martin in the 1980s and 1990s. From the very beginning of the project DeltaGlider was designed as a civilian spacecraft, oriented for commercial use. Because of this it is simpler in control than its predecessors, but more expensive and complex in design than SpaceX Crew Dragon. With a loaded mass about 27 tons, delta-v within 2.5 km/s and crew capacity up to 5 it is still good for active operations within LEO, and, using booster stage, is able to reach GSO and return to Earth.*

Modifications

Originally, DeltaGlider was provided in Orbiter 2016 distributive as the default spacecraft. But in the original configuration it was equipped with thrusters with unrealistic $I_{sp}=40000$ s. Of course, that did not match the rules we chose for the project. So, DeltaGlider thruster parameters were changed to realistic using a special LUA script. In this document any mention of DeltaGlider always implies this modified version ("DeltaGlider CPS" - with Chemical Propulsion System) of the mentioned simulated spacecraft.



Pic. 2: DeltaGlider spaceplane on the runway at SRC Polygon, Indonesia.

DeltaGlider tech specs

Empty mass:	11.0 t
Max takeoff mass:	24.9 t
Length:	17.76 m
Wingspan:	17.86 m
Main thrusters thrust - specific impulse (vacuum)	465.4 kN 316 s
Retro thrusters thrust - specific impulse (vacuum)	31.12 kN 310 s
Hover thrusters thrust - specific impulse (vacuum)	140.0 kN 305 s
Total delta-v (main tank)	2.49 km/s
Crew capacity	up to 5
Life support system lifespan (with full crew)	7 days

Justification of choice

DeltaGlider with a realistic propulsion system has mediocre performance as a spacecraft and is clumsy as a plane. Actually, it is a visual explanation why crewed spaceplanes, despite their great look and impressive concept, did not find a place in modern practical astronautics.

But in Orbiter 2016 DeltaGlider is one of the very few "relatively realistic" manned spacecrafts which have normal exterior and interior visualisation, good simulation of internal systems and working without crashes of simulation app.

Stable working and add-on-compatible alternatives included only an add-on implementing Crew Dragon. This spacecraft was more convenient than DeltaGlider, and in some configurations had comparable performance. But in case of usage of this spacecraft, simulation would become focused on usage of SpaceX LV and infrastructure, which, again, would make simulation less challenging. That's why DeltaGlider was chosen.

Envisat heavy Earth-observing satellite

Description

Envisat ("Environmental Satellite") is a large Earth-observing satellite which had been launched in March 2002, and became inactive in April 2012. It is still in orbit and considered space debris. Operated by the European Space Agency (ESA), it was the world's largest civilian Earth observation satellite.



Pic. 3: ESA Envisat in orbit over East Siberia.

Envisat tech specs

Launch mass:	8211 kg
Dimensions:	26 x 10 x 5 m
Orbital parameters - mean altitude: - eccentricity: - inclination:	773 km 0.00042 98.4°
Launch date:	1 March 2002
Contact lost:	8 April 2012

Simulated launch sites and airports

SRC Polygon

Description

Fictional launch site placed near Duri village, north of Pekanbaru town, on Sumatra island, Indonesia.

Following the simulation plot... *SRC Polygon was founded in the early 2000s by the US and Russia, within international space partnership. Initially SRC Polygon was built as a launchpad for missions of international space projects (including ISS and Sea Launch), and as a test site for civilian launch vehicles.*

But in the late 2000s Russia left the project because of inner political reasons. In the early 2010s US also stopped funding of the project, and for a time the future of the launch site was unclear.

Despite these troubles, commercial companies of South-East Asia found funding for the launch site.

With a time, SRC Polygon turned into a unique technological and business hub for space startups and "new space" companies, developing innovative space solutions and producing launch vehicles, satellites and spacecrafts.

Launch site of SRC Polygon equipped with 3 launchpads, able to operate light, medium and super heavy launch vehicles. VABs of launchpads are able to work with all ranges of payloads. Industrial facilities of residential companies allow them to produce most advanced samples of space tech.

Airport of SRC Polygon has 2 runways, able to receive planes and helicopters of any type.

SRC Polygon has a local flight control center, advanced surveillance facilities and deep space communication facility (placed north-west from the launch site in the Dumai town).

Technical details

Coordinates:	N 1.274° E 101.097°
Launchpads:	2 light/medium 1 super heavy
Runways:	08/26 3500 m 13/31 2800 m

Modifications

SRC Polygon was provided with the Energia HLLV add-on. But, it was made for earlier versions of Orbiter (2006/2010). So, in Orbiter 2016 it had compatibility troubles, preventing proper landing on its runway. To fix this, I had to change the layout of ground base facilities though editing source code of this add-on.

Justification of choice

As I said earlier, SRC Polygon was provided with Energia HLLV add-on, thus, it had all ground equipment required for Energia launch. Same time, it was positioned very well (close to equator in a very convenient area). Using other launch sites for Energia would require installation of additional add-ons or editing simulation scenario files. That could require additional time, so to avoid this, "off-the-shelf" solution (SRC Polygon) was chosen.

Singapore - Changi airport

Description

One of the biggest aviation hubs with highest cargo traffic within all airports in the world, Changi lies in the city of Singapore, and is used for military planes as well as for civilian air traffic.

Technical details

ICAO:	WSSS
Coordinates:	N 1.359° E 103.989°
Runways:	3 x 02/20 4000 m

Justification of choice

Closest airport to SRC Polygon, available in Orbiter 2016 add-ons, was chosen as reserve landing site for simulated mission.

Chiang Mai International Airport

Description

Airport of the city of Chiang Mai in northern Thailand, is used by civilian air traffic as well as by Royal Thai Air Force (Wing 41).

Technical details

ICAO:	VTCC
Coordinates:	N 18.767° E 98.963°
Runways:	18/36 3100 m

Justification of choice

Looking ahead - mission analysis showed that Chiang Mai airport is very close to the reentry trajectory of DeltaGlider. Because of this, it was chosen as secondary reserve landing site for the mission.

Software development

Mission planning and analysis software

Description

This software was made to provide functionality identical to one of GMAT, Poliastro and other mission planning tools. For its realisation I choose Python as a calculation environment, and matplotlib and CesiumJS as visualisation tools. Most of the algorithms used in this software were taken from David Vallado's "Fundamentals of Astrodynamics and Applications. 4-th edition".

To say honestly, one of the "unspoken" objectives of the project was to check, is it possible to plan a space mission using only knowledge from "Fundamentals..." and related books. Practice showed - **yes, this is possible**.

Now, let's focus on the most important parts of this software.

Launch parameters calculation

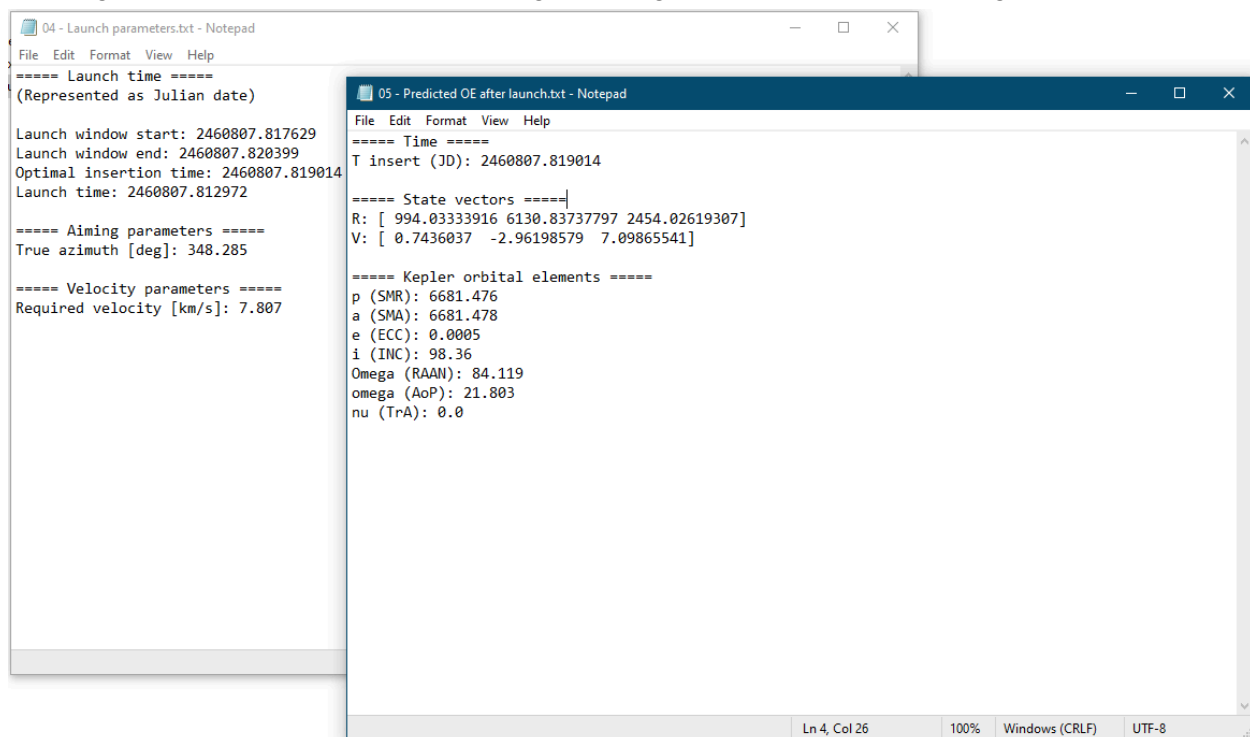
One of the things briefly touched in "Fundamentals..." was questions about launch window, launch azimuth, proper aiming of launch vehicle and prediction of orbital parameters immediately after launch.

Of course, most of you may say that this is not so important, because in typical cases we just need to wait while surveillance stations provide for you state vectors of your satellite. This is correct for typical commercial missions to SSO, but in rapid response missions you may be constrained in time.

Some of you may say that even in this case it is not important, because this is not a task for an astrodynamist. But in my opinion **a proper astrogator must be able to control his vehicle from the very launch till it touches the ground** - and even manually if this is required. Success of the mission is your responsibility! That's why you need to be able to aim your launch vehicle to that only right point in spacetime where you really need it.

So, for launch it's good to be able to predict time and launch azimuth, allowing you to put payload to required orbit. For planning further maneuvers it's good to be able to predict state vectors of spacecraft after it separates from the launch vehicle.

My software for calculation of launch parameters is able to do both things. It is still not perfect (for example, launch azimuth calculated as absolute geodetic, without accounting for magnetic declination of geographical position, and state vectors prediction utilizing circular orbits which is not right for some launch vehicles). But this software allows to launch simulated LV to right orbit, and provides a fair enough starting point for mission planning.



The image shows two overlapping Notepad windows. The background window, titled '04 - Launch parameters.txt - Notepad', contains the following text:

```

===== Launch time =====
(Represented as Julian date)

Launch window start: 2460807.817629
Launch window end: 2460807.820399
Optimal insertion time: 2460807.819014
Launch time: 2460807.812972

===== Aiming parameters =====
True azimuth [deg]: 348.285

===== Velocity parameters =====
Required velocity [km/s]: 7.807
  
```

The foreground window, titled '05 - Predicted OE after launch.txt - Notepad', contains the following text:

```

===== Time =====
T insert (JD): 2460807.819014

===== State vectors =====
R: [ 994.03333916 6130.83737797 2454.02619307]
V: [ 0.7436037 -2.96198579 7.09865541]

===== Kepler orbital elements =====
p (SMR): 6681.476
a (SMA): 6681.478
e (ECC): 0.0005
i (INC): 98.36
Omega (RAAN): 84.119
omega (AoP): 21.803
nu (TrA): 0.0
  
```

Pic. 4: Launch parameters calculator report

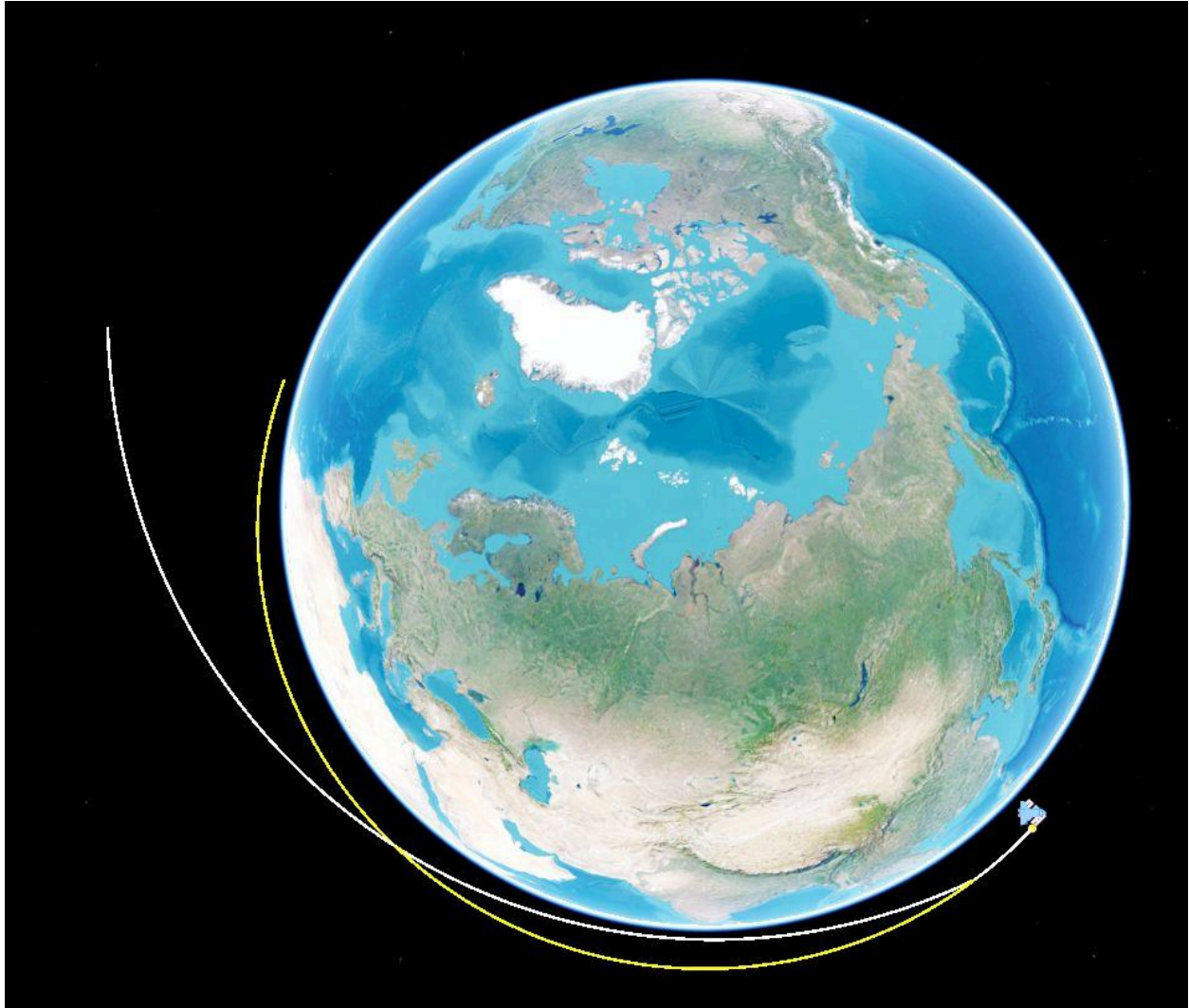
Lambert problem solver

Second important piece of software required testing through this project was Lambert problem solver. In "Fundamentals..." Mr. Vallado mentioned that, despite its versatility, Lambert problem solver has a number of limitations, especially linked with a way to apply it. The only good way to understand its strengths and weaknesses was to use it for solving practical targeting problems.

So I dive deeply into the appliance of targeting algorithm. As a test subject was chosen a targeting algorithm, utilizing Lambert problem solution with universal variables. Multiple tests showed points of put attention on, especially:

- questions of choosing short/long arc
- problem of collision detection
- strict result checking requirements
- questions of convergence

Using the results of this testing I implemented a properly working targeting application. (More information about insights from studying Lambert problem solver you may find on my [LinkedIn page](#))



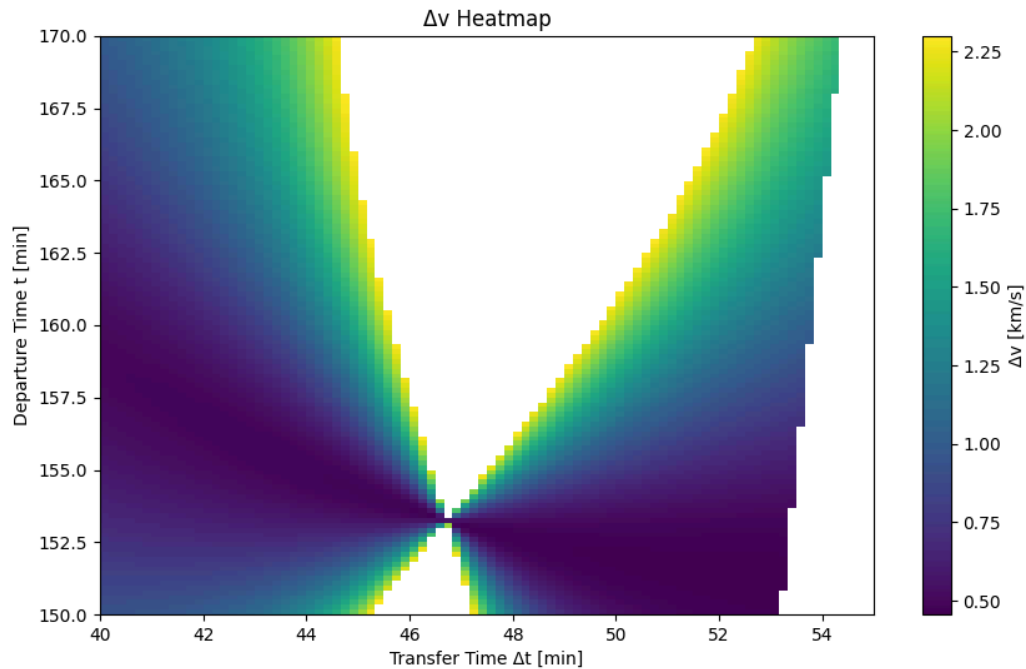
Pic. 5: "Intercept on the knife edge" - result of work of Lambert problem solver.

Delta-v heatmap plotter

Last important thing I need to mention in the section about mission planning software is a tool for plotting delta-v heatmaps. This app plots a delta-v heatmap for all Lambert transfers between two states in spacetime within given constraints of transfer duration and start time, generating human-readable representation of possibilities for maneuver.

This way of representation appeared to be very useful for planning transfers during mission analysis.

Application utilizes the targeting algorithm mentioned in the previous section.



Pic. 6: Delta-v heatmap - darker the better.

Flight control software

Avionics and flight control software simulation in Orbiter 2016

Before we can continue to the flight control software itself, I need briefly describe principles of simulation of avionics in Orbiter 2016.

Avionics in this simulator can be realized with two main ways - as a "built-in" device, hard-coded in DLL of an add-on (which is often used for simulation of standalone, analog or mechanical flight controls), or as an application for Multi Functional Display (MFD).



Pic. 7: "built-in" attitude indicator



Pic. 8: "Surface" MFD with attitude indicator

MFD represents a flight computer with color display and programmable buttons. It runs RTOS, and is able to execute different applications. It works in hard real time mode which is linked with real time through simulation cycle update. Applications for MFD can be written in C++ 6.0 or LUA 5.1.2. MFD is connected with the "data exchange bus" of the spacecraft. This "exchange bus" is represented (and can be accessed) as an object in code, through which it can receive information from virtual sensors (IMU, speedometers, radio- and barometric altimeters, magnetic and radio compasses, GPS sensors, etc.), and send commands to controllers of different spacecraft systems (thrusters, RCS, aerodynamic surfaces, radio transceivers etc.) Sensors and hardware controllers are represented as methods of "data exchange bus" object. MFD also has "non-volatile memory", to which it can save different data. This "memory" is, actually, a sub-directory in Orbiter 2016, so MFDs are able to read and write files from/to the disk of the computer where Orbiter 2016 is running.

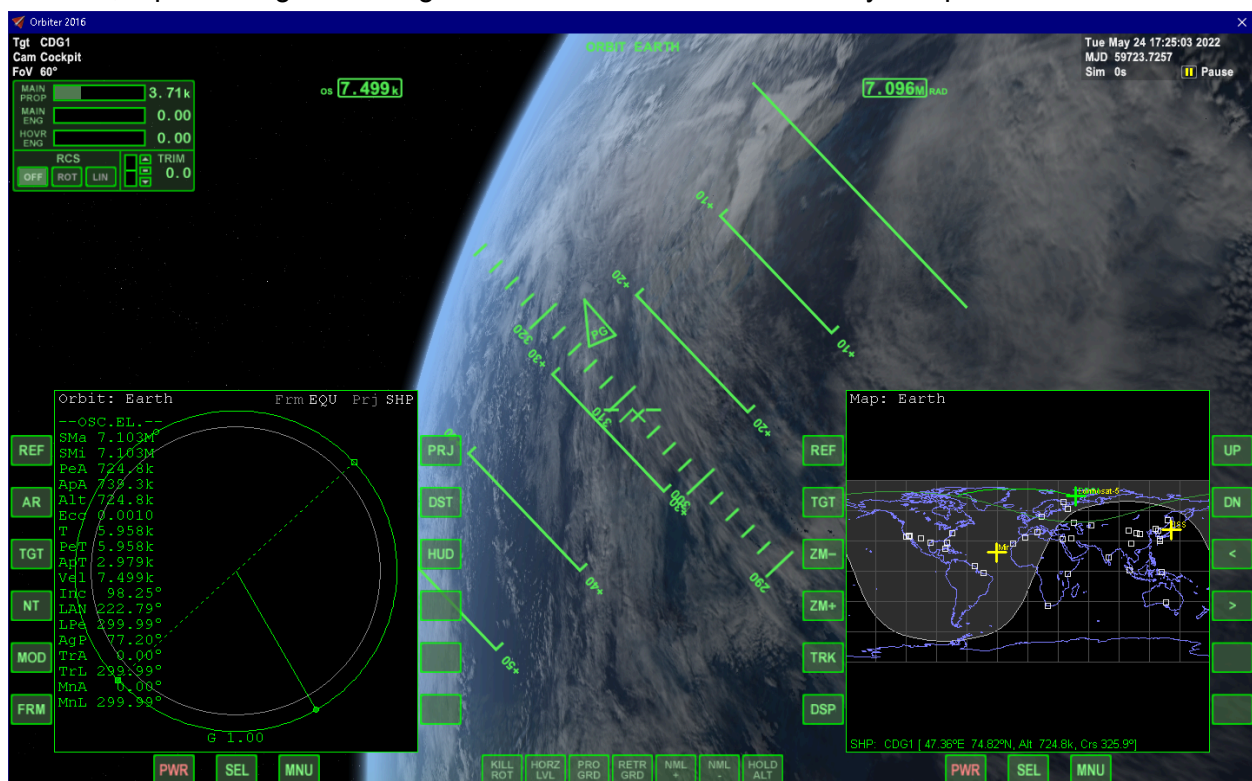
All these methods with short examples represented in Orbiter 2016 documentation.

Additionally, MFD apps can utilize other functions (for example, control simulation flow), but we will omit this in this paper because it is out of the area of our studies.

In architecture of Orbiter 2016, MFD is an interface wrapper over methods of class representing simulated spacecraft.

In Orbiter 2016, most of spacecrafts are equipped with at least 1 MFD. For some spacecrafts MFDs can be disabled or replaced with different systems. DeltaGlider by default is equipped with 2 MFDs, and 1 additional MFD can be connected to it with ExternalMFD add-on.

Besides built-in avionics and MFD, Orbiter 2016 provides functionality of "virtual external camera", representing something like the external camera of a "Soyuz" spacecraft.



Pic. 9: Orbiter 2016 "virtual external camera" UI. Flight information and 2 MFDs are visible.

This is the "default" user interface of the simulation, available for most of the spacecrafts in simulation. It can be treated as part of the flight control UI, because on this screen represented most basic flight information for selected spacecraft, and first two of available MFDs. Same as with MFD, for some spacecrafts it can be disabled or replaced with a different UI.

Orbiter 2016 provides other UI modes (control panel mode, virtual cockpit mode) and external views but their functionality is the same from the point of view of control, so they are realized mostly for aesthetics.

Important note! By default, Orbiter2016 does not simulate any relativistic effects, such as delay in data transmission caused by limited speed of light etc.

Orbital AP

Description

Orbital AutoPilot (OrbitalAP) was the core element of all software set, "a main steering wheel", allowing the mission to be simulated and the whole project to be done.

The purpose of this software was:

1. Calculate orbital maneuvers
2. Execute calculated burns the proper way in the proper time.

And this is not as simple as it possibly looks. Those of you, who worked with real missions will understand me. Orbiter 2016, despite not simulating real satellite/spacecraft hardware, greatly imitating "feeling" from it. MFDs of DeltaGlider have very limited performance (comparable with those of Pentium 2 CPU) - enough to real-time control thrusters and aerodynamic surfaces, but too small for fast enough optimization of Lambert transfer. Simulated sensors are convenient and precise enough (fortunately I didn't face a need to filter and convert "raw readings")... But, just for a start, their readings are in a coordinate system fixed to spacecraft. This causes that you need to develop conversion mechanisms, for which you need to develop enough math - libraries for work with vectors, quaternions, auxiliary math, and so on. Next, for proper control of attitude and thrust of spacecraft, you need to make it aim the right way, so you need to develop control algorithms for RCS, main thrusters, etc. This causes that you need to add to code implementation of PID controllers, and so on. And you need to do this yourself - because LUA does not have proper and greatly tested libraries.

So, development of OrbitalAP required a lot of effort, but was a very, extremely useful experience.

In a nutshell, OrbitalAP allows its operator to plot optimized Lambert transfer between two states in spacetime. After this, it can automatically execute calculated transfer, by aiming spacecraft in calculated direction and burn thrusters to generate required amount delta-v at precalculated moments. Circular error probable (CEP) of intercept for transfer LEO->GSO is about 15 kilometers. During cruise flight, OrbitalAP is able to do stationkeeping by correcting orbital drift with a 6D PID correction system.

Additional functionality of OrbitalAP includes assessment of discrepancy between current state, estimated state and next planned state, visual guidance subsystem, allowing the operator to execute planned maneuvers manually, in machine-guided mode, and delta-v budget calculator.



Pic. 10: OrbitalAP user interface, idle mode.

Attitude control methods

Development of a method of attitude control appeared to be probably the hardest task in the whole project.

Original idea was pretty simple - just enter a velocity change vector $\vec{v}$ given in IJK (or VNB¹) coordinate system, and, well, just aim the fore of a spacecraft towards this vector... Well, somehow.

But this "somehow" appeared to be the most difficult thing.

As a starting point I had only:

- spacecraft-centered spacecraft-fixed left-handed ZXY CS
- current velocity vector from simulated IMU, presented in ZXY CS
- current gravity vector from simulated artificial horizon, presented in ZXY CS

¹ Velocity-Normal-Binormal, typically used in GMAT. Same as LVLH.

Actually, Orbiter "under the hood" utilizes left-handed CS - because DirectX, used during development, uses this CS, and creator of Orbiter prefer use it to avoid errors.

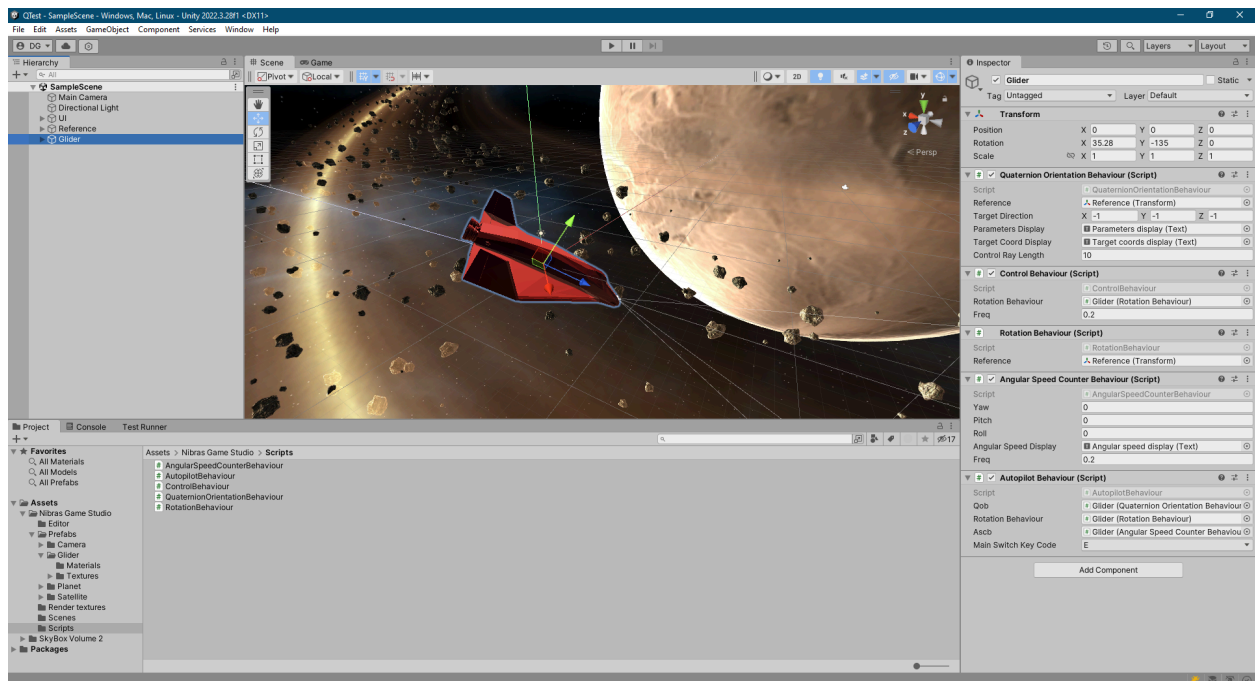
So, I needed to transform this CS to right-handed, and using data from sensors, transform velocity change vector from IJK to local CS.

Here came the greatest trouble.

Initially, I expected just to transform all vectors to right-handed CS, and then, using simplest quaternion multiplication logic, rotate $\vec{v}$ from IJK to ZXY CS. I did such things previously being a Unity3D developer, and there that classic method worked. But things were not so simple here.

Quaternion multiplication did not work properly, at all. I implemented it multiple ways, but each time I faced a situation when the result vector was strangely inverted. I implemented 4 types of quaternion-vector multiplication, copied the algorithm of quaternion multiplication from Unity3D sources, from Wertz book and from Hamilton's original explanation, but still faced strange inversions of aiming vector from time to time.

I made a digital twin of DeltaGlider in Unity, and applied there same algorithms of orientation. In Unity everything worked!



Pic. 11: DeltaGlider digital twin in Unity3D.

In despair, I wrote to Mr. Vallado, asked for help with understanding the problem. He sent to me an article by Renato Zanetti, named "ROTATIONS, TRANSFORMATIONS, LEFT QUATERNIONS, RIGHT QUATERNIONS?", and this helped a lot.

Briefly, my first mistake was in interpreting the results of the calculation. In CG/3D and in aerospace problems we are using *same* quaternions. But (if to say very simple) they are doing different things with entities they are applying to. In 3D they *rotate* vectors *in external* CS. In attitude dynamics we facing passive interpretation of vector rotation, i.e., here quaternions "*turning coordinate system around vector*", or, following Zanetti terms, *transform* them.

This insight helped me to defeat inversions of aiming vector. I hoped that was the end of the problem. But - it was not!

I turned back to code after a few months break because life troubles required urgent solutions. And after a few tests I found out that the quaternion transformation of the aiming vector still has strange errors. It looked pretty weird - quaternion started to "lose precision" when the aiming vector turned more than 45° from forward direction. That was not uniformly, but definitely following a strange law - precision losing on increasing pitch angle. Solution of the problem required almost 2 weeks of work. Using ChatGPT as co-programmer, I totally covered vector and quaternion LUA modules with tests, and made a full-scale test solution with about 100 test cases. Quaternion and vector calculation functions were 100% correct. In synthetic tests all worked perfectly. But on simulated spacecraft losing precision still applied.

Almost insane, I left aside the quaternion transformation, and tried to use Rodrigues vector rotation formula for turning the $\vec{v}$. And suddenly it worked fine. That forced me to watch the problem from another point of view, and I understood my second mistake. The operation I was trying to make *was not a rotation or transformation*. IJK and ZXY CS were *not linked*. So, I need to *present* $\vec{v}_{\text{IJK}}$ in ZXY system, and Rodrigues rotation formula appeared to be the simplest way for this. After applying it for work in two planes, and setting a PID for precise control RCS on rotation, I achieved all-3D aiming of DeltaGlider with precision within 0.1 degree.

LuaInterpreter.dll modification

Besides the troubles with attitude dynamics, unusual choice of coordinate systems in Orbiter 2016 required reverse-engineering and source modification of existing DLL in the simulator.

"Under the hood", Orbiter keeping all coordinate data in left-handed CS with the center in barycenter of the solar system. For conversion of state vectors of spacecraft into planet-centered systems like IJK (ECI), we need to transform them with a planet rotation matrix and planet obliquity matrix. These matrices were not accessible from the LUA subsystem.

Studying the C++ sources of the Orbiter 2016, I found a place where this data was accessible. I thought about how to pull this data out, but then chose just to add functions with conversion into the existing C++/LUA interface of Orbiter. That required studying C++ compiler settings, but, fortunately, I had enough experience with C++ to configure VS 2022 MSVC compiler to build a .dll for Orbiter.

Calculation module and Lambert problem solver

Calculation module utilizes the same algorithms as mission planning and analysis software. But, as I mentioned earlier, Orbiter 2016 MFD has pretty low performance and working in "hard real time". So, the main difficulty was to optimize calculation algorithms (especially, Lambert problem solver) to reach high speed of maneuver calculation on such a slow computer.

That was done through usage of LUA coroutines in targeting algorithm and optimisation of search parameters of delta-v heatmap plotter. With this optimisation, I achieved next performance values:

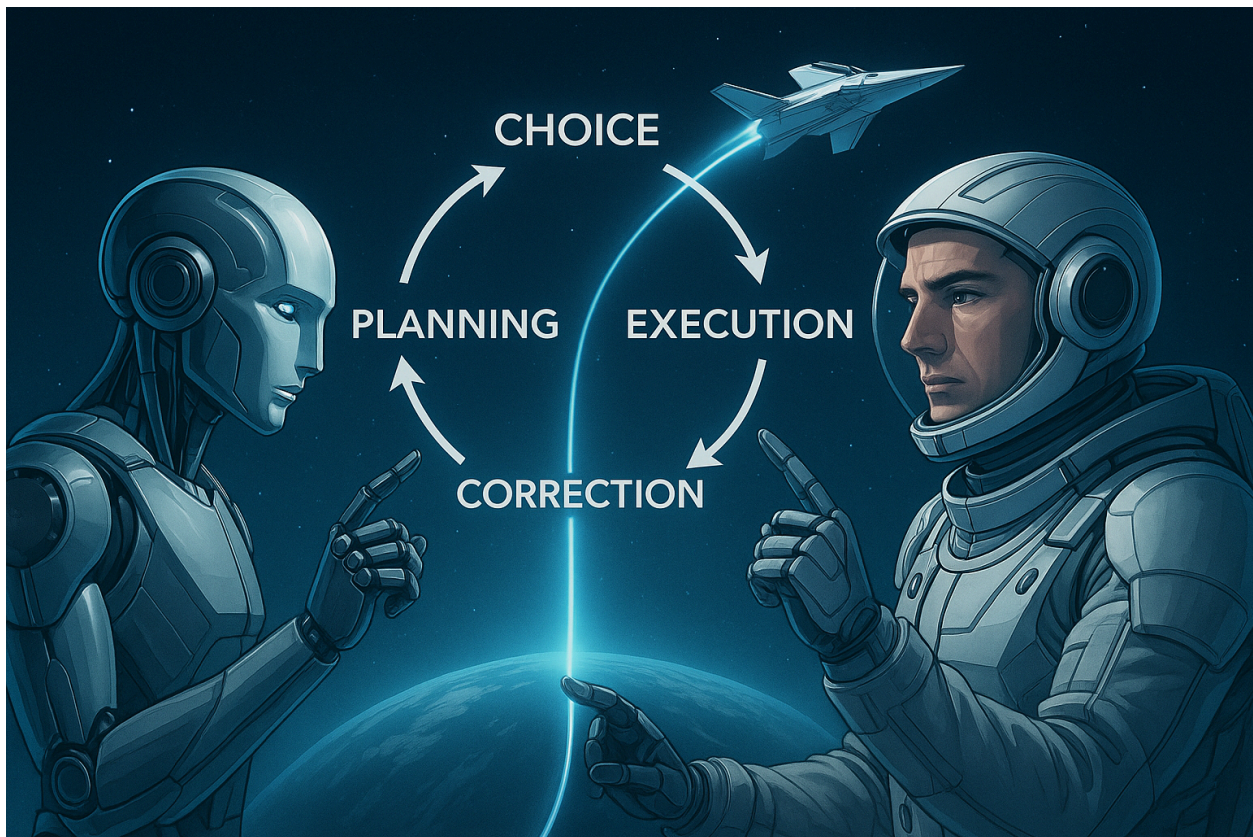
- **Time-constrained Lambert transfer:** 0.03 seconds
- **Delta-v optimized Lambert transfer:** 0.23 seconds

Formalizing states through "choices" conception

This was the second of most difficult things I needed to develop in this project. I found a way to steer spacecraft, and found a way to calculate maneuvers. But I need a way for my software to *interpret* and *manipulate* these maneuvers. Machine should be able to *assess* and *understand* these maneuvers - to protect the operator from a mistake. And the human should be able to do the same! I needed to make a metric to compare the current state of the spacecraft with the planned state, and link this with maneuvers done before and those that would be done later. And all of this must be transparent, human-readable, easy to understand. After a long thinking - about a week required to build and formulate the concept in my mind - I found a way to realize this. But that should be done not only with just code.

So, I built an architecture based on "choices" - a decisions which change and determine states - which would be made by human and machine together.

In this paradigm, a "choice" defines the next desired state in space and time. The human (astrogator) defines the choice, then machine plans, executes, and maintains it. When the mission context shifts, a new choice is made — by man and machine together.



Pic. 12: Artistic representation of "choice"-based AP architecture (generated by ChatGPT)

6D PID drift correction

Last very important functionality of OrbitalAP that needs to be mentioned is the drift correction system. For precise following maneuvers you planned during the mission planning phase, you need to execute them in proper (planned!) states in spacetime. But the spacecraft is always drifting, affected by gravity anomalies, space weather, gravity pull of third-party bodies

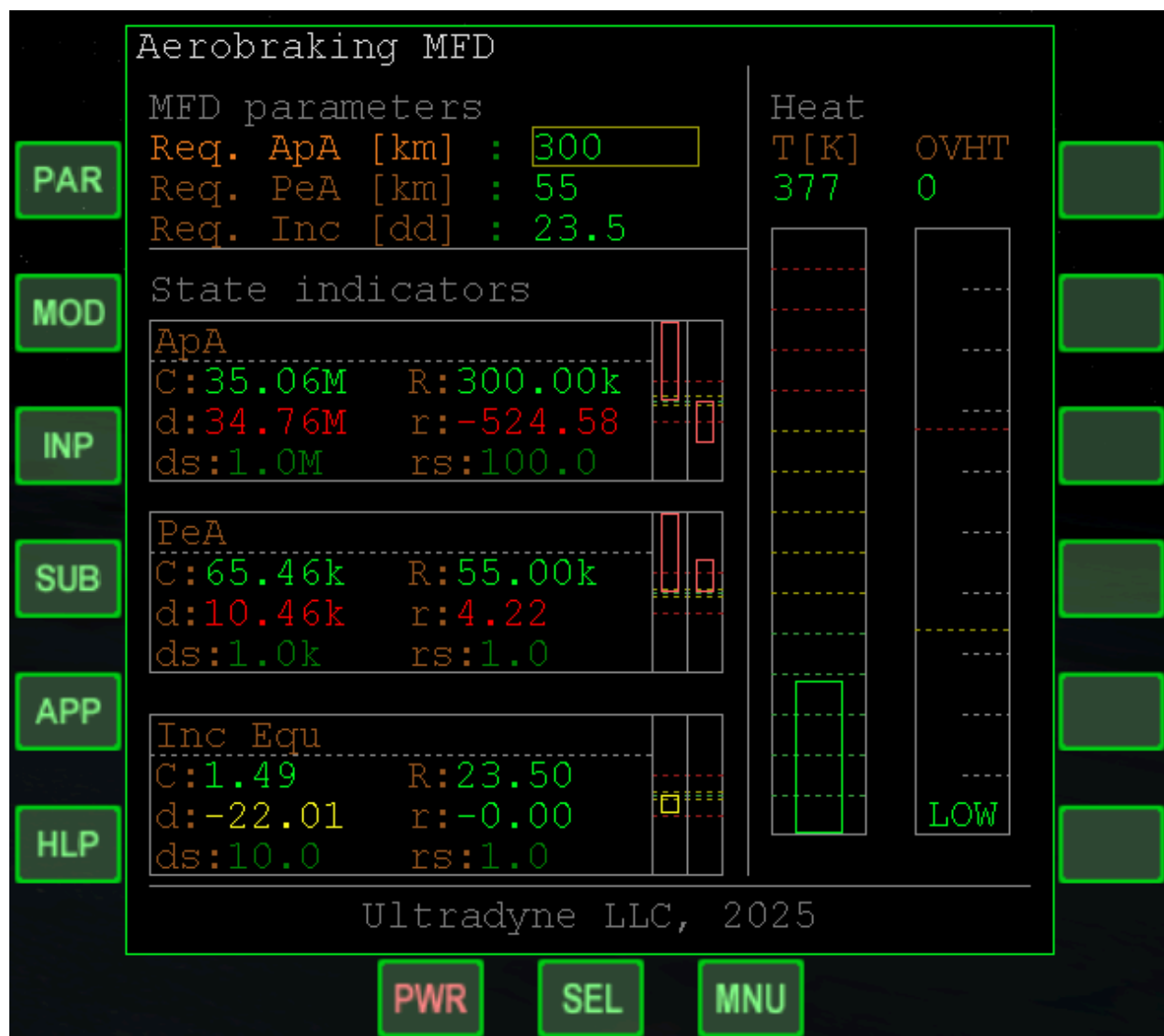
and other factors. Drift values can reach hundreds of kilometers per day from position calculated via numerical propagation methods.

To prevent this, I added into the OrbitalAP a drift correction system. Using PID, it calculates correction impulses and reduces orbital drift. With enabled drift correction within 24 hours test on orbit with altitude 730 km next results were achieved:

- **Mean position deviation:** 805 meters
- **Mean velocity deviation:** 4 m/s
- **Delta-v spent:** 291 m/s

Not cheap by delta-v. But it worked!

Aerobraking MFD



Pic. 13: Aerobraking MFD user interface, active.

Aerobraking was made as auxiliary software, measuring and indicating parameters of aerobraking in the high Earth atmosphere. Processing current state of spacecraft and braking rate during non-reentry aerobraking, it predicts orbital parameters after exit from atmosphere.

Additional menu on the left indicates calculated temperatures and overheat level on the frontal (faced to airflow) area of DeltaGlider.

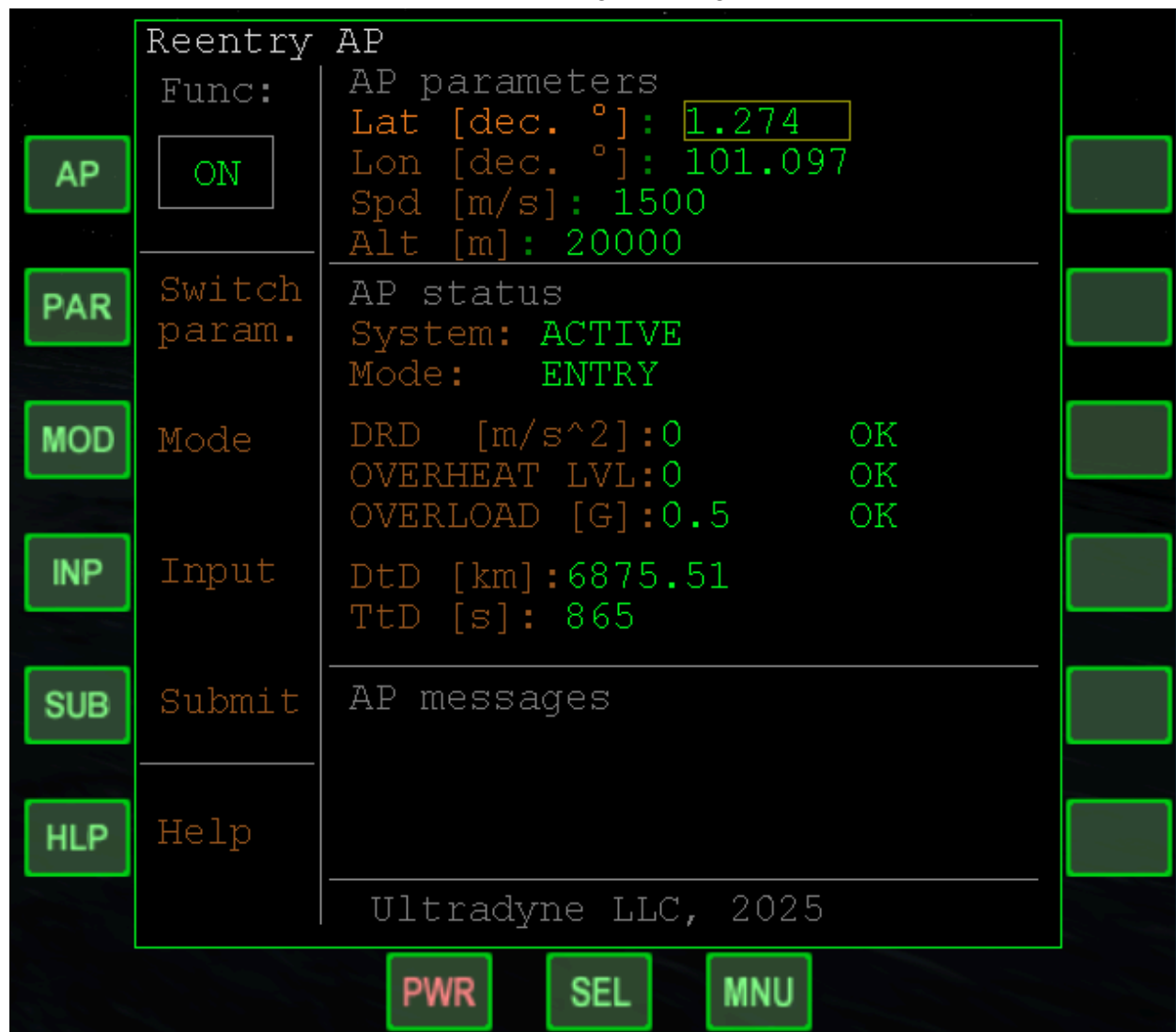
MFD purpose is to help the pilot to avoid "digging" into the atmosphere and overheating of the spacecraft. The software does not control DeltaGlider - it's only an indicator. Temperature calculated roughly from equivalent body heat rate for speed and altitude (Orbiter 2016 does not provide thermal sensors for spacecraft aerodynamic surfaces).

ReentryAP

Description

Reentry AutoPilot (ReentryAP) was the second most important flight control software in the project.

Experience from previous simulations showed that reentry (same as with landing in aviation) is the most important phase of all the flight of winged spacecraft.



Pic. 14: Reentry AP user interface, AP engaged.

Because of its duration (typical reentry on DeltaGlider requires about 30-40 minutes), complex aerodynamics and high stress for the crew, it is very hard to perform clear, safe, and precise reentry. That is why I decided to automate this process using the logic applied on the "Buran" spaceplane.

Control method

Basic method for reentry, applied in all known spaceplanes, built on control of ratio of lift and drag, generated by aerodynamic surfaces of the vehicle. This ratio complexly depends on flight altitude and current angle of attack, and controls heat rate, overload, and (what is the most important parameter) deceleration rate of the vehicle. This last one defines the profile of the trajectory, which itself controls the destination position - final point of the whole reentry.

On US' "Space Shuttle" pilots controlled deceleration rate manually, following pre-calculated tables and keeping the Shuttle within a given "entry corridor". On Soviet "Buran" were used analytical prognosis algorithms, generating optimal trajectory profile in real time, depending on current deceleration rate, distance to destination, side deviation etc.

In my case I chose to combine both methods, trying to achieve simplicity of the algorithms within required precision and safety requirements.

I made a simplified analytical algorithm, allowing to control trajectory profile in a way allows to reach the destination point, and tested it. After a number of simulated flights I set up a boundaries of the "entry corridor", and added them to an algorithm, preventing DeltaGlider digging into the atmosphere or bouncing outside the Karman line. Fine tuning of an algorithm required about 50 test simulations (about 5 work days) to find optimal polynomial for trajectory optimization.

Using current version of algorithm next results were achieved:

- Max overload on reentry: 3.1 G
- Max overload change ratio: 0.2 G/s
- Max temperature on the heat shields: 480° C
- Destination point CEP: 2.5 km

Limitations

Used method, despite it bringing DeltaGlider to the required position, is highly empirical. I can not strictly and in a scientifically correct way explain why I chose operations used in my trajectory control algorithm. They were chosen from practical tests of simulated spaceplane in simulated environment "because this way it is flying better".

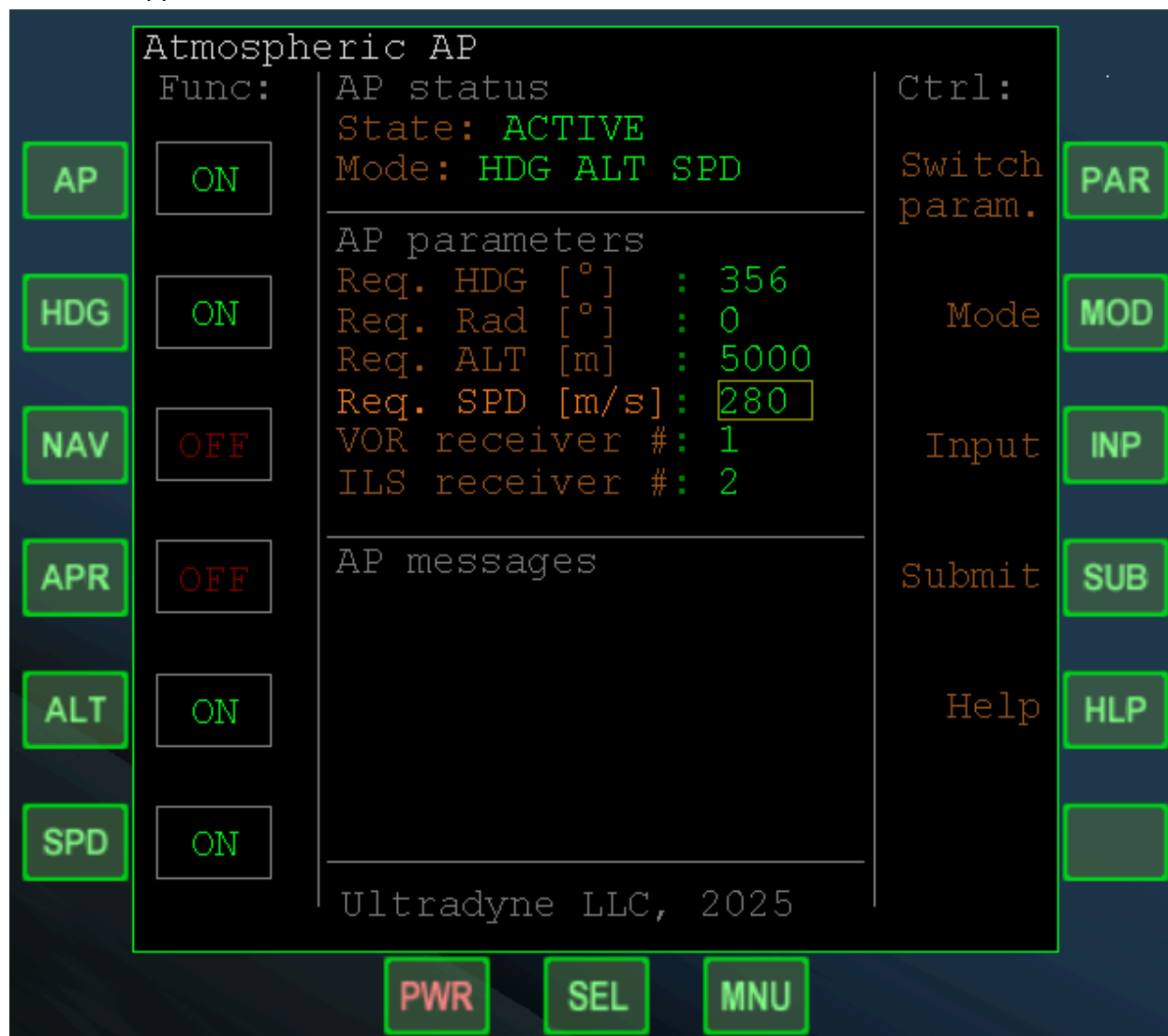
So, if the OrbitalAP is made in a way that it can be (at least partially) transferred into the control system of a real spacecraft, the ReentryAP should not be used in any real or reality related applications without testing by specialists and scientifically correct foundation.

AtmosphericAP

Atmospheric AutoPilot (AtmosphericAP) represents typical autopilot for simple winged aircrafts, for example general aviation or small/medium business jets. It provides flight control on altitudes from 0 to 20 km and in speed range from 0 to 3M.

During development of the autopilot I expected difficulties with flight control on transonic and supersonic speeds. But, fortunately, the aerodynamics of DeltaGlider (or *a way how it is simulated*), allows sustainable control of the flight within all mentioned range speed.

AP provides functions of typical general aviation autopilot, including holding direction and vertical speed, heading control, altitude control and speed control. AP is connected with DeltaGlider simulated radio stack, so it can perform basic radio navigation (using VOR), and can do an ILS approach.

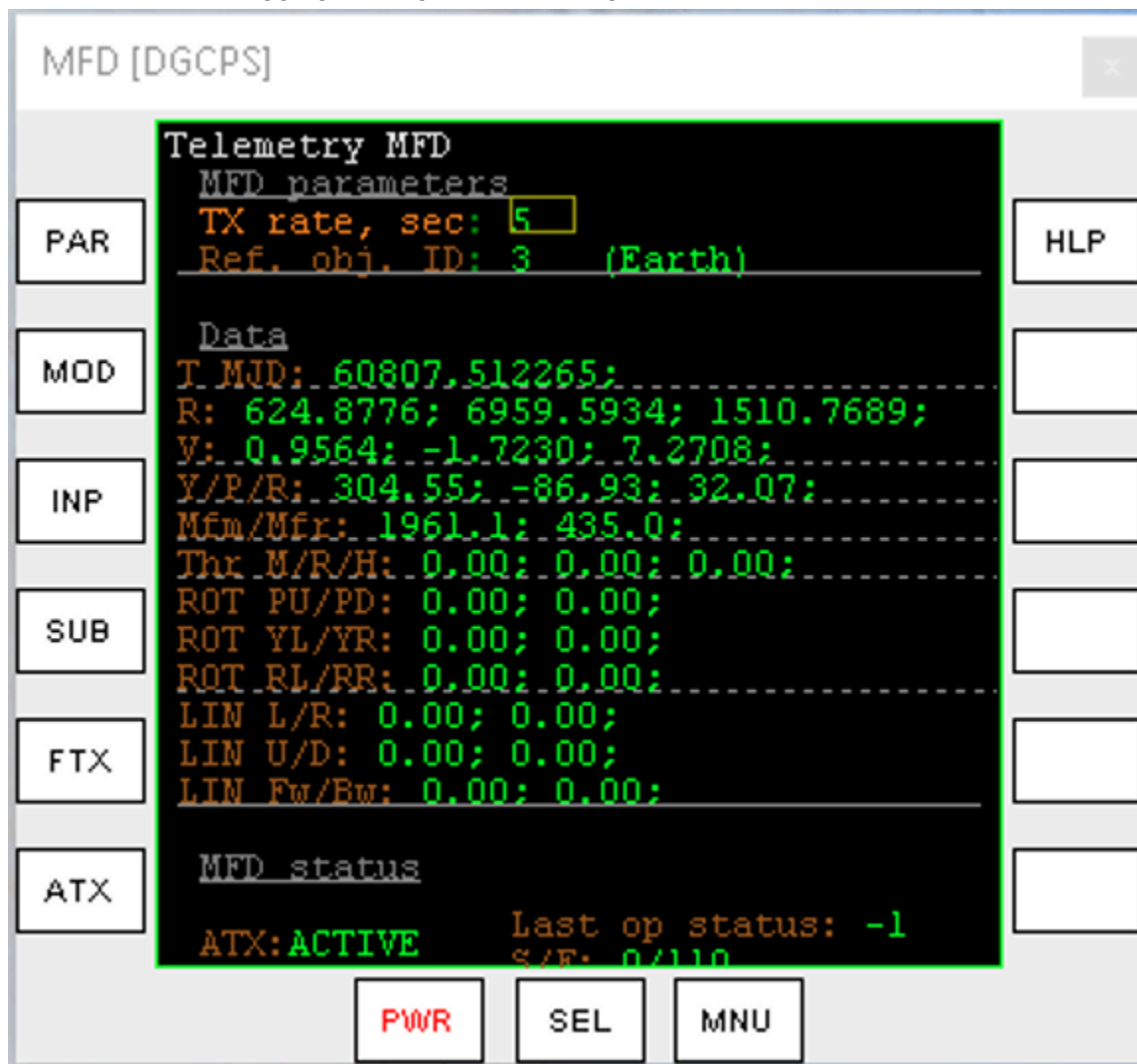


Pic. 15: Atmospheric AP user interface, AP engaged, normal flight performed.

Flight control is performed using PID controllers, so, actually, it's nothing special in this software - it's just an autopilot. With minimal changes it can be used on civilian purposed winged drones.

Telemetry MFD

Last application from the flight control stack is Telemetry. This application allows transmitting vital simulation data "from spacecraft to ground control" - i.e., from simulator to external apps (for logging, saving, data exchange, etc.).



Pic. 16: Telemetry MFD, running in ExternalMFD window.

"Under the hood", this app contains an ISO/OSI-like data exchange framework, I developed a few years ago and used in my previous simulation projects. The framework allows to push the data in a stable and sustained way, with control of errors, between MFD app and external application, using files on disk as transmission environment. In simulation, this system was used for transmitting data from spacecraft to Ground Control (i.e., from simulator to special app which logged our flight data and allows to send it to ChatGPT).

Auxiliary software

DeltaGlider modification LUA script

As I mentioned earlier, to match realism requirements, I needed to change parameters of the DeltaGlider' thruster system. That can be done in two ways - through modification of DeltaGlider by changing C++ sources, or by using an additional LUA script. I prefer the second way, as one requires the shortest time. The only limitations for this method was a need to run this script on each start of the simulation scenario. But the LUA API of Orbiter 2016 allows this.

input

On my move to Thailand I faced a need to leave part of my equipment in Kazakhstan. So, the flight control stick, pedals and throttle, required for simulation, are not available for me for now. To get around this limitation, I used Python for simulation of flight control stick movement with a computer mouse. Result app provides very smooth control, but only for two axes. Still, with flight control software I described above, this is enough.

telemetry_server

During preparation for simulation we faced a need to extract actual flight data from the simulator to log it externally and send it to ChatGPT. So, I commanded ChatGPT to develop a Python-based software, allowing:

- To receive data from Telemetry MFD
- Log received data in .csv file
- Realize web interface, allowing to see telemetry information through a browser on other devices in the local network
- Formulate telemetry package (for sending it to ChatGPT manually)

Using Flask, ChatGPT created a simple, fast and stable application, matching our requirements.

manage_bases

For convenience, in simulation we used add-on Scenic 500, providing for Orbiter 2016 data about 500 real airports, runways of which match DeltaGlider's landing requirements. But it was very inconvenient to operate them inside the simulation, because the simulator did not provide mechanisms for searching the required airbase.

Fortunately, these airbases actually were described in a list of text files with coordinates of bases and other required information, placed into one of Orbiter 2016 subfolders.

So, the solution was obvious: I commanded ChatGPT to develop a simple Python app, sorting bases by names and geographical coordinates and adding into simulation only required bases - especially those which are within required radius around the point of interest. Using basic Python functionalities, ChatGPT successfully wrote the required app.

CesiumJS interface

For graphical representation of designed trajectories a local copy of CesiumJS was used. To pass data from mission planning software, a simple interface was used. Interface was written by ChatGPT, following my commands.

Development methods used

Test-driven development

TDD is a standard technique I am using in all my software projects. But in this case it was relatively hard to follow this method. It was number a reasons for this:

- LUA 5.1.2 has a lack of unit test modules.
- Orbiter 2016 does not allow "to test" separated functionality of an element of an autopilot.
- UI systems (Orbiter 2016 MFD) can not be tested other than manually.

Despite this, all elements of mission planning software, all of calculation modules of flight control software, and all shared modules (PIDs, interface functions, math libraries, etc.) were fully tested (test coverage > 93%).

Someone can say that this is too nerdy. But I know no other way to make a solid code.

Reverse engineering

Orbiter 2016 API is documented, but a number of its methods work in a not so obvious way. Because Orbiter 2016 is old and unpopular software, the Internet could not give me enough information in cases when I faced misunderstanding.

Because of this, I used reverse engineering methods, including testing of API functionality (with a specially made app inside MFD), and rude and straightforward memory scanning (with x64dbg).

Both methods helped me in the state of attitude control development, when I faced troubles with aiming vector conversion.

AI support

As I already mentioned, during this development, I used ChatGPT AI as a co-programmer and support operator.

I already utilized AI support in my previous commercial projects, so usage of ChatGPT wasn't something new in my practice. By my previous experience, I knew that usage of AI can 33% reduce total time costs of a project.

But here that did not work. ChatGPT, as always, fails in topic-specific areas. Same thing I faced here. ChatGPT can offer brilliant solutions, but in most cases it is a machine, representing more common sense than deep knowledge in specific topics. It can create code for you. But it can not understand details of problems. It's sapient, but not like a human.

So, with time, we (together!) got to the optimal way of interaction. I designed ideas and implemented complex algorithms, while ChatGPT implemented standard (mid-level) tasks,

searched and summarized information for me, documented functions, wrote tests, checked my code, etc. And working this way appeared to be very productive.

Main insights from AI usage in this projects are:

- AI can not do your work for you. You are a specialist in a topic, not him.
- Bargain ashore. Express tasks and constraints clearly before work started. Use correct, straight, understandable lexic. To say simpler - formulate tasks like you would do this for human junior programmers.
- Workflow should be transparent. You should express not only a task, but a process you want AI to set up on.

Summarizing, I can say that usage of AI really reduces *total time costs* about 30%. But to achieve this, you should use it on specific tasks. And it would not reduce *the amount of your actual work*.

Final time costs assessment

Actual time spent on this project was 4 months, from which 3.5 months were spent in software development and debugging.

But this estimation does not include my previous work on flight control software - with AtmosphericAP and ReentryAP, same as with the initial implementation of OrbitalAP. Totally, this work required 1 month additionally.

Also, Python implementation of astrodynamics algorithms was made by me much earlier (before spring 2022), so, in case of need of their implementation from scratch, that could require 1 month additionally.

So, in total, this project required about 6 months of work. This is a very short duration, but that was because I was in a hurry, and this software would be used in simulation only.

In case of a real space project I'd estimate the time cost of such work as "1 year+".

Peculiarities of software development for space applications

So, development of a software for space applications have peculiarities, especially:

1. Space software would need much higher quality standards (you wouldn't solve a problem with a patch).
2. For development of such software, the team will need a test environment of some sort (not just unit tests or simple imitation of input, but a fully-fledged test environment).
3. Best option for a test environment would be a software simulation of a controlled process.
4. This test environment must be designed in a way allowing to test software behaviour fast on VERY long time durations.

This last item is extremely important. Typical space operations are long (average time scale - hours to months). Thus, you need to be able to quickly and precisely change the state of your subject of control within this time scale. And it's much more difficult than this looks at first glance.

Mission simulation

Preparation

Legal questions of space debris operations

Right after we chose Envisat as a target for our mission, Ms. Lee, basing on her previous work experience, raised a question about legal norms of such operation. Initially, we assumed that legal rules for operations with derelict satellites would be close or same as in marine law in case of work with abandoned marine vessels.

But after consultations with ChatGPT and studying the "Outer Space Treaty" by UNOOSA, we understood that here we do not have too many similarities. Following the OST of 1967, Article VIII, space debris (even dead, or exploded) still is in jurisdiction and control of the launch state. So, we can not "just fly and deorbit it" - it would be equal to "deorbit a part of the EU"! So, in real life for such an operation we need to have a number of official allowances and papers, confirming safety of the operation. Here presented a basic list of required documents:

1. *Core Mission Documents*
 - 1.1. *Mission Profile Document (MPD)*
 - 1.2. *Concept of Operations (ConOps)*
 - 1.3. *Technical Capabilities Report*
 - 1.4. *Risk Assessment & Safety Analysis*
 - 1.5. *Debris Mitigation Plan*
2. *Legal & Administrative Documents*
 - 2.1. *Ownership Acknowledgment & Waiver Agreement*
 - 2.2. *International Liability & Insurance Statement*
 - 2.3. *Launch Authorization*
 - 2.4. *Spectrum Allocation & Communications License*
 - 2.5. *Telemetry Sharing & Ops Coordination Plan*
3. *Communication & Diplomatic Correspondence*
 - 3.1. *Formal Request Letter to ESA*
 - 3.2. *Memorandum of Understanding (MoU)*
 - 3.3. *UN Notification Draft*
4. *Supporting Materials*
 - 4.1. *Spacecraft Design Diagrams & Renders*
 - 4.2. *Ground Control & Comms Plan*
 - 4.3. *Budget & Funding Source Declaration*

Looks a lot, and this really is. But this is proof that space is not only about speed, vacuum and hydrazine, but also about tons of paperwork, and you need to get this in count during planning space projects and missions.

During the mission we possibly needed to do an EVA on Envisat. As we understood, on board the satellite still were about 100 kg of hydrazine-based propellant compound. So, based on mine and Ms. Lee's previous experience, we needed to have a licenses for work with

hydrazine-contaminated hardware. We checked the legal requirements for this. Following ChatGPT-suggested data, in real life for such operation we may require:

1. *Hazardous Material Handling License*
2. *Environmental & Occupational Safety Certification*
3. *Toxic Propellant Handling License (Aerospace-specific)*
4. *Space Operations Safety Certificate*

Additionally, these licenses must be provided by legal organisations of the European Union. In case we have got these licenses somewhere outside the EU, we should additionally have an *International Recognition / Mutual Recognition Agreement (MRA)*.

So, this brief study about international space law was in the end very useful. It clearly showed that sometimes in space missions you can face difficulties not only in outer space, but in the bureaucracy offices.

Envisat engineering analysis

After revealing legal questions, we continued to engineering analysis of Envisat, using published papers and studies as sources of information.

Following these papers, we slowly understood what we faced - a titanium box size and mass of a school bus, with a huge "sail" of solar array, and belly spiked with antennas and scientific equipment. A dead high-tech dragon with a scale of unnatural metal and poisonous flammable blood, silently arrowing through cold emptiness high above Earth.

Because the objectives of the project and available simulated tools implied manned flight to Envisat, our first question was - how dead is this dragon? If the satellite still has any active EM emitters (like radar altimeter or any scientific equipment) their radiation can hurt crew or equipment. Additionally, we had questions about the status of Envisat thruster systems and fuel storage. If they still had propellant inside that must be counted in the mission plan. Following my knowledge of such systems, there definitely was. But how much?

Following the official documents and articles on ESA official site and in Wikipedia, we determined that:

1. Officially published reason for Envisat malfunction (power regulator failure or short circuit on one of power buses, triggered switching the whole system to safe mode) seems to be most plausible.
2. Since official declaration of the end of its mission (May 9, 2012) Envisat is officially treated by ESA as "completely unresponsive", which means that:
 - 2.1. Envisat does not answer to telecommands and does not send any telemetry
 - 2.2. No active EM emitters on board are functional
 - 2.3. No power cycling or data processing performs
3. At launch Envisat had in tanks 314 kg of hydrazine-based propellant, from which till June 19-23 2006 129 kg was consumed. Following the papers about orbital correction strategy on Envisat mission we may expect that at the moment of the end of the mission there was about 10% (30 kg) of hydrazine. The most nasty thing was that as always with liquid fuel spent, there was 4% uncertainty - meaning that more than 10 kilograms of aggressive, corrosive, poisonous liquid were left somewhere in pipes, fittings, turbopumps and valves.

That definitely must be counted, especially in safety guidelines for EVA.

Summarizing all above, work with this satellite was possible, but required precautions, because it still contained (and, possibly, contaminated with) hazardous chemicals.

Envisat motion analysis

After assessment of engineering troubles, we continued to analyze the current state of Envisat. Following the official documents and TLE data, Envisat occupied "high LEO" orbit with altitude 773 km and very low eccentricity. Following the papers about a mission, its orbit was a true example of high astrodynamical art. Highly inclined (98.4°), with period 100.16 minutes and repeat interval in 35 days, it was maintained with a complex but extremely fuel-efficient algorithm and designed in the most optimal way for Envisat mission objectives.

However, after Envisat malfunction, it appeared to be an obstacle for natural and safe deorbiting of the satellite. At this altitude, the typical decay time for a satellite is about 150 years. Moreover, this altitude is almost on the lower edge of Inner Van Allen belt - high radiation zone, formed in altitudes 1000 km - 6000 km by interaction of Earth's geomagnetic field and solar wind. This is still not a place of high radiation, but work in this region requires strict time constraints.

In addition to this, studying the papers about Envisat brought to us information that the satellite doesn't lay still on its orbit. Following the papers, after losing contact in 2012, Envisat kept its orientation for a while, but about the year 2016-2017 it started to rotate with unstable speed, reaching 3 degrees per second. Probably, something was broken inside the satellite - like orientation flywheels - broken, jammed and transferred its momentum to a satellite body. But with time, under the effects of tidal forces, geomagnetism and solar wind - the rotation started to calm down. The question was - how fast is Envisat's rotation now, at 2025?

Using ChatGPT, we create a script for calculation, extrapolate rotation speed in time. Using it, we calculate the angular velocity of the satellite as

$$\omega_{xyz} = -1.782 \ 0.016 \ -0.786 \text{ deg/s.}$$

So the satellite rotated roughly with speed about 2 degrees per second, in two planes simultaneously. That was not so good news. But I remembered a similar operation, made in 1985 by soviet cosmonauts Dzhanibekov and Savinykh. On spacecraft "Soyuz-T13" they approached and docked abandoned and damaged space station "Salyut-7", rotating almost the same way.

Unlike the crew of "Soyuz-T13" we don't need to dock to the satellite. The question was - is it possible to get close to Envisat using a Manned Maneuvering Unit (MMU)? We built a short test scenario for Orbiter 2016, just to look, how fast the rotation of Envisat is, and is it possible to work with such a thing? On the screen, slowly rotating Envisat didn't look dangerous - if not to think that thing was the size of a truck. Ms. Apalkova, looking at the screen, presumed that in simulation she can reach the Envisat with MMU and attach herself to it. After a short conversation, she and Ms. Lee persuaded me that this is possible. I and ChatGPT accepted their opinion.

Envisat deorbiting risks assessment

In the section about studying space laws, I already mentioned the requirement to provide "Risk Assessment & Safety Analysis" for such missions we planned. Initially I planned to omit this step, leaving it in a draft of mission planning documents. But when we published news about our simulation in LinkedIn, I received very important feedback from Mr. Christophe Bonnal. Following his opinion, non-controlled deorbiting of such massive objects as Envisat may cause casualties on the surface. So, if we are trying to follow a high level of realism, we need to count this in our project. Additionally, Mr. Bonnal kindly provided links to Re-entry Safety Requirements, used in the space industry.

Briefly, main requirements from this documents were next:

1. Reentering objects must not pose risk to human life, cause environmental contamination, or damage infrastructure.
2. Surviving fragments with kinetic energy >15 J are considered hazardous.
3. Toxic and radioactive materials must not reach Earth, or must be neutralized upon impact.
4. The total acceptable risk of human casualty is limited to 1 in 10,000 (10^{-4}).
5. As a practical rule: any satellite >500 kg requires controlled reentry unless designed to fully demise.

Importance of these requirements was obvious, so I decided to follow them in our simulation project.

Envisat deorbiting method development

Drift Augmentation Device

Taking in count all requirements mentioned above, we proceeded to develop a method to deorbit Envisat. As a first step, we commanded ChatGPT to summarize for us early developed methods of big satellites deorbiting (as a list of methods with description and strengths and weaknesses of each).

His answer slightly disappointed us. The only successful examples of deorbiting large and non-cooperative satellites existed in 2000, when NASA deorbited Compton Gamma Ray Observatory (CGRO) using Shuttle. But that was NASA and that was Shuttle. Most other options were very theoretical and included very conceptual, but purely not practical ideas like electrodynamic tether, ion beam shepherding or laser push. Looked nice for sci-fi, but we tried to do some real things.

The only realistic options were a combination of robotic arm and propulsion module, or some sort of drag augmentation device.

After a short analysis I declined the idea with the propulsion module. It looked most realistic, but it would require hard docking with Envisat, which doesn't look healthy for me because:

1. Envisat had a "rear support ring", through which it was attached to LV, but it was not designed for docking, and access to it would be inconvenient because of a big solar array.

2. Also, Envisat had no attachment points through which it could be grabbed by any type of robotic arm.
3. Getting in count that internal structures of Envisat, after 23 years in orbit, could be degraded, and satellite can have internal damage after failure of orientation flywheels, any "hard" interactions, including pushes from touching of docking mechanisms or robotic arm grabbing, would be strongly unwanted.
4. Last but not least, we had not too many options where to attach any type of propulsion module or robotic arm on DeltaGlider. It was possible to install it after launch (in space), that also seems not an optimal solution - too complex, too long, and too unreliable.

Drag Augmentation Device (DAD) of some sort, from the other side, looked to us as a more attractive solution. It could be cheap and easy to produce, easily transportable and relatively easy to deploy. Before this project, I already calculated atmospheric drag affection on satellite decay, so it was possible for me to calculate parameters of this device with enough precision. In the end, that was a gracious solution! So, we decided to study this option deeper.

DAD concept

Following a simplified model for studies of satellite decay, we assumed DAD as a very light sphere, made from some sort of polymer, and inflated. As is known, such satellites (like "Echo 1" and alike) were produced and successfully worked earlier.

After simple calculations in Python, Anna Apalkova with ChatGPT found that the **optimal size for such a sphere, able to deorbit mass equal to the one of Envisat, in 1 year, would be not so big - about 5.2 meters in diameter**. Made with Mylar, aluminized for thermal balance and double-layered with tension ribs to resist micrometeoroid hits, it could be packed in a transport container with size less than 0.1 m³ (size of a hiking backpack), with weight about 10 kg. Inflated with AIBN (azobisisobutyronitrile) from a small cartridge, it can increase Envisat drag and relatively safely deorbit it.

DAD attachment method

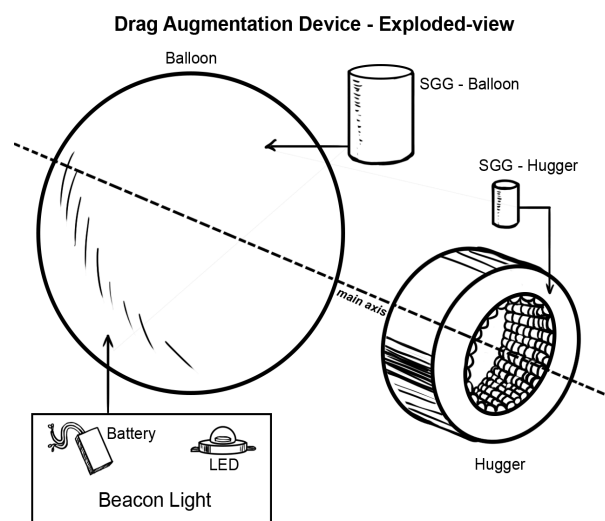
Next question was - how to attach this sphere to Envisat? For a time we gathered information about different glues and nylon screeds. But after this I came up with a much better idea!

To avoid all that additional elements, we can use the pressure from the inflatable balloon itself to create a kind of self-tightening "sleeve" or collar that gently grips the satellite. That way, we don't need a complex robotic arm or hard mechanical clamp — the inflatable structure does double duty.

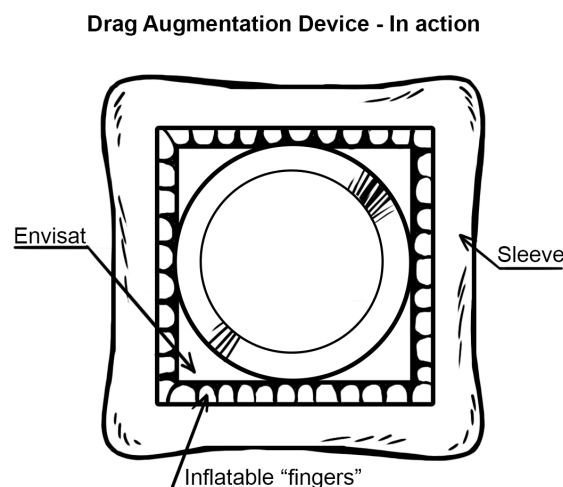
In this case:

- The DAD is a large spherical balloon.
- It's connected via a short cylindrical "sleeve" or cuff — also inflatable — that wraps around one end of Envisat.
- The sleeve inflates first, expanding around a segment of Envisat's body.
- As pressure builds, the sleeve gently conforms to the satellite's shape and grips it with distributed pressure — similar to a blood pressure cuff or an inflatable cast.
- Then the balloon inflates behind it.

Simple and elegant. We pull on the "sleeve" (or "Hugger", as ChatGPT correctly named it) on the rear end of Envisat and inflate it - and it tightly holds the satellite. By work logic it repeats biological systems, such as intestinal villi, vaginal folds or octopus suckers. Made with reinforced Mylar or Vectran, with inner folds of combination of soft polymer and fabric, it would hold Envisat gently but firmly, inflating around Envisat's irregular body and providing gentle but firm multi-point contact.



Pic. 17 (concept sketch by Anna Apalkova)



Pic. 18 (concept sketch by Anna Apalkova)

Beacon and Passive Emergency Reentry Control (PERC)

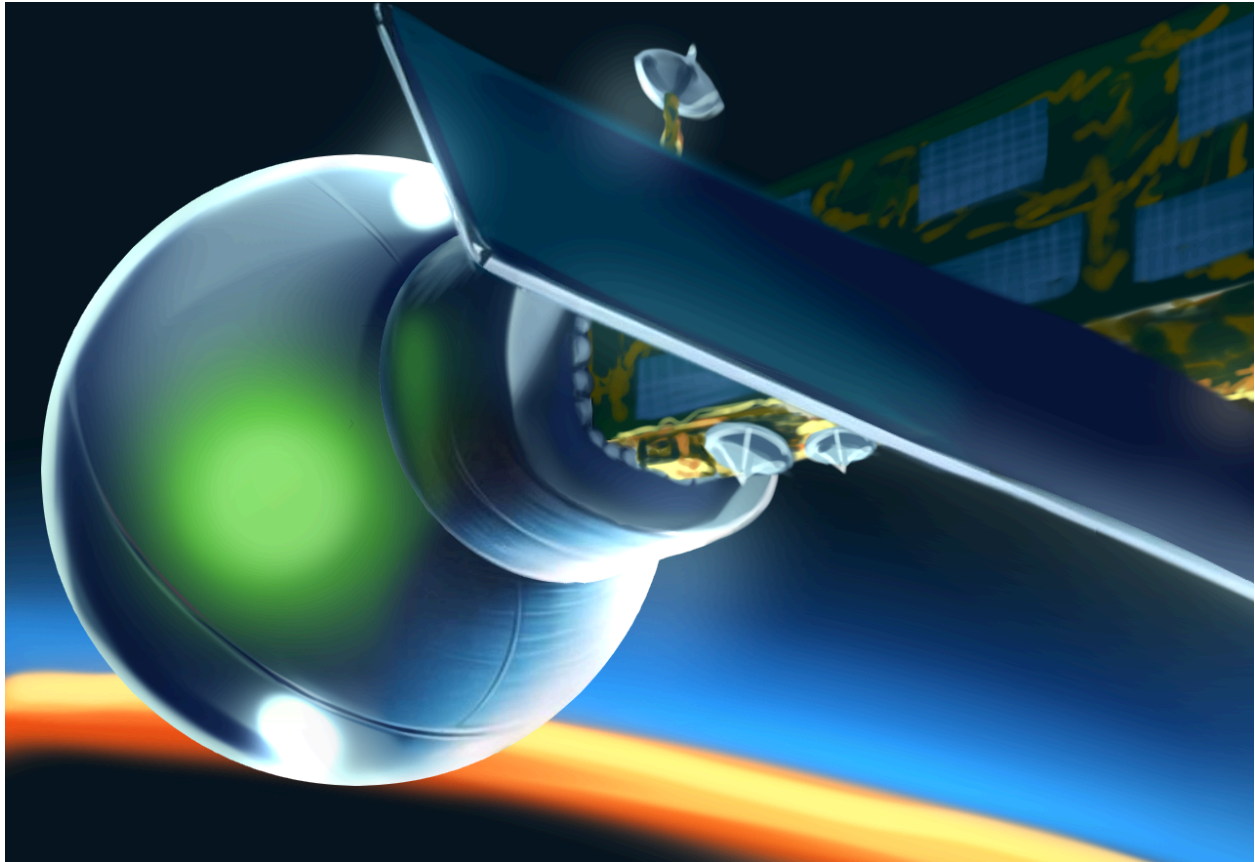
To provide the ability of precise tracking of Envisat with attached DAD, we included into the concept LED-based beacon installed inside the balloon and emitting green light. To be visible from Earth, the beacon would require about 0.5 W of power. So it can be powered with a common Li-Ion battery, charged with energy for all time of active working (about a year).

To enhance risk mitigation abilities of DAD, we also added in the design of DAD the PERC - Passive Emergency Reentry trajectory Control mechanism.

It consists of a nichrome wire or pyro-cord loop integrated around the base of the balloon. In the final orbits, if reentry is predicted to occur over a populated region, Ground Control can remotely trigger a burn-off, detaching the balloon from Envisat and reducing drag. This adjustment may shift the reentry footprint to a safer area like the South Pacific Ocean Uninhabited Area (SPOUA). While not a substitute for full guidance, this adds a limited layer of control — and fits within the low-cost, semi-passive philosophy of our concept.

Production duration and cost estimation

To get proof of concept (and to estimate difficulty and possible duration of production) we sent inquiries to commercial aerospace companies, oriented on work with inflatable structures. Of course, we clearly mentioned that this is an educational and simulation-based space project. Unfortunately, nobody answered us.



Pic. 19: DAD, mounted with Hugger on Envisat, artistic representation (art by Anna Apalkova)

DAD model for Orbiter 2016

To add DAD into simulation, I made a 3D-model of it, using my skills from previous experience (game development), and then developed a basic add-on for Orbiter 2016, implementing (with LUA) DAD parameters mentioned above. With this, this part of preparation was finished.

DAD tech specs

Diameter:	~5.2 m.
Mass (with "Hugger", beacon and battery):	~20 kg.
Volume (packed):	~0.1 m ³ .
Material:	Balloon - aluminized mylar, "Hugger" - mylar + PU-coated fabric, SGG - AIBN-based
Cross-section area:	~ 20 m ²
Drag force (inflated)	~7 N

EVA planning

Safety rules

During the planning of the EVA for deployment of the DAD, we started work from development of the safety rules. With help of ChatGPT and involvement all participants of the simulation, we developed list of rules, including next items:

1. Chemical Safety Guidelines
2. Radiation Safety Guidelines
3. Structural and Mechanical Safety Guidelines
4. General EVA Procedure Rules
5. Emergency Behavior Summary

Additionally, safety rules included EVA preparation- and post-EVA checklists.

Attachment methods

Additional problem for EVA was lack of railing or any attachment points on Envisat. The satellite was never designed as maintainable. Initial solution was offered by ChatGPT, who offered to use Portable Foot Restraints (PFRs), attachable to Envisat body with portable adaptable brackets or clamps. Ms. Apalkova approved this idea as useful on EVA, but I was still unsure that that would work. As an additional method of attaching, I offered to use thermal adhesive anchors - something like a hot glue, pouring on the surface of the satellite and then hardens, making a strong anchor.

This idea required additional studies. Using ChatGPT we found examples of such plastics (they actually existed and even been tested in space by NASA). But the possibility of hardening was determined by the surface temperature of the satellite. So, we roughly assessed the temperature of Envisat, from time of its exposure to the sunlight per orbit turn and materials of the satellite.

Estimated temperature of the surface was not exceeded 61° C, so as an adhesive polymer we chose Low-Temperature EVA (Ethylene Vinyl Acetate). This material was tested in space by NASA, worked well and matched our requirements of safety and useability.

EVA time estimation

By opinion of Ms. Apalkova and Ms. Lee, overall EVA time should not exceed 2 hours. From this time:

1. 20 minutes - for exiting DeltaGlider airlock, attachment safety tethering, connecting MMU and locking DAD package to spacesuit attachment system.
2. 10 minutes - for final approach to Envisat
3. 15 minutes - for attachment to Envisat
4. 45 minutes - DAD deployment
5. 10 minutes - returning to DeltaGlider
6. 20 minutes - disconnecting MMU, decontamination procedures, packing safety systems

Their estimation was clearly within safety limits, but my experience told me that we need to add time for different extra cases. Altogether, we considered that EVA should take 3 hours.

Mission planning

Key constraints

Before start the planning, I wrote down a list of most important constraints for the mission:

1. Total required amount of delta-v must not exceed 2.49 km/s (by construction limits of DeltaGlider).
2. Total time on LEO $h=773$ km must not exceed 12 hours (by radiation safety limits).
3. EVA must start within 5 hours after launch and end within 8 hours after launch (by psychophysiological limits).
4. Total flight time must be as short as possible (by psychophysiological limits)

Of course, during mission planning any third-party software (like GMAT, Polastro, etc.) must not be used.

Launch parameters calculation

In this mission, the launch window determined not only launch time, but also time of Envisat intercept. My first goal was to find the most optimal date for launch, when intercept was possible in the shortest time after launch. As an extra condition, the flight could not being done at Tuesday and Thursday (when I and Ms. Apalkova were at Thai Language Classes at Chiang Mai University).

From TLE, for epoch (UTC) 09 May 2025, 21:14:39.5, orbit of Envisat had next parameters:

- Semi-Major Axis (a): 7132.2457 km
- Eccentricity (e): 0.000906
- Inclination (i): 98.334°
- Right Ascension of Ascending Node (Ω): 86.540°
- Argument of Perigee (ω): 163.776°
- True Anomaly (v): 258.422°

We need to choose a launch window when the SRC Polygon launch position will lay in the orbital plane of mentioned orbit, estimate the probable position of DeltaGlider after separation from Energia LV, and from this state build an intercept with less delta-v budget possible.

Using Python and my mission planning software, I plotted these maneuvers for dates from May 9, 2025, to June 8, 2025, and optimal configuration appeared to be May 12, 2025, with a launch window at 07:30:40 UTC. Because repeat of Envisat orbital configuration was expected only in 35 days, and we already were in this project for more than 3 months, we decided to fly at 12.05.2025.

Orbit parameters and maneuvers

Using my mission planning software, I designed the mission with next steps:

Step 1: Start at Initial LEO.

Time: 12 May 2025, 07:39:22.8 UTC

Orbit parameters:

- a: 6681.4777 km
- e: 0.0005
- i: 98.360°
- Ω : 84.119°
- ω : 21.803°
- v: 360.0°

This orbit must be achieved after injection with LV into a co-planar, lower circular orbit, designed for phasing and Lambert intercept with Envisat. Notice that Ω is not the same with one of Envisat's orbit, but this is normal - this orbit altitude is lower, its period is shorter. So, of course, it will have shifted Ω for the same orbital plane. True anomaly taken as 360.0°, showing that after decoupling with LV we're actually in the periapsis of our own orbit.

Step 2: Intercept Transfer

- Transfer Burn:
 - Time: 12 May 2025, 10:10:38.8 UTC
 - dv: 0.2709 km/s
- Arrival Burn:
 - Time: 12 May 2025, 11:02:37.8 UTC
 - dv: 0.1861 km/s
- Total dv: 0.4570 km/s
- Transfer time: ~3120 s (~52 min)

These values, of course, were approximate, and should be used as "optimal" reference values. **Actual values should be calculated with OrbitalAP** within required time constraints and delta-v tolerance, which I presumed to be about 15% bigger than given here. Because of the same reason, **all delta-v vectors are expected to be precisely plotted by OrbitalAP**, so, I am not presenting here results achieved by mission planning software.

During this transfer, EVA specialists should prepare for EVA, to start it right after final approach to Envisat

Step 3: Reached rendezvous orbit

Time: 12 May 2025, 11:02:37.8 UTC

Orbit parameters:

- a: 7132.2457 km
- e: 0.000906
- i: 98.334°
- Ω : 86.540°
- ω : 163.776°
- v: 299.457°

Step 4: Final approach

Time: 12 May 2025, 11:02:37.8 UTC

Duration: 1800 s (30 min)

At this step we would need to approach Envisat to a distance of about 50 m. This phase will require a manual approach using RCS. Estimated delta-v spent (from RCS tank) would be about 50 m/s.

During this phase, EVA specialists should get into suits to be ready to start EVA.

Step 5: EVA

Time: 12 May 2025, 11:32:37.8 UTC

Duration: 10800 s (3 h)

Main step of the mission - EVA for attaching and deploying DAD onto Envisat. Estimated delta-v spent (from RCS tank) - 10 m/s

Step 6: Holding phase

Time: 12 May 2025, 14:32:37.8 UTC

Post-EVA operations. Control of DAD inflation, checking beacon signals, slow fallback from the target. Estimated delta-v spent (from RCS tank) - 25 m/s.

Step 7: Return transfer

- Retreat Burn:
 - Time: 12 May 2025, 17:43:36.8 UTC
 - dv: 0.4793 km/s
- Parking LEO Insertion Burn:
 - Time: 12 May 2025, 18:39:30.8 UTC
 - dv: 0.4093 km/s
- Total dv: 0.8886 km/s
- Transfer time: ~3360 s (~56 min)

This transfer would move DeltaGlider to a parking orbit with altitude about 250 km, from where reentry interface, optimal for algorithms of ReentryAP, could be successfully achieved. Same as before, these values, of course, were approximate, and should be used as "optimal" reference values. Delta-v spent here was not optimal, because we need to reach the required state (need to appear on parking LEO at half of turn - by position and by time - before deorbit burn).

Step 8: At the parking LEO

Time: 12 May 2025, 18:39:30.8 UTC

Orbit parameters:

- a: 6628.1265 km
- e: 0.00025
- i: 98.326°
- Ω : 86.145°
- ω : 247.863°
- v: 110.858°

Step 9: Deorbiting and reentry

- Deorbit Burn:

- Time: 12 May 2025, 18:52:52.7
- dv: 0.0749 km/s
- Expected Reentry Interface Altitude: ~120 km

At this step, DeltaGlider should be moved to suborbital trajectory, and then, under control of ReentryAP, follow towards a given landing site.

Step 10: Landing

Time: 12 May 2025, 19:40 - 19:50 UTC

Landing Zone:

Primary: SRC Polygon (Sumatra);

Secondary: Chiang Mai or Singapore contingency corridors.

Mission summary

Following presumption about 15% tolerance for maneuvers calculated "in place" with OrbitalAP, **total delta-v budget** estimated as 1.4205 - 1.6223 km/s, which was well inside required constraints.

Time in Envisat orbit was estimated as ~6 h 40 min, which was within required constraints.

Total simulation time estimated was about 12.5 h, which was also OK.

Orbit determination, navigation, communications

Despite in Orbiter 2016 navigational data transferring to simulated sensors directly from simulation core, following realism requirements we especially chose orbit determination facilities, navigation sources and communication channels.

For orbit determination we planned to use data from:

- SRC Polygon Ground Control and Space Surveillance Facility
- NARIT observatory (Chiang Mai, Thailand)
- Tien Shan Astronomical Observatory (Kazakhstan)
- Svalbard Satellite Station (Norway)
- Public NORAD data sources (celestrack.org etc.)

As a navigation data sources, we planned to use:

- DeltaGlider onboard navigation complex, including IMU, astro- and radio-navigational systems.
- GPS data (orbital altitude allows us to use it)
- Data from orbit determination ground facilities

For communication, we planned to use:

- Starlink network
- TDRSS (U.S. Tracking and Data Relay Satellite System)
- Basic FM transceivers from DeltaGlider radio stack (as a backup communication and telemetry channels)
- SATNOGS network (as a backup telemetry channel)

Emergency situations

Using ChatGPT, we developed number of short instructions for emergency cases, including checklists for:

- Launch vehicle malfunction (DeltaGlider TWR and delta-v allowed to escape the LV on any stage of launch)
- DeltaGlider hardware malfunctions (depressurisation, fuel leakage, thruster malfunctions, aerodynamic surfaces problems)
- EVA emergencies (suit damage, chemical contamination, MMU malfunction)

Additionally, data from development log, including reentry corridor parameters, could be used as instruction for reentry in case of ReentryAP malfunction.

In cases not included in emergency checklists, we agreed to adhere to the following policy:

1. If we have enough time, try to find a solution for a problem via consulting with crew and Ground Control.
2. In other cases, act in a way less compromising the safety of crew and equipment.

Flight

Briefing

In early 2025, a critical alert was issued by the global space surveillance network: the long-defunct ENVI-7 (Envisat) satellite had a close approach with a piece of space debris, raising the probability of generation of Kessler syndrome to dangerous values. Once a flagship Earth-observation platform, Envisat's massive structure now posed a threat to multiple active orbital assets, including those operated by ESA and ISRO. International commission formed by space agencies of European and Asian countries decided to develop and perform a space mission, designed to reduce risks caused by Envisat. This mission needed to be developed in the shortest time possible. With major launch providers unable to respond in time due to preexisting commitments and political reasons, a Southeast Asia-based initiative led by Ultradyn stepped in.

Operating from the SRC Polygon facility in Sumatra and leveraging heritage platforms such as the Energia LV and the high-agility DeltaGlider spaceplane, Ultradyn initiated "Operation Silent Giant" - a time-critical interception mission aimed at attaching a beacon for long-term tracking and deploying a drag augmentation device to begin controlled deorbiting of the 8-ton satellite. These are the only viable responses identified through engineering review, as orbital repair or reactivation of Envisat looked unfeasible due to its structural, electrical, and propellant-related hazards.

After intensive preparation, including consulting with topmost astrodynamics and space debris experts, engineering studies of Envisat design, development of effective Drag Augmentation Device (DAD), improvements of guidance software, mission planning and EVA training, mission was scheduled at May 12, 2025.

Mission objectives

1. Using Energia LV and DeltaGlider spaceplane, reach Initial LEO.
2. Intercept Envisat with 2-impulse transfer.
3. Perform close approach with Envisat
4. Perform EVA and deploy Drag Augmentation Device with tracking beacon on Envisat.
5. With 2-impulse transfer, retreat to Parking LEO.
6. Perform reentry and land planned landing site.

Mission crew

The crew of the mission were:

- **Daniil Galakhov** - Mission Commander, Chief Astrogator, DeltaGlider Pilot.
- **Anna Apalkova** - EVA Specialist 1, Life Support Systems Lead.
- **Rebecca W. Lee** - EVA Specialist 2, Technical Specialist, Operations Support Advisor.

Ground control and mission support was provided by:

- **ChatGPT AI** - Virtual Ground Control and Crew Assistance Specialist, Spaceflight Psychology Expert.

Launch

Energia LV, carrying DeltaGlider, was launched from SRC Polygon at 07:30:40 UTC on May 12, 2025.

First Obstacles

After reaching the orbit, at T+445 s., DeltaGlider was separated from the Energia booster stage, and the crew proceeded to spacecraft systems checkout and procedure of orbit circularizing.

At 07:53:49 Circularisation Burn was applied, and DeltaGlider reached circular LEO with SMA=6735.3780 km, Ecc=0.000540, Inc=98.495° and RAAN=85.040°, which was relatively close to planned orbital parameters.

Since that moment, the crew has been proceeding with preparations to orbital transfer towards Envisat. But at 08:18:32 crew and Ground Control revealed that guidance software contained errors, preventing plotting intercept maneuver in the planned time range. During development, an input field for initial time for optimal transfer search was not added to software, so OrbitalAP provided results outside planned timings and delta-v range. Despite this, orbital autopilot software allowed the crew to plot a new transfer maneuver outside planned timings, which was done at ~08:45.

At that point, crew and Ground Control faced troubles with telemetry. Telemetry transmission software had not been able to refresh telemetry data working in the background (while MFD executed other apps). Telemetry packages were transmitted correctly, but data in them was not actual.

But at that moment neither the crew nor ground control didn't notice this.

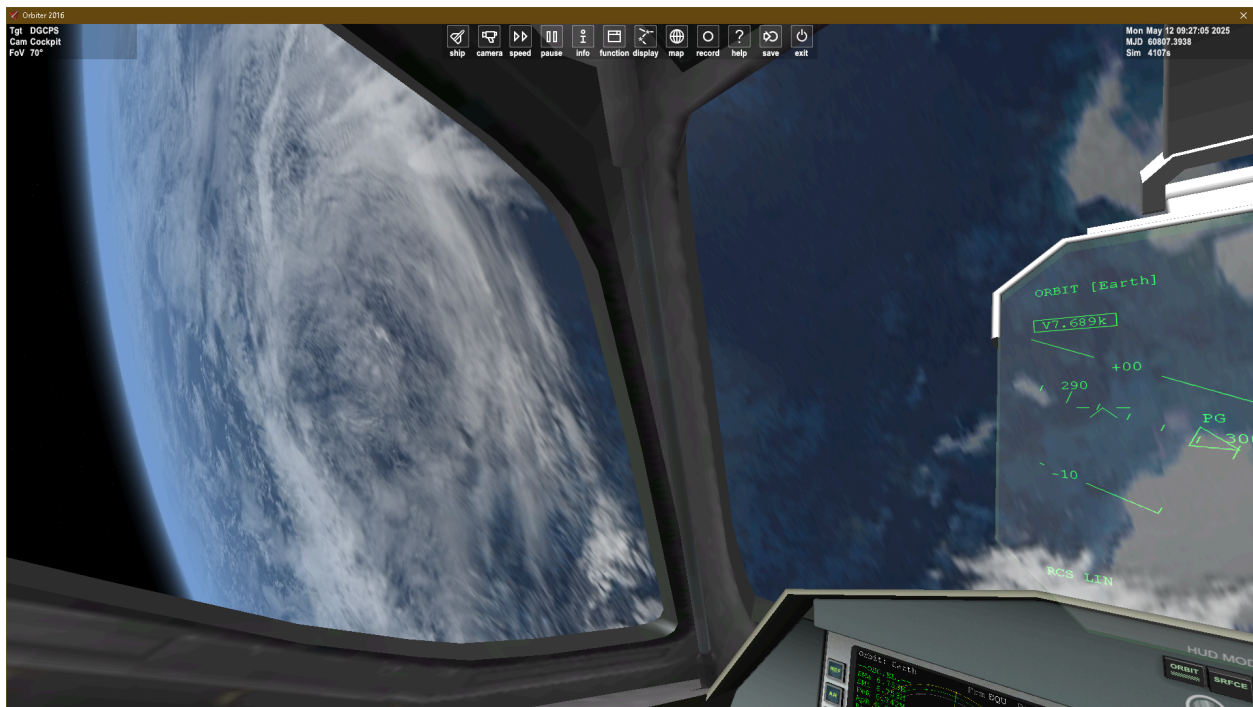
With guidance software pair of transfer maneuvers was planned:

Maneuver	UTC (approx)	Δv [km/s]
Transfer Burn	12.05.2025 10:31:14	0.555
Arrival Burn	12.05.2025 11:23:35	0.438

This way, the intercept had to be performed 20 minutes later than by the initial mission plan. Because of this, close approach phase and EVA durations had to be shortened, and the total amount of delta-v required for transfer grew up to 0.993 km/s - almost twice larger than planned. But these values were still within the range of the mission plan and delta-v budget of DeltaGlider. At ~08:50 orbital autopilot was engaged, and DeltaGlider continued cruise towards intercept insertion point.

6D PID-based orbit correction algorithm kept deviation from planned orbit less than 10 km by range and less than 10 m/s by velocity, so crew proceeded to meal, keeping moral state high by watching recorded YouTube videos of Geoscan company Head of Small Spacecraft Projects and space blogger Alexander Khokhlov and Russian astronomer Vladimir Surdin.

At ~09:25 DeltaGlider reached position over the north polar area of Earth. Orbital drift started rising - slowly but unstoppable. Drift correction system kept DeltaGlider on track, but the spend of delta-v drastically rose, reaching maximum over the Hudson Gulf and Central Canada.



Pic. 20: View from virtual cockpit, polar cloud vortex visible through cockpit window.

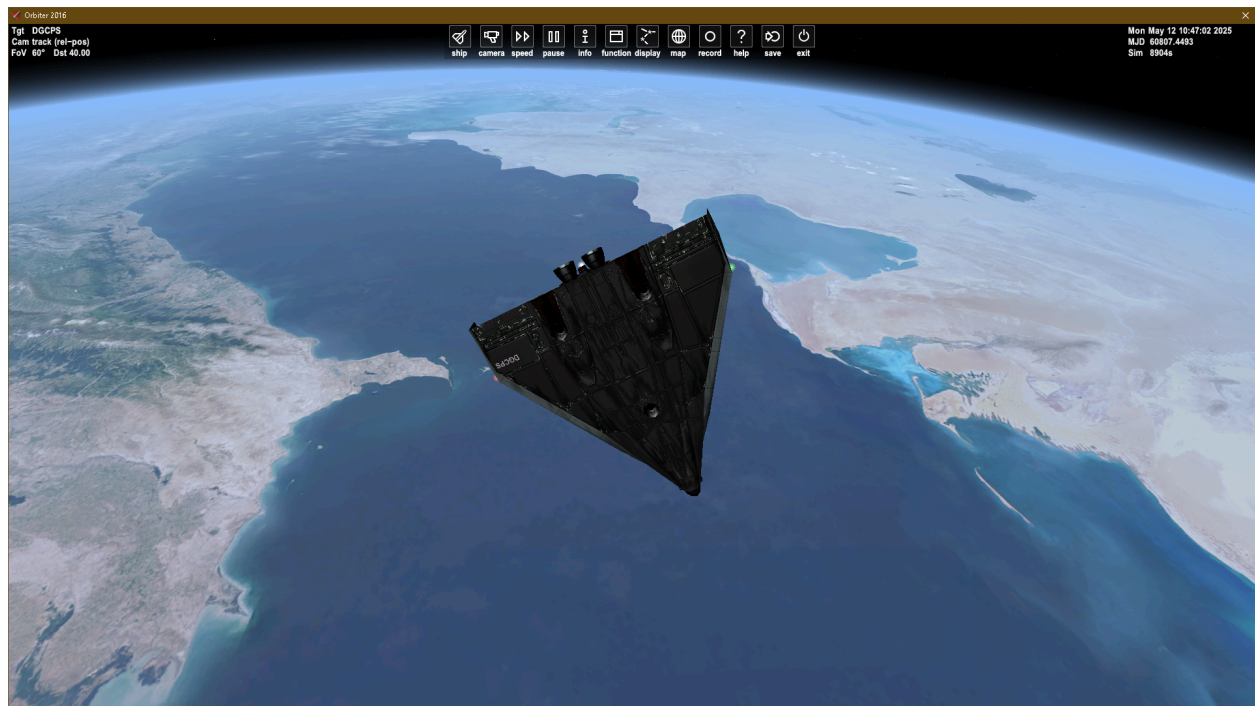
This effect caused lively conversation between crew and Ground Control. Mission Commander supposed that the spacecraft faced an effect from mascon, but Ground Control stated that that drift was caused by a change in density of the upper atmosphere over the terminator zone.

At ~10:00 orbital drift values were restored within green levels. But later at ~10:25 (300 sec. before Transfer Burn) drift raised again up to 10 km by range and 23 m/s by velocity. That

moment DeltaGlider passed terminator over the southern polar area of Earth. So, the opinion of Ground Control was proved - (within this simulation model) at LEO drift of winged spacecraft could be caused by a change in density of environment.

At 10:31:14 Transfer Burn was applied, and DeltaGlider passed to intercept trajectory. Lambert solver of guidance software plotted transfer "on a knife edge" with periapsis at altitude ~213 km. By the opinion of Mission Commander and Ground Control, this was within safety parameters. DeltaGlider had slide downwards, towards the edge of the atmosphere.

Spacecraft passed periapsis over the north part of the Caspian Sea and delta of the Volga river. Drift raised again. It increased even more at the terminator pass over the polar region. Correction system reduced it, but not totally, and with high spent of delta-v. After short discussion with crew and Ground Control, Mission Commander decided not to perform any actions, allowing automatics to do its job.



Pic. 21: View from external virtual camera, DeltaGlider passing over Caspian Sea.

At 11:23:35 Arrival Burn was automatically applied. Drift reached 19 km by range and 67 m/s by velocity. After calculating relative position of spacecraft using external navigation data sources and on-board radar system, **Envisat was detected at range 280 km, with relative speed less than 2.0 m/s.**

Miss or hit?

That put the crew in a troublesome situation, with raising questions about trust to guidance software and possibility of mission success. That was a hard impact. But the crew didn't retreat.

After short period of observation Mission Commander noticed, that:

1. Range between DeltaGlider and Envisat decreases with time.
2. Same time, relative speed grows.

3. The only difference in orbital parameters was TrA, other parameters matched almost perfectly.
4. Orbits of both spacecraft, plotted on MFD, looked crossing.

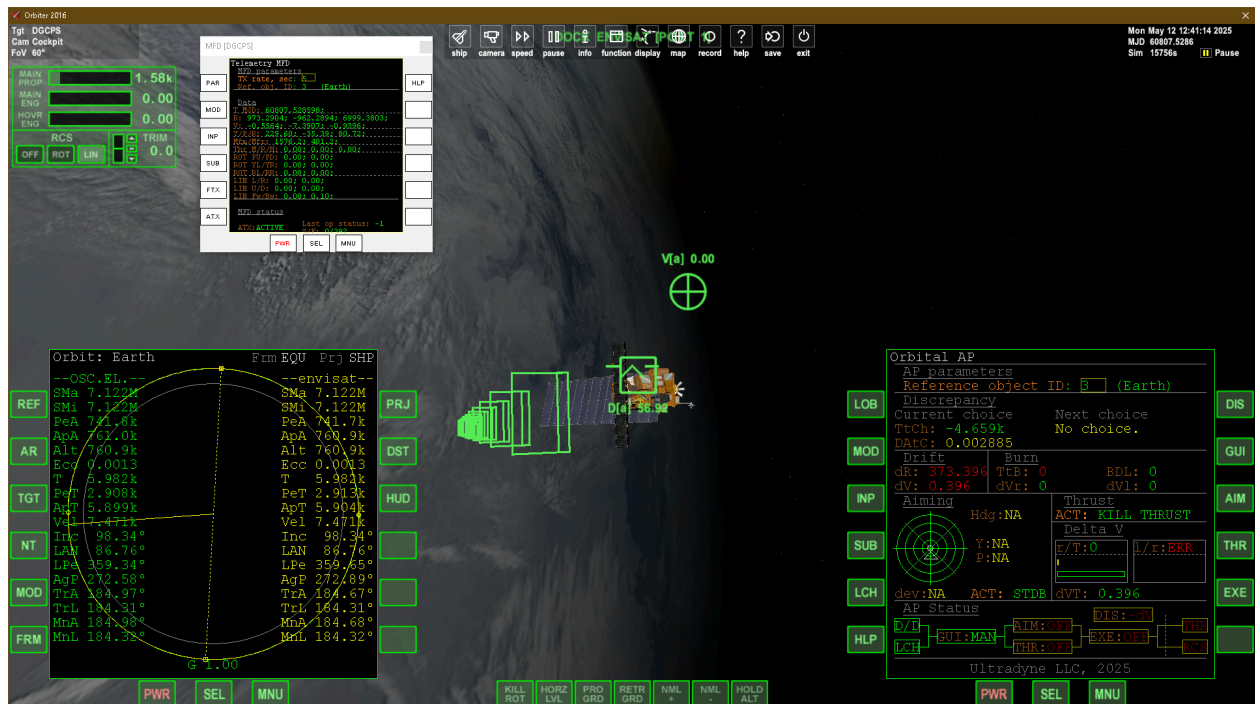
So, after a short discussion with the crew and Ground Control, Mission Commander decided to wait at least 30 minutes to understand the situation.

And that decision gave a result!

Within 10 minutes it became visible that the distance to Envisat was actively reducing. At ~11:40 distance was reduced to 168 km but relative speed grew to 156 m/s. Crew made a fast check of the delta-v budget. After intercept transfer and active orbit corrections total delta-v from propellant in main and RCS tanks was 724 m/s. Mission Commander decided to save at least 240 m/s of delta-v for return, so the available delta-v budget was 484 m/s. After ~11:50 relative speed started to fall. This looked promising, the crew spirit boosted. Mission Commander, while stayed alert, decided to initiate preparations for EVA. At ~12:05 distance was 80 km, while relative speed reduced to 100 m/s. At this moment, reading telemetry exchange data, Mission Commander noticed that Ground Control operates incorrect fuel masses in transmitted data.

Quick check showed that Telemetry MFD can not work in the background. After this, Mission Commander (using untranslatable Russian word-play) characterized the work of the Ground Control as bad and his and other crew members' level of attention as insufficient. Telemetry MFD was restarted and since this moment worked fine.

At 12:11:34 the distance to Envisat was reduced to 47 km. At 12:21:34 the target satellite became visible via binoculars (simulated with virtual camera zoom). At 12:29:20 after applying manually controlled Braking Burn (delta-v spent 74 m/s, propellant cost 210.6 kg) rendezvous was completed. Distance after intercept was 2.8 km, relative speed 0.15 m/s. While EVA Specialists worked on EVA preps, Mission Commander proceeded to the close approach phase.



Pic. 22: Simulation interface. Envisat is visible in the center of image, rangefinder reports 57 m relative distance; Orbital AP and Telemetry MFD interfaces are visible.

At **UTC 12.05.2025, 12:40:55**, DeltaGlider got 50 meters close to Envisat. All EVA preparations were finished, and Mission Commander decided to initiate EVA.

Hard work in the darkness

At 12:52:25, after required suit integrity checks, tools latching and depressurisation of airlock, EVA Specialist 1 Anna Apalkova, using MMU, detached herself from DeltaGlider and navigated her path towards the huge and silent space derelict. EVA Specialist 2 Rebecca W. Lee, locked herself to railings in the hatch of DeltaGlider, kept safety ropes in her gloved hands. Mission Commander Daniil Galakhov, hands on RCS controls, stayed ready for momental response to any dangerous changes.

Anna reached the satellite in about a minute, pulling safety tethering behind her. Envisat rotated slowly, turning in two planes simultaneously. For a time, she could not sync her rotation with satellite movement, but then, following methodics developed during training, "caught" the main axis of rotation. At 13:00:00 she stuck the adhesive anchor of the fastening system to the frame of Envisat slightly behind its star tracker system.

From short range, especially with a human figure nearby, gave scale to the scene, Envisat looked uncomfortably huge. With the deployed solar array it was the size of a torpedo boat.

Work was very hard because of lack of light - spacecrafts slided to eclipse, environmental light was weak and soon all turned to pitch black. Despite this, Anna insisted on continuing work.

After a short discussion, the crew decided to improve illumination with precise orientation of DeltaGlider towards Envisat, which could help to focus docking lights of a spacecraft on Envisat's front end. Rebecca winded off tethering ropes, while Daniil turned DeltaGlider to a better attitude. At 13:05:12, work had continued with better light conditions.

At 13:14:25 Anna unpacked DAD from the transport container, and continued unfolding its "sleeve". Same time, Mission Commander translated DeltaGlider 10 meters back, to keep optimal illumination parameters. Rebecca monitored the state of safety tethering and kept radio contact with Anna, being ready for support.

With proper illumination work goes faster, but still was tricky, mostly because of the size of Envisat. Using the words of one of the crew, it was "like pulling a tent cover on the rear end of a school bus".

Despite this all, at 13:24:18, DAD "sleeve" was deployed and attached. EVA Specialist 1 plugged AIBN cartridges into places, checked beacon set-up, and retreated back towards DeltaGlider, ready to initiate DAD inflation from a safe distance. After EVA Specialist 2 confirmed all safety checks, AIBN cartridges were initiated, and **at UTC 12.05.2025, 13:29:08 DAD was successfully deployed.**

At 13:35:57 all crew returned on board. After sealing the airlock, EVA Specialists continued to post-EVA procedures, while Mission Commander focused on situation analysis.



Pic. 23: View from virtual cockpit. Envisat with deployed DAD and EVA Specialist 1 Anna Apalkova in a space suit equipped with MMU are visible through the front window of the cockpit.

Astroagation

It was obvious, that after massive spend of delta-v for trajectory corrections further mission plan should be reworked. Next pass of the orbital plane over SRC Polygon was expected at ~19:37, so before this time DeltaGlider should return to Parking LEO, from where reentry interface should be reached by deorbit burn. But with limited delta-v budget planned maneuvers became... sophisticated.

So, at 13:42:37 crew reported to Ground Control that they'll use time left till the planned end time of EVA phase, for analysis of the situation.

After brief analysis, Mission Commander decided to rebuild the mission plan from the end of EVA phase to reentry, in a way most effective by delta-v spend. **At UTC 12.05.2025, 14:02:28, Mission Commander reported his decision to Ground Control, and initiated the next phase of the mission.**

It was obvious that for rapid planning of low-cost return transfer simplest and most effective maneuvers should be used. Thus, basic Hohmann transfer was chosen as initial point for analysis, and Mission Commander, using his calculating software, optimised the chain of maneuvers required to reach reentry interface.

At 15:28:32 mission planning was finished, next maneuvers was designed:

Maneuver	UTC (approx)	Δv [km/s]
Hohmann A Burn	12.05.2025 17:13:41	0.1394
Hohmann B Burn	12.05.2025 18:01:03	0.1420
Deorbit Burn	12.05.2025 18:45:48	0.0735

Total delta-v spend amounted to 0.3549 km/s, which was perfectly within the delta-v budget.

Planned maneuvers had been entered into orbital autopilot software. Trajectory correction was disengaged to avoid delta-v spend.

Long way home

DeltaGlider smoothly slid its orbit, slowly moving away from Envisat. At 15:47:52 the crew proceeded to energy and spirit restore mode, including high-calorie food combo (chocolate and energy drinks) and Khokhlov and Surdin lectures about cosmonautics.

Orbital drift grew slowly, reaching 35 km from planned orbit over the terminator zone. Flight was going fine, but - suddenly! - Hohmann A Burn was not applied automatically at 17:13:41 because of yet another bug in autopilot software. Happily, the crew already expected software glitches, so autopilot was switched in semi-auto mode and burn was applied manually.

Hohmann B Burn was applied manually at 18:01:03, and DeltaGlider reached parking LEO, from where reentry interface could be achieved.

EVA Specialists prepared all tools and payload for reentry, while Mission Commander re-checked manuals of reentry autopilot software. Following its instructions, deorbit burn should be applied later than planned to guarantee reaching target base. Based on previous experience and manual of reentry autopilot, Mission Commander applied the Deorbit Burn at UTC 12.05.2025 18:50:12. **Delta-Glider passed to suborbital trajectory.**

Final

Crew prepared the spacecraft for reentry, by switching all required systems to atmospheric entry mode, and setting SRC Polygon as target landing site. During preparation Mission Commander stayed in doubt, worried about too early deorbit burn. To prevent possible troubles, additional landing site (Chiang Mai RTAF base) were reserved for an emergency case. DeltaGlider passed over NYC, reached the Canada border and continued further towards the polar region. At 19:08:02 altitude 140 km was reached.

Slightly later **spacecraft reached the reentry interface**, passed it and rushed to the upper boundary of dense atmospheric layers. At altitude ~70 km zero gravity disappeared, and overload started to grow. Connection with Ground Control was lost during plasma blackout. Despite this, the automatics kept DeltaGlider on course, forcing it to slowly reduce speed. Maximal overload at altitude 55 km reached 3.1G, max overheat on wings edges didn't exceed 450 C.

At ~19:35 DeltaGlider passed from reentry to horizontal flight with speed $M \sim 3.9$ at altitude 20 km in 350 km from SRC polygon. Atmospheric autopilot was engaged, but speed reduced rapidly.

For a moment the crew were worried that the spacecraft was losing control. Altitude control in AtmosphericAP was turned off. Mission Commander tried to keep speed, burning the rest of the fuel from the tanks, at the same time turning DeltaGlider into a smooth dive. At altitude 18 km speed stabilized at $M \sim 2.0$, and control was restored. Spaceplane glided over Malacca strait, and after a deep left spiral reached final to land to runway 28 of SRC Polygon.

DeltaGlider landed normally at UTC 12.05.2025, 19:45:35.



Pic. 24: virtual cockpit of DeltaGlider. Landed on runway 28 of SRC Polygon.

Debriefing

Against all odds, **the mission was finished successfully.**

Despite malfunctions in flight control software, **all mission objectives were completed successfully.**

Mission planning software worked nominal.

All vehicle systems worked nominal.

After 12 hours and 5 minutes of flight, the spacecraft successfully returned to the base. No equipment damage detected. All crew safe, health conditions normal.

That was hard, but we did it.

Insights and conclusions

About software for space applications

- Space software (especially software for flight control) **requires much higher development quality standards** than any existing software, because:
 - **It must work on a macroscopic time scale.**
 - **You can not fix it "in production".**
 - **Cost of mistake higher than you may imagine.**
- Space software **requires full-scale testing in precisely simulated environment.**
 - **Typical unit- or acceptance testing will not give acceptable reliability.**
- Development of aerospace software **requires an aerospace approach.**
 - **Typical modern programming techniques and methodologies will not give results you need.**
 - In case of aerospace software, not software forms data and execution environment, but **data and execution environment forms software.**
- To make proper aerospace software, **you need related knowledge in subject area.**
 - **Hard skills**, hated by IT specialists, **are more important here.**
- Development of such software **always will require more time than you planned.** Your goal is to reduce development time to match at least mission timings.

About algorithms for flight control and mission planning

- Books provide algorithms, but **can contain errors, or be incorrectly understood.**
 - You **always need to test (in conditions as real as possible)** algorithms you implemented.
 - You need not just implement algorithms, but **clearly understand what and for what purpose they are doing.**
- Practical testing of algorithms **is urgently needed.**
 - Even if the algorithm is implemented correctly, has no sense errors and understood clearly, **in practice it can throw out a trick.** That trick even could be expected, but typically it happens in the worst possible moment, and its reasons are not obvious.
- **If you can not test your algorithm** practically:
 - **Use the simplest solution possible.**
 - **Move suspicious logic to a safe (controlled and testable) environment.**

About astrogator work during spaceflight

- Everything that can be prepared before launch, **must be prepared before launch.**
- Astrogator **should participate** in spacecraft- and mission design processes.

- Your spacecraft is doing, typically, almost **8 kilometers per second**. Plot this distance on the map, and remember it (or better walk it, on foot). This distance your spacecraft passes in 1 second.
- You **can not stop** the movement of your spacecraft.
- You **can not revert or change decisions** you made earlier.
- The Earth you are flying around is humongous big. Space is endlessly bigger. Even with your tremendous speed you need 90 minutes to turn around our planet. This gives the impression that "everything is slow in space". Inexperienced astrogators making from this a conclusion that they have a lot of time. But this is wrong. **You have time. But it's running out!**
- Never expect that the flight will follow your clear and elegant plan. So, **design the highway, not a trail.**
- **Ballistics engineer calculates trajectories. Astrogator looks for ways.**

Psychological aspects

They are consequences of the above. Briefly:

- Problems you will face will **always be harder than you can bear**.
- All **space projects require stubbornness**. Despite hardest conditions, it is necessary to push projects through to completion.
- All must be done just in time. You shouldn't do it "yesterday", shouldn't do "fast", shouldn't "catch in time" - but do things **"just in time"**.
- During a mission, especially in critical situations, **you will face the hardest stresses**. These stresses could continue not seconds or minutes, but hours and days. **This is normal**. You must be ready for this.
- Astrogator **must be more than a system engineer**.

Philosophy of spaceflight

Even short spaceflight, simulated in our educational mission, required 12.5 hours. Longer missions may require months, years and decades.

It's obvious and well known, but it has one aspect, very important by my opinion. For proper mindset, you should not treat space missions as "flight". This is more "life". I.e., you need to learn to "live" within mission (and flight).

Spaceflight in some aspects is very like the life of a human entity, or, better to say, like a process of being itself. Same as in life, you need to make choices, and face consequences of them. As in your life, you are the only person responsible for the consequences of your choices. And as in your life, you can not change anything after you made a choice. In space all this is just very visible, because here your choices primitivized to basic vector math and interaction with gravity, air drag and solar wind.

Being an astrogator means that you need to learn how to make choices. A right one. And it's not about "space doesn't forgive mistakes". It is forgiving. Or, better to say, it doesn't care.

But you must care.

Acknowledgments

To my team - **Anna Apalkova**, **Rebecca W. Lee**, and **ChatGPT** - for support, patience, trust and stubbornness.

To my mentors - **David A. Vallado**, **Aleksandr Khokhlov** and **Christophe Bonnal** - for shared knowledge and personal example.

To Orbiter 2016 developer - **Martin Schweiger** - for best space flight simulation software.

To add-on developers - **Yuri Kulchitsky**, **Brian Jones** and **Roman Starkov** - for their Orbiter 2016 add-ons, without which this project would not be possible.

To my LinkedIn contacts and followers, and also to other people knew about the project - for their support and attention.

Legal information

Legal Disclaimer

This document describes a simulated space mission conducted entirely within a virtual environment using Orbiter 2016 space flight simulator and supporting software. All mission procedures, hardware representations, maneuvers, and EVA operations were executed for educational and research purposes only.

The mission involved simulated interaction with ESA's Envisat satellite. We fully acknowledge that Envisat remains the legal property of the European Space Agency, in accordance with the Outer Space Treaty of 1967 (Article VIII). No claims, assumptions of control, or implied rights over the satellite are made.

All interactions in this simulation were conducted in full recognition of international space law and ethical boundaries of space activities.

This simulation was not affiliated with any space agency, corporation, or institution. All participants were private individuals operating without commercial, governmental, or political interests.

Any similarity to real missions, organizations, or future plans is purely coincidental or used as speculative educational fiction.

Use of Software and Intellectual Property

This simulation project used a combination of freeware and payware software products, including:

- **Freeware:**
 - Orbiter 2016 (by Martin Schweiger)
 - Add-ons for Orbiter 2016 (by Yuri Kulchitsky, Brian Jones, Roman Starkov and others)
 - Python 3.9.13 and standard Python libraries (numpy, matplotlib, flask, etc.)
 - LUA 5.1.2
 - CesiumJS 1.129 (used locally)
 - Blender 4.1, MshMaker, and other free modeling and visualization tools
 - Various open-source tools for telemetry, joystick emulation, and code development
- **Payware:**
 - Paint Tool SAI version 1.2.5 (used for creation of illustrations)
 - ChatGPT (GPT-4-turbo, March 2025) — used as co-programmer, support assistant and ground control simulation

All software was used in accordance with the **terms and limitations of their respective licenses**.

Special note is made regarding **Orbiter 2016**, which is distributed as freeware with restrictions on modification of binary files. According to the license available at <http://orbit.medphys.ucl.ac.uk/terms.html>, modification of Orbiter binaries is not permitted **for redistribution**. This simulation involved a **private modification of *LuaInterpreter.dll*** solely for educational and research purposes. The modified DLL will **not be published or distributed** under any circumstances, in full compliance with the Orbiter license.

All trademarks, spacecraft models, brand names, and satellite names remain the property of their respective owners and are referenced here for educational and illustrative purposes only. No commercial use of any third-party intellectual property was made.

List of literature

1. David A. Vallado, "Fundamentals of Astrodynamics and Applications, Fourth Edition"
2. Richard H. Battin, "An Introduction to the Mathematics and Methods of Astrodynamics, Revised Edition (AIAA Education) Revised, Subsequent Edition"
3. David A. Vallado, "Methods of Astrodynamics - a Computer Approach, Version 3.0"
4. Granino A. Korn and Theresa M. Korn, "Mathematical Handbook for Scientists and Engineers"
5. Malcolm D. Shuster, "The Quest for Better Attitudes"
6. James R. Wertz, "Spacecraft Attitude Determination and Control"
7. E.T. Whittaker, "A treatise on the analytical dynamics of particles and rigid bodies"
8. William Rowan Hamilton, "ON QUATERNIONS, OR ON A NEW SYSTEM OF IMAGINARIES IN ALGEBRA"
9. Renato Zanetti, "ROTATIONS, TRANSFORMATIONS, LEFT QUATERNIONS, RIGHT QUATERNIONS?"
10. J.L.Simon et al. "Numerical expressions for precession formulae and mean elements for the Moon and planets"
11. Nathaniel Bowditch, "American practical navigator, 2019 edition"
12. NAVPERS 92988 "[Space Navigation Handbook](#)"
13. E. R. Lancaster and R. C. Blanchard, "A Unified Form of Lambert's Theorem"
14. E.M. Gaposchkin and A.J. Coster, "Analysis of Satellite Drag"
15. Kikuko Miyata, Rei Kawashima, Takaya Inamori, "Detailed Analysis of Aerodynamic Effect on Small Satellites"
16. John C. Adams, Jr. "Atmospheric Re-Entry"
17. Reentry Safety Requirements:
 - 17.1. [ESSB-ST-U-004](#)
 - 17.2. [NASA-STD-8719.14A](#)
 - 17.3. [JMR-003E](#)
18. Kirpischikov V.P., "Trajectories of BURAN Orbiter's Descent and Landing Algorithms of the Automatic Guidance and Control"
19. "Энциклопедия крылатого космоса"
 - 19.1. [Посадка ОК "БУРАН"](#)
 - 19.2. [Алгоритмы автоматического управления](#)
 - 19.3. [Основные аспекты аэродинамического проектирования ОК "БУРАН"](#)
20. UNOOSA "[The Outer Space Treaty](#)"
21. D. Milligan, P. Bargellini, A. Rudolph, L. Ventimiglia, and Y. Eren: "Envisat Flight Experience: FDIR and Lifetime Optimisation"
22. ESA, "ENVISAT-1 Mission & System Summary"
23. S. Sommer, J. Rosebrock, D. Cerutti-Maori, and L. Leushacke, "TEMPORAL ANALYSIS OF ENVISAT'S ROTATIONAL MOTION"
24. Wikipedia, [Orbital decay](#)
25. Joyce Dever, Bruce Banks et al, "Materials degradation in space"
26. Wikipedia, [Project Echo](#)