Titanic Space Odyssey 1-2

Realized by:

Group 14

Michelle Lear (i6240491)

Gunda Karnite (i6252172)

Hemachandra Konduru (i6254397)

Benjamin Gauthier (i6256403)

Simon Köhl (i6242698)

Mentor:

Otti d'Huys

Project coordinator:

Katharina Schuller

Examiners:

Nico Roos

Pieter Collins

Maastricht University

Faculty of Science and Engineering

Department of Data Science and Artificial Intelligence

ABSTRACT

This paper discusses the planning of a manned mission to Titan and return to Earth by means of computer coding.

Titan is Saturn's largest moon and the second largest in the Solar System. It is the only moon in the solar system that has a dense, planet-like atmosphere. Astrophysicists believe that the conditions on Titan are similar to that of Earth in its early days and Titan is the most Earth-like world to date. The biggest difference between the two is that Earth is closer to the sun, thus it is warmer.

Titan has an orange haze that surrounds it and hides its surface which had kept it a closed book until the Cassini mission. The Cassini spacecraft traveled 7.9 billion kilometers for twenty years in space, thirteen of those were exploring Saturn and unveiling mysteries the world did not know existed. It showed us the complexity of Saturn's rings and provoked scientists to look at the knowledge they had on the solar system again. (Barnett, 2021)

To do create a working program, research on ordinary differential equation solvers, such as Euler's method, Runge-Kutta's fourth order method and the Verlet method were used as well as a method to find the minima, Newton-Raphson's. These made it possible to simulate the full trajectory from Earth to Titan and back. The use of mathematical methods helped model the external factors, like gravitational forces, wind, air resistance and fuel consumption, which made the research as realistic as possible.

SYMBOLS

- h: step size
- t_i: time
- $k_{i,}$: iterations in the Runge-Kutta method
- w_i: the function value at each iteration of Runge-Kutta
- $f(x_n)$: function value at x_n for $n \geq 0$
- $f'(x_n)$: derivative value of the function value at x_n for $n \geq 0$
- x_n iterations of the Newton-Raphson method for $n \geq 0$
- $\vec{x}$: x vector
- N: Newtons
- km/s: kilometers per second
- Δt: change in time
- x_n: a real number x representing a point in a function f
- $\vec{a}$: acceleration vector
- $\vec{v}$: velocity vector
- M: final mass
- V_R: final velocity
- V_e: effective exhaust velocity
- V_0: initial velocity
- F_{max}: maximum thrust
- G: Universal constant of gravity
- m_i: mass

ASSUMPTIONS

The following is assumed about the workings of the spaceship:

- The mass of the craft is 78000 kilograms (dry mass[1])
- The mass of the lander is 6000 kilograms
- The effective exhaust velocity is 20 km/s
- The maximum thrust is 3e7N

[1] The mass without fuel

TABLE OF CONTENTS

1 INTRODUCTION

This paper describes the inner workings of a mission to Titan. Primarily, knowledge on the solar system and how all the planets work in conjunction with each other is needed. Most importantly, the planets surrounding Titan and Earth. Titan is the twentieth-most distant moon of Saturn and the sixth-farthest among these large enough to assume a spheroid shape. It is also Saturn's largest moon - and the second-largest moon in the solar system - and the only object other than Earth for which clear evidence of stable bodies of the surface liquid has been found. Titan's diameter is roughly 50% larger than Earth's moon. To reduce the scope of the investigation, not all bodies in the solar system were considered, the focus was on the following planets: the Sun, Earth, the Moon, Saturn, Jupiter, Venus, Mars, Mercury, Uranus, Neptune and, of course, Titan.

A simulation of the Solar System in which a launch to Titan, a landing on Titan and journey back to Earth is modelled. The travel must be optimized to reduce the cost of the mission. Thus, we had to find the best trajectory path and reduce the fuel consumed. To explore and grasp how exactly to go through with this mission, an exploratory mission was conducted first. An exploratory mission is done in order to discover and learn about the physics in the Solar System before a manned mission. This was done by shooting a ballistic cannon into space and seeing how the gravitational forces of the planets affected it, as well as how far the probe travelled with a certain starting speed. Doing that allowed us to develop algorithms to calculate the trajectory beforehand and this in-turn allowed us to optimize other specifications.

This paper aims to ultimately answer the following question: *"What are the optimal specifications for a mission to Titan?"*. The research was conducted by a group of first year bachelor students at Maastricht University so "optimal", in this case means the best possible specifications to our capacity. To answer that we need to answer the following questions as well: How much fuel is needed? What is the best step size for these calculations? Which methods to solve differential equations should be used, and which method should be used to plan the trajectory? This is all answered through modelling a manned mission to Titan from Earth, and then from Titan to Earth and performing several tests and trying different implementations.

There are eight sections in this paper, including the introduction. In the Methods section, there is step-by-step, in-depth information on the formulas and numerical methods used to calculate physical aspects of the mission: differential solvers and path planners. Implementation covers how the calculated values were implemented and how the working simulation was made. The Results section presents what is described in the previous section, Methods, using tables and figures for visual representation and better understanding. Then, we find the Discussion, this is where the results are interpreted, and analysis were performed against the research. The Conclusion summaries the study and answers the research questions. The final two sections, namely Bibliography - contains references - and Appendices contain any additional figures that can aid in further understanding the paper.

2 METHODS

When planning this mission through deep space, many mathematical and computational methods had to be used to obtain a close-to-exact simulation of the ongoing physics, plan trajectories and design controllers that exploit or counteract the effects of the physics to ensure safety and minimize the cost of getting to Titan and landing on it. This section aims to describe the methods used and implemented to tackle these problems.

2.1 DIFFERENTIAL EQUATION SOLVERS

To complete this mission - orbit Titan, land on Titan and return to Earth – the equation below needed to be solved.

$$F_i^G = \sum_{j \neq i} -Gm_i m_j \frac{x_j - x_i}{\|x_i - x_j\|^3}$$

To solve it, the use of differential equation solvers is required. This section will discuss three different solvers that perform the same task but in different ways, namely the Euler Method, Fourth order Runge-Kutta Method and the Verlet Method.

2.1.1 EULER METHOD

The Euler method is a first-order method that numerically solves initial-value problems with differential equations. This means that the global error is O(h), where h is the size of each step: a relatively big error, however Euler's method is an easy method to implement and start with. The Euler method is defined by the following formula[2]:

$$w_{i+1} = w_i + hf(t_i, w_i) \qquad for\ i > 0$$

Where w_i is the current known value, h is the step size and$f(t_i, w_i)$ is the function value evaluated at ti, the current time value and w_i. Euler's method has a local error of O(h^2).

2.1.2 FOURTH ORDER RUNGE-KUTTA METHOD

The Runge-Kutta method is a numerical integration method for ordinary differential equations. It has 5 steps in each iteration. This method works with time values, step values and w values. To initialize the method, it was necessary to have starting values and end values for certain variables. We needed to know what the step size (h) was, what the initial w0 value was, the initial time and final time. These helped us calculate the derivatives with a global error of O(h^4). The formula can be seen below[3]:

[2] Source: Numerical Analysis (Burden & Faires, 2011)
[3] Source: Numerical Analysis (Burden & Faires, 2011)

$$1.\, w_0 = \alpha$$

$$2.\, k_{i,1} = hf(t_i, w_i)$$

$$3.\, k_{i,2} = hf\left(t_i + \frac{1}{2}h, w_i + \frac{1}{2}k_{i,1}\right)$$

$$4.\, k_{i,3} = hf\left(t_i + \frac{1}{2}h, w_i + \frac{1}{2}k_{i,2}\right)$$

$$5.\, k_{i,4} = hf(t_{i+1}, w_i + k_{i,3})$$

$$6.\, w_{i+1} = w_i + \frac{1}{6}(k_{i,1} + 2k_{i,2} + 2k_{i,3} + k_{i,4})$$

$$all\ for\ i > 0$$

2.1.3 VERLET METHOD

The Verlet[4] method is a numerical integration method that is more accurate and stable than the Euler method. It has a global error of $O(h^2)$ for the velocity and $O(h^3)$ for the position. (Branislav, n.d.). The Verlet method needs to be bootstrapped to be used as it is not self-starting.

$$\vec{x}(t + \Delta t) = \vec{x}(t) + \vec{v}(t) + \frac{\vec{a}(t)\Delta t^2}{2}$$

$$\vec{v}(t + \Delta t) = \vec{v}(t) + \frac{\vec{a}(t) + \vec{a}(t + \Delta t)}{2}\Delta t$$

2.2 PATH PLANNING

To simulate the path taken to a planet, we needed to compute the trajectory using some numerical method. The numerical method needed to be able to find the root of a function to minimize the cost function which in turn helped find a good trajectory.

2.2.1 NEWTON RAPHSON

The Newton-Raphson method is a numerical method for finding the root of a function. It is used iteratively, and each iteration produces a more accurate approximation of the root. (Burden & Faires, 2011). The method is denoted by the following formula[5]:

[4] Source: (Verlet Integration, 2021)
[5] Source: Numerical Analysis (Burden & Faires, 2011)

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

In our case, $f(x_n)$ is the same equation prescribed in Section 2.1 Differential Equation Solvers. This method used a cost function which gave us numerical values between 0 and 1 which represented the cost of getting to Titan given some starting parameters. Using this algorithm, we were able to minimize the cost function and therefore find optimal starting parameters for the mission. This worked to an accuracy of $7.5e^{-7}$.

2.3 CONTROLLERS

To solve the problem of landing safely on titan, several approaches from the field of control theory have been considered. The simulation of the physics for the landing consists of and ordinary differential equation known as Newtonian harmonic motion in two dimensions. Therefore, 2 parameters, that influence 6 variables, namely x, y, ϑ, $\dot{x}$, $\dot{y}$ and $\dot{\vartheta}$ need to be controlled.

2.3.1 OPEN LOOP CONTROLLER

An open loop controller is rather primitive, since it does not consider any changes in the system, but instead only considers the beginning state. An example of an open loop controller in our use case would be a timer that only considers the time needed to slow down the lander, which can be obtained by direct integration of the motion terms with a constant thrust and therefore constant acceleration. However, we did not implement this controller since it cannot deal with stochastic changes in the environment, which is why it would perform poorly when used in combination with the stochastic wind simulation. Also, since our closed loop controller was performing well already, we did not implement the open loop controller because it would have led to worse results.

2.3.2 CLOSED LOOP CONTROLLER

Inspired by drone flight controllers, our closed loop controller uses an outer and inner loop architecture. The outer loop takes the error in position as inputs and returns a desired theta and a desired $\dot{y}$ or correcting the lander's x-position. The errors in the desired and actual states are then fed into the inner loop, which returns the correction thrusts *u* and *v* respectively to get to the desired state. This architecture allows for a higher generalization and therefore more tolerance when dealing with disturbances or different heights than what the controllers were individually tuned for.

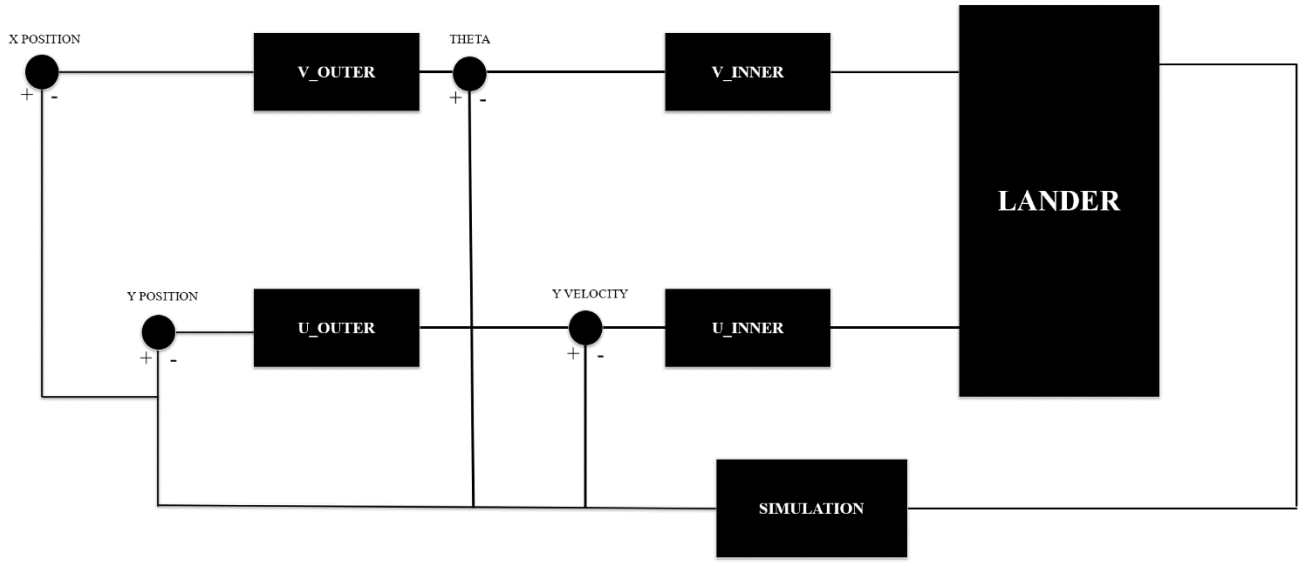


Figure 1 : Closed Loop Controller Diagram

3 IMPLEMENTATION

Writing the algorithms that implemented the numerical methods was not enough alone. It was necessary to integrate all the algorithms. This section describes those processes.

3.1 GRAPHICAL USER INTERFACE

To represent the work in a more understandable fashion, a graphical user interface was implemented using JavaFX. JavaFX is an open-source client application platform that is built on Java (JavaFX, 2021). It is necessary to have JavaFX installed to run the program.

First, the Solar System was modeled using Sphere objects. Each planet was represented by a Sphere object called. We went on to create a new class called 'CelestialBody' which created these spheres, and it also included the position in the solar system by using position and velocity vectors supplied by the project coordinators.

Appendix A shows the GUI of our program. At the top of the window, there are buttons which can be used to change the view from one planet to the next. In the figure, the planets that are seen are Saturn and Titan. A white sphere is also seen which a trajectory is point of Saturn's orbit around the Sun. Each planet has its own texture which resembles that of the planet in real life.

The last two buttons, from left to right, change the speed at which the simulation is shown. "x100" for one iteration of the draw loop, originally 1 second, the simulation is updated 100 times. For example, with a timestep of 300 seconds, we would have 300×100 updates, with a factor of 100. This means 1 second in the draw loop would cause our simulation to jump to the 30000^{th} index in the trajectories.

As soon as the landing is initialized, we switch from three-dimensions to two-dimensions. We did this using JavaFX again. We used a .png picture as the rocket object. This object was moved around the scene using the calculations from the trajectory planning and the controllers for the landing. The animation shows the forces that are acting on the rocket and how the mathematical calculations correct the landing. **Appendix B** shows a screenshot of the program when the rocket is taking off once again on its course back to Earth.

3.2 SOLVERS

The rocket was launched on the 1^{st} of April 2020 at midnight. Since three solver algorithms were coded, we had to find a way to easily interchange between these solvers to facilitate testing and implementation. This was done by creating a Solver Driver which inherits the ODESolverInterface. We initialize the Solver Driver object with a String that contains the name of the desired solver, and the solver driver implements the corresponding algorithm into the rest of the program.

When the rocket is launched, the full trajectory to the desired planet, Titan in this case, has already been calculated and saved in position and velocity vectors. The accuracy of these calculations can differ depending on which step size is used. In the final product, a step size of 300 seconds was used.

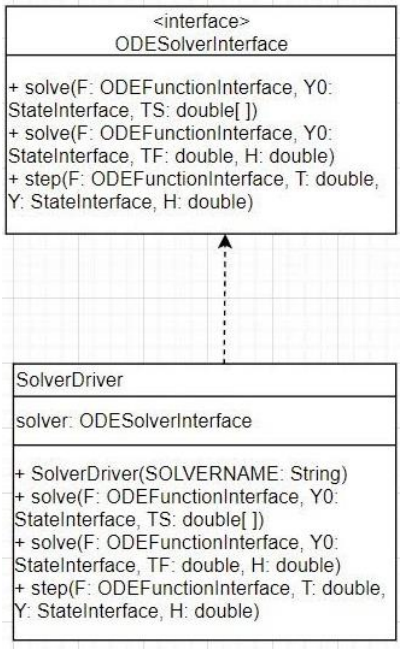


Figure 2 : SolverDriver UML Class Diagram

Figure 3 represents a UML class diagram of the Solver driver and its implementation specifications.

3.3 CONTROLLERS

The controllers for the third phase were implemented to receive an array of gains which are all applied to their respective term in the PID-Controller, represented by the following formula[6]:

$$u(t) = K_p e(t) + K_i \int_o^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

where the gains are denoted as K_p, K_i, K_d respectively. Those PID-Controllers are then combined in a FlightController class, which takes in the number of required controllers and their gains. Using those classes, we implemented one PD controller in the inner loop responsible for adjusting the side thrust v and three P controllers in the inner and outer loop for adjusting the desired $\dot{y}$, ϑ and main thrust u respectively. Tuning of the gains for the controller responsible for determining v was done by fixing the lander to one position and adjusting the gains such that the response in adjusting ϑ to be close to the desired ϑ using a side thrust v is reasonably fast and does not lead to large oscillations. To improve the stability of our rocket, we also saturated the output of the V_OUTER controller to be $\mp\frac{\pi}{2}$ at most. This way, the controller would never give

[6] Source: https://en.wikipedia.org/wiki/PID_controller

values larger than that which would cause the rocket to spin to an angle where almost no vertical thrust can be applied, causing it to drop to the ground. The main thrust is also saturated to be 3e7N at most and 0N at least (to prevent negative thrust), but the limit of 3e7N was never reached during the lander's descent. The side thrust is also saturated to be $\mp$25N at most. This was done to prevent unreasonably high thrust forces.

3.4 WIND MODEL OF TITAN

The Huygens lander was launched as a joint mission by NASA, the European Space Agency, and the Italian Space agency to explore Titan. The Huygens Lander experienced a wind speed of 120m/s at an altitude of around 120km. The wind speed dropped gradually during the descent and a wind speed of 0.3m/s was measured at surface level.[7] Based on this data, the following linear function was modeled:

$$y = 0.0009975x + 0.3$$

where y represents the wind speed at an altitude x.

A wind was calculated only for altitudes less than 600km from the surface of Titan. The resulting wind speed calculated by the function above was scaled by a number between 0 and 1 to introduce stochasticity.

During the descent of the Huygens Lander, two accounts of reversal in wind directions were recorded.[8] To introduce this uncertainty in our wind model, we generated a random value of type double between 0.0 and 1.0, and if the value was found to be less than 0.1, the wind speed was scaled by -1. This implies that there is a 10% chance of reversal of wind direction.

[7] Source: (The way wind blows on Titan, 2007)
[8] Source: (The way wind blows on Titan, 2007)

4 EXPERIMENTS

To have the best possible working product, we tried and tested different methods of computing the same values and different methods that resulted in the same final state. By doing this, we could recognize which methods were most suitable with the rest of our product and which were not feasible.

4.1 COMPARING SOLVERS

To get an optimal working product, performing tests on different ordinary differential equation solvers was the best way to see how the solvers differed from each other. These tests were the following:

A. Comparing the trajectories calculated with different step sizes.
B. Comparing the runtime with different step sizes.
C. Calculating the absolute and relative errors of the computations

Test A is useful because we wanted to use the step size that computes an accurate trajectory in a reasonable amount of time. Next, test B, is relatively important because the product needed to be functioning and pleasant to use. We stopped any testing if the program did not manage to compute a trajectory under fifteen minutes. And finally, the absolute and relative errors were calculated because if these errors were too high, our simulations were not accurate enough and we encountered problems within space like crashing into celestial bodies.

4.2 PATH PLANNERS

Throughout the project, we tried several different methods for correcting and planning the path. Although there were pros to these methods, the cons outweighed them. The following paragraphs explain what was done and why we did not use them in the final product.

We used the Newton-Raphson method for the ballistic trajectory, but it could not be used for entering orbit because the rocket was moving too quickly[9] and we could not bring enough fuel, nor exert enough thrust to slow down. We then tried to expand the deceleration process over a long period of time using the magneto plasmatic thrusters, but this shifted the actual point of impact in Titan's orbit because it would take longer to get there.

These problems could have been solved by implementing more sophisticated control logic or pre-calculating an optimal trajectory. For example, the Newton-Raphson's method could

[9] The average orbital velocity was $1000 m.s^{-1}$ to $2000 m.s^{-1}$ for an orbit in the range of 100km to 300km above Titan's surface.

have been implemented with the inverse Jacobi matrix[10]. The inverse Jacobi matrix would give us the ability to aim for a specific point in Titan's orbit rather than a random point at some distance. This would allow us to decelerate until our current velocity matches the velocity needed at the actual computed entry point into the orbit. Regrettably, time constraints did not allow for this to happen.

4.3 FUEL CONSUMPTION

This mission used engines and thrusters to apply a force to the rocket and move it through space. These engines and thrusters were also used to land the rocket, with the help of controllers. We needed to find out how much fuel is needed to complete the mission and then calculate how much this would cost us.

$$\Delta t = M\frac{V_R - V_0}{F_{max}}$$

$$a = \frac{V_R - V_0}{\Delta t}$$

$$\Delta m = \frac{Ma}{V_e}$$

Firstly, we calculated how much time is required for rocket to speed up from initial velocity to desired velocity. Secondly, we calculated what is the acceleration needed to speed it up in such time frame. Thirdly, we calculated the mass of fuel needed to acquire desired velocity with the previously calculated acceleration. Those formulas were then used in our Rocket class to simulate fuel usage at given time steps. The fuel is reduced once at the start and another time when reaching titan to get to orbital velocity. The orbital velocity needed is calculated using the closest distance the probe can get to Titan acquired by Newton Raphson and the probe's current mass. The formula for the orbital velocity is the following[11]:

$$v_{orbit} = \sqrt{G\frac{m}{r}}$$

acquired by setting the centripetal force F_c equal to the gravitational force F_G.

[10] A representation of the derivative of a function f at every point where f is differentiable (Khalil & Dombre, 2004)
[11] Source: https://www.youtube.com/watch?v=nxD7koHdQhM

5 RESULTS

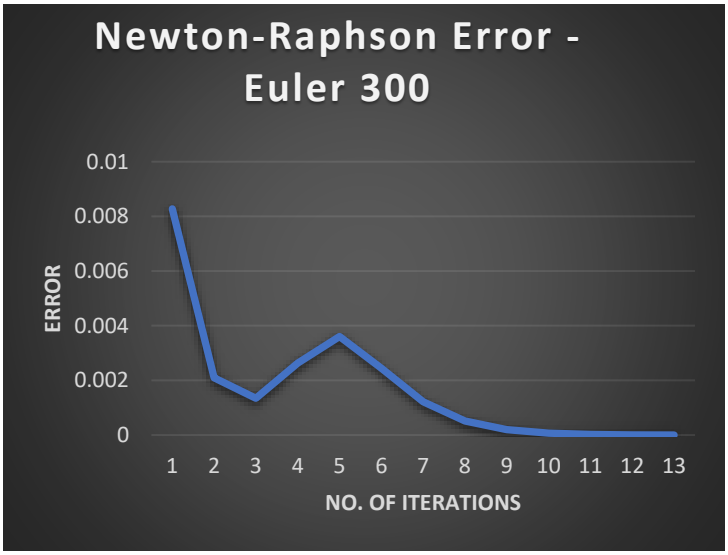


Figure 4 : Newton-Raphson Error Euler h = 300

Figure 4 illustrates the error over 13 iterations for a step size of 300 using the Euler method.

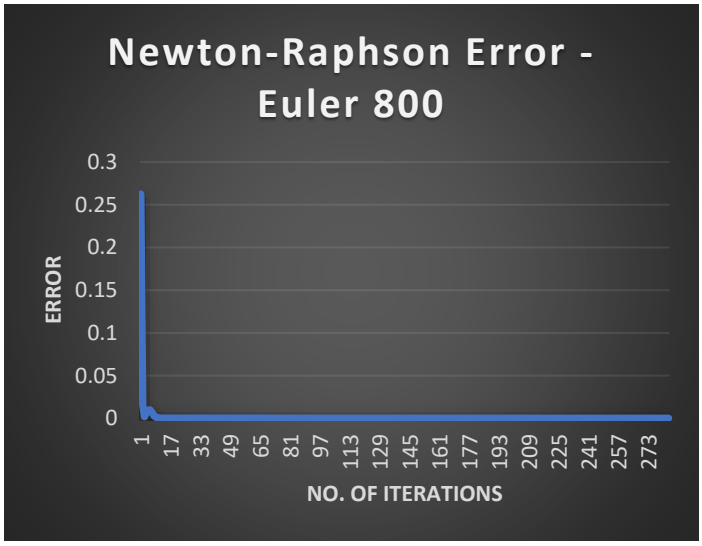


Figure 3 : Newton-Raphson Error Euler h = 800

Figure 3 illustrates the error over 274 iterations for a step size of 800 using the Euler method.

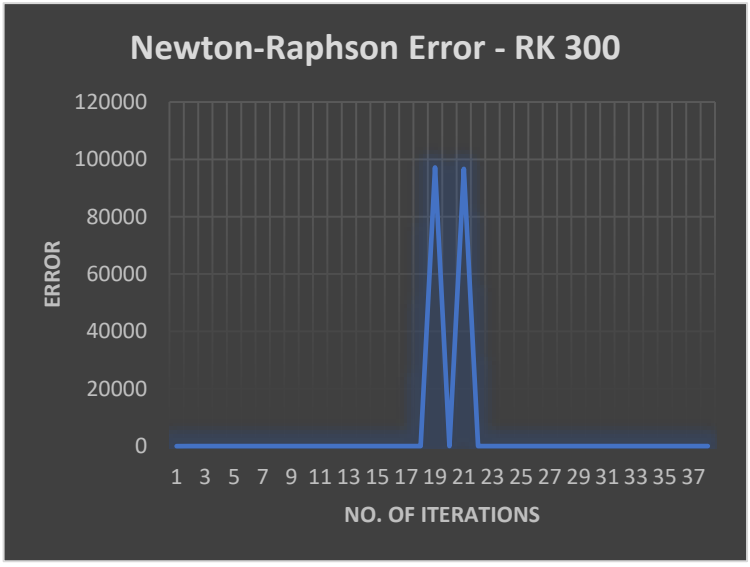


Figure 6 : Newton-Raphson Error RK h = 300

Figure 6 illustrates the error over 37 iterations for a step size of 300 using the Runge-Kutta method.

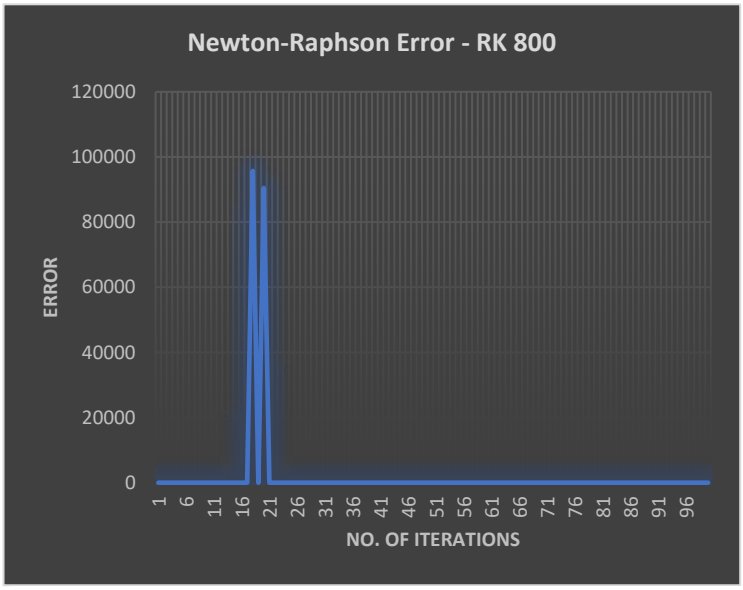


Figure 5 : Newton-Raphson Error RK h = 800

Figure 5 illustrates the error over 37 iterations for a step size of 300 using the Runge-Kutta method.

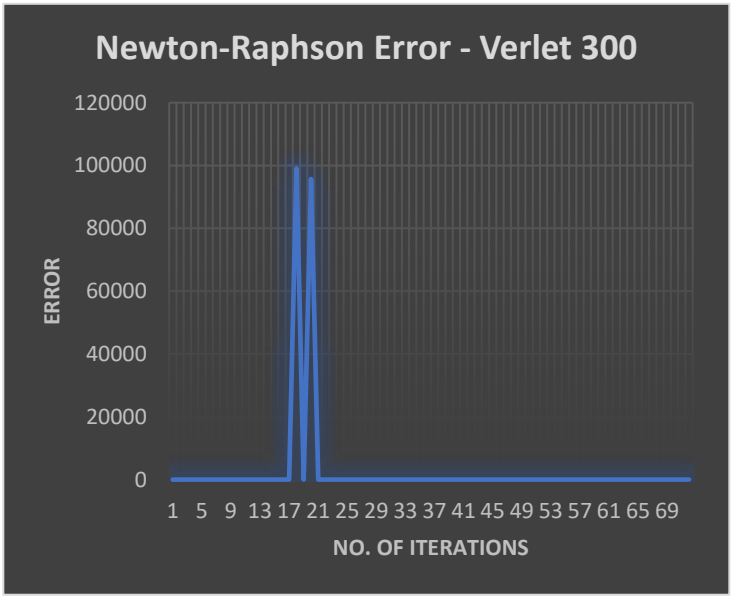


Figure 8 : Newton-Raphson Error Verlet h = 300

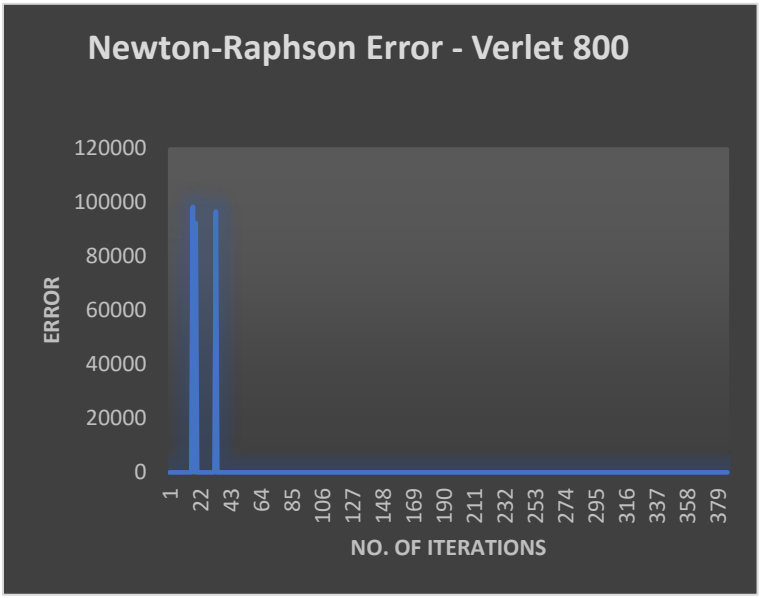


Figure 7 : Newton-Raphson Error Verlet h = 800

Figure 8 illustrates the error over 69 iterations for a step size of 300 using the Verlet method.

Figure 7 illustrates the error over 379 iterations for a step size of 800 using the Verlet method.

	Mean Absolute Error		
Step size	**Euler**	**Runge-Kutta**	**Verlet**
300	1.24E+13	2.06E+12	6.25E+12

Figure 9 : Mean Absolute Error

Figure 9 shows the mean absolute error of the solvers for step size 300.

6 DISCUSSION

In this section, the results described in the previous section are interpreted and explained.

To analyze the Euler method, we looked at figures 3,4 and 9. Figure 4 illustrates the computed Newton-Raphson error as the program was running for step size 300 using the Euler method. The error has a steep decline in the first computations, then it spiked on the fifth iteration and then value dropped once again. The x axis shows the number of iterations and this graph converged after only thirteen iterations. This tells us that the step size 300 for the Euler method is very reliable and efficient. Figure 9 shows us that the mean relative and absolute error are very low, therefore it is not only efficient and reliable, but also very accurate. Whereas figure 3 shows us that the step size 800 for the Euler method takes much longer to converge, 274 iterations, and the Newton-Raphson error is 1.99E+00 times larger than the error for a step size of 300 using the Euler method.

Next, we look at the Runge-Kutta method. This method was a lot more computationally complex hence, the calculations did not converge quickly when using Newton Raphson in combination with it. Figure 4 shows only 36 iterations, at first glance, this may seem to be good but, it took over one hour and a half to compute this and the algorithm never converged.

Lastly, the Verlet method struggled to converge. Figure 6 and 7 show that several iterations of the error are calculated, and the method did not converge within the given time. Overall, the Verlet method did not work well with the final landing.

From this we can deduce the local optima for the step sizes are 300, 400 and 100 for Euler, Runge-Kutta and Verlet respectively. Furthermore, a step size of 300 was the global optima as we could see common trends in each of the solvers here.

7 CONCLUSION

In conclusion, to find out what the optimal specifications for a mission to Titan are, we found that multiple algorithms, written with knowledge of known numerical methods, needed to be used to have a smooth and safe trip. The numerical methods used were proven methods by several mathematicians and they have been used for years. We discovered, although the differential equation methods are distinct, that some methods work better than others. A method was considered better than another if its relative and absolute error was lower than another, and if the algorithm converged in a shorter amount of time than another.

We found that the method that did this the best was the Euler method with a step size of 300. The trajectory planning was done using the Newton-Raphson method which converged 85% of the time. The full distance travelled was approximately 856 million kilometers. Using these two numerical methods together resulted in the shortest possible path to Titan and back to Earth. With a starting mass of 707800 kilograms, the rocket needed to take 6 million liters of fuel to take off which cost us 7740000€[12].

This study can be used to gain information and get ideas for modelling any trip to space. The methods used and algorithms created can be used to plan trajectories to any planet in the Solar System, given the assumptions are the same as in this project. In closing, we believe that the program was successful in achieving its goals as well as answering the research question concisely.

[12] Source: https://www.statista.com/statistics/603745/diesel-fuel-prices-netherlands/

8 BIBLIOGRAPHY

Barnett, A. (2021, June 9). *Cassini*. Retrieved from Solar System Exploration: https://solarsystem.nasa.gov/missions/cassini/overview/

Branislav. (n.d.). *Computational Methods of Physics PHYS4660*. Retrieved from Physics.udel.edu: <http://www.physics.udel.edu/~bnikolic/teaching/phys660/numerical_ode/node5.html>

Burden, R. L., & Faires, J. D. (2011). *Numerical Analysis.* Brooks/Cole.

JavaFX. (2021). Retrieved from openjfx: https://openjfx.io/

Khalil, W., & Dombre, E. (2004). *Jacobian Matrix*. Retrieved from ScienceDirect: https://www.sciencedirect.com/topics/engineering/jacobian-matrix

OMEGA. (2021). *What is a PID Controller?* Retrieved from Omega: https://www.omega.co.uk/prodinfo/pid-controllers.html

Planet Texture Maps. (2006). Retrieved from Planet Pixel Emporium: http://planetpixelemporium.com/neptune.html

Planetary Fact Sheet - Metric. (2019, October 21). Retrieved from NASA.gov: https://nssdc.gsfc.nasa.gov/planetary/factsheet/

Reddy, L. (2021, May 24). *Runge-Kutta 2nd order method to solve Differential equations*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/runge-kutta-2nd-order-method-to-solve-differential-equations/

Shiffman, D. (2012). *The Nature of Code.*

The way wind blows on Titan. (2007). Retrieved from European Space Agency: https://www.esa.int/Science_Exploration/Space_Science/Cassini-Huygens/The_way_the_wind_blows_on_Titan

Titan. (2019). Retrieved from Planet Texture Maps Wiki: https://planet-texture-maps.fandom.com/wiki/Titan

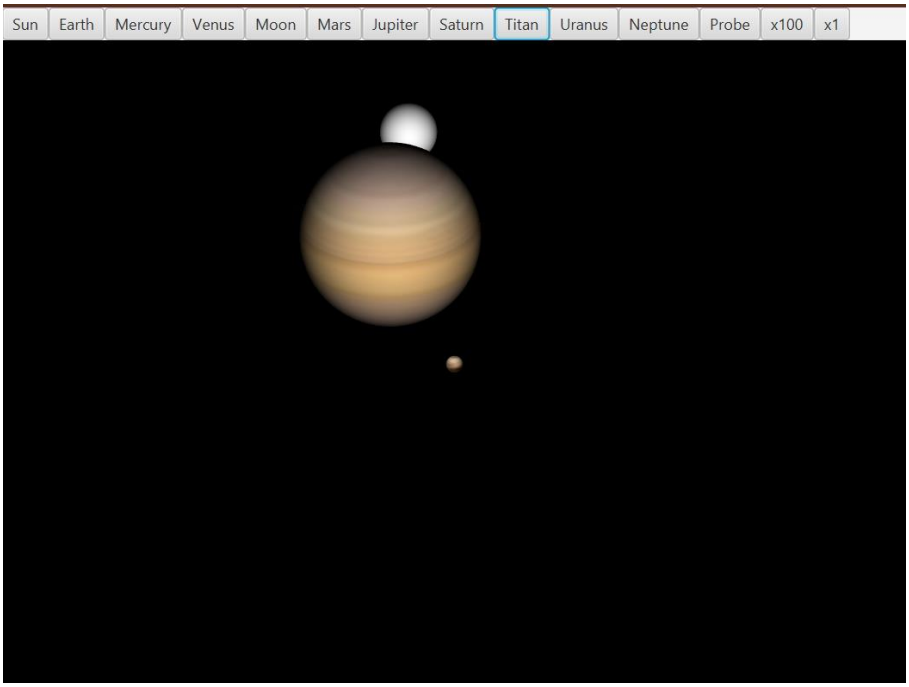
Verlet Integration. (2021). Retrieved from Algorithm Archive: https://www.algorithm-archive.org/contents/verlet_integration/verlet_integration.html

Walker, E. (1996, May 8). *The Rocket*. Retrieved from MIT Department of Aeronautics and Astronautics: http://web.mit.edu/16.00/www/aec/rocket.html

Walter, E. (2021, June 14). *geostationary orbit*. Retrieved from Cambridge Dictionary: https://dictionary.cambridge.org/dictionary/english/geostationary-orbit

9 APPENDICES

APPENDIX A



APPENDIX B