# Group 8 – Final Report

POȘTARU Damian, RIO Diogo, MEKHALLALATI Husam, MUGARZA PASCUAL Imanol, YANG Yiqiu (Isa), SMIT Tom, PLOEGSTRA Willem

## Abstract

The main aim of this report is to simulate a space mission from earth to one of Saturn's moons, titan. During this mission, the spaceship has to enter a geostationary orbit around titan, deploy a module that has to land on titan and finally return back to earth. A stochastic wind model is implemented to simulate the wind during the landing on titan. For the landing, both an open-loop as well as a closed-loop controller is implemented.

A successful landing was performed with the open-loop controller, with the closed-loop controller coming very close. The addition of wind to the simulation showed the open-loop controller performing very poor during landing, while the closed-loop controller showed better results. The implemented differential equation solvers were tested on accuracy, with the result that the Verlet solver is more accurate regarding the calculation of the positions than the fourth-order Runge-Kutta solver. The complete simulation is visualised in a Graphical User Interface.

# Table of Contents

# 1. Introduction

## 1.1 Saturn's moon Titan

Titan, Saturn's largest moon, attracted scientist's attention when the Cassini-Huygens space probe revealed hitherto unknown details about its atmosphere and surface. It was discovered that the moon shows certain similarities to the Earth [1], including the atmosphere, which like the Earth's atmosphere is also composed of nitrogen. Although Titan is outside the habitable zone and due to the low temperatures, the water takes the form of ice [2], it is said that the preforms of life could exist on or below its surface [3]. The conditions on Titan of these times are said to be similar to the conditions on Earth at the time life began. Analysing this could help understand how life on earth started.

The goal of project 1-2: A Titanic Space Odyssey, is to simulate an exploratory mission back and forth from Earth to Titan. To this end, we have created a programme which is separated into three distinct parts. The first part concerns the launch of a rocket from Earth. After a successful launch, the journey from Earth to Titan is simulated. Once the rocket in the simulation enters Titan's orbit, the simulation of the landing begins. This graphical representation mimics the descent of a lander on Titan's surface using an open-loop or closed-loop control system and includes a stochastic wind model of Titan's atmosphere.

## 1.2 Problem Statement

This report will investigate, analyse, and test multiple algorithms that help us answer the following problem statement: "*Can we make a simulation that brings a manned mission to land on titan and safely return back to earth*".

### 1.2.1 Research Questions

To tackle the problem statement for this research, a selection of research questions will be answered:

1. What impact does a variation of wind speed have on the landing success rate of both the open-loop and the closed-loop controller?
2. What are the effects of the percentage of deviation in our Stochastic wind model?
3. Which of the implemented solvers give more accurate results?

# 2. Methods

## 2.1 Differential Equation Solvers

To solve the problems described in **section 1.2** a number of Differential Equation solvers are implemented. For this research a Euler solver, a Runge-Kutta solver and a Verlet solver were implemented. A description of these solvers is given in sections **2.1.1**, **2.1.2** and **2.1.3**. A description about their implementation is given in **sections 3.1.1-3.1.3**.

### 2.1.1 Euler Solver

The first solver that was implemented in this research is Euler's method. Euler's method is the most basic first-order method to solve differential equations [4]. Euler's method provides the approximation of the solution of a differential equation. It calculates a new function-value based on an already known value and the timestep. The formula for Euler's method is [4]:

$$w_{i+1} = w_i + h * f(t_i, w_i)$$

where $w_{i+1}$ is the new approximated value, $w_i$ is the initially known value, $h$ is the used timestep and $f(t_i, w_i)$ is the evaluated function at time $t_i$. To achieve the next time, we can use the formula $t_{i+1} = t_i + h$.

### 2.1.2 Runge-Kutta Solver

The second implemented solver is the fourth-order Runge-Kutta solver. It uses a similar approach as Euler's method to approximate the solution, except it uses more intermediate calculations. This increases the accuracy by a lot compared to Euler's method. Since the Runge-Kutta method is a fourth-order differential equation solver, it has a global error of $O(h^4)$, compared to Euler's method's error of O(h) [5]. The equation for the fourth-order Runge-Kutta method is:

$$k_1 = f(t_i, w_i)$$

$$k_2 = f\left(t_i + \frac{h}{2}, w_i + h * \frac{k_1}{2}\right)$$

$$k_3 = f\left(t_i + \frac{h}{2}, w_i + h * \frac{k_2}{2}\right)$$

$$k_4 = f(t_i + h, w_i + h * k_3)$$

$$w_{i+1} = w_i + \frac{1}{6} * h * (k_1 + 2 * k_2 + 2 * k_3 + k_4)$$

again, $w_{i+1}$ is the new approximated value, $w_i$ is the original value, $h$ is the timestep and $f(t_i, w_i)$ the evaluated function at time $t_i$.

### 2.1.3 Verlet Solver

To further increase the accuracy of our approximation, a Verlet differential equation solver is implemented. The approach for the verlet method is a bit different, compared to both the Euler and Runge-Kutta solvers. The verlet solver does not make use of function values to calculate the approximation, but does this just with previous approximated values. The equation for the Verlet method is:

$$w_{i+1} = 2 * w_i - w_{i-1} + a_i * h^2$$

where $w_{i+1}$ is the new approximated value, $w_i$ and $w_{i-1}$ are the previous calculated values, $a_i$ is the acceleration in the i-th step and $h$ is the used timestep. Because the verlet solver requires the two previous calculated values, it has to be bootstrapped for the first two calculations. In this research the fourth-order Runge-Kutta solver described in **section 2.1.2** is used for that.

## 2.2 Visualization of the Solar System

In order to demonstrate the models established, as well as the physics engine - capable of computing paths, states and landing motions - a graphical user interface (GUI) was developed.

Following the chronological evolution of our GUI, it initially started with a simple double scene (the Intro Scene and the Visualizer Scene). The Intro Scene was created with no extra purpose than to welcome the user into the simulation, for which matter it has only the “Begin!” button. Once clicked, the user switches to a second scene, the Visualizer Scene. This scene presents a 2D simulation that effectively fulfils the mentioned goals of developing this visualization. The rocket's take-off with Titan as the destination, as well as the overall Solar System motion behaviour, begins by pressing the "Launch Probe!". At any time, if the Visualizer Scene’s "Exit Simulation!" button is pressed, the simulation ends, and the user exits the program.


*Figure 1 - Intro Scene*


*Figure 2 - Visualizer Scene*

Since one of our goals was to have a working model of all the planets in the Solar System, including the far distanced Uranus and Neptune, a zoom and pane functionality were implemented. This had the intent of providing the user the option of freely zooming and moving around the simulation (this last feature with the aid of the “xy” sliders), enabling the visualization of any point of the Solar System.

Continuing the chronological progress, in the final phase of the project, a few more goals were established: have a landing scene and merge that new scene in a seamless way with the ones already developed.

Thus, as the animation of the model advances (after the clicking of the “Launch Probe!” button), once the rocket has landed on Titan the user is shown two options. The first option is to press the "Observe Landing!" button which takes the user to a newly added scene, the Landing Scene. In this scene, it is possible to see in detail how the lander behaves/lands on Titan. The second option is to press the "Return to Earth!" button, which makes the user stay in the very same scene, as you see the remaining of the animation of the Solar System, as the probe makes its return to Earth.

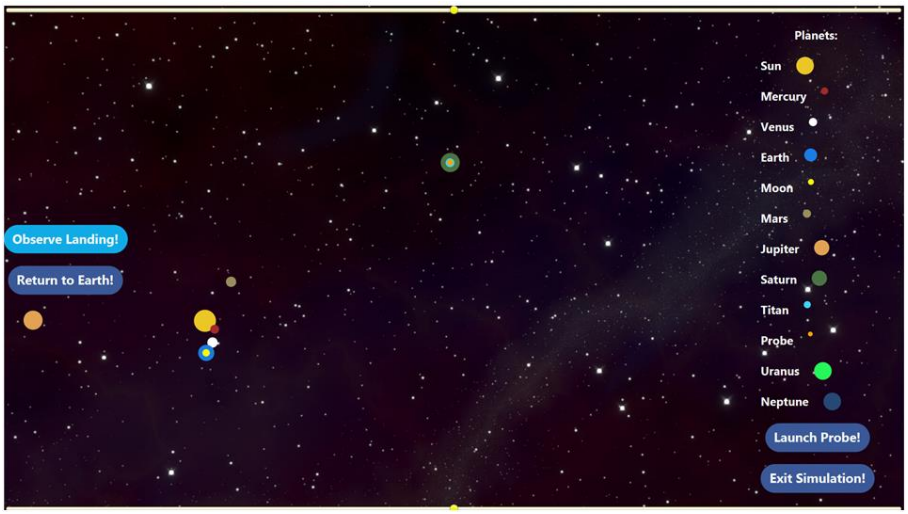



*Figure 3 - Visualizer Scene with newly added options on the left, upon reaching Titan*

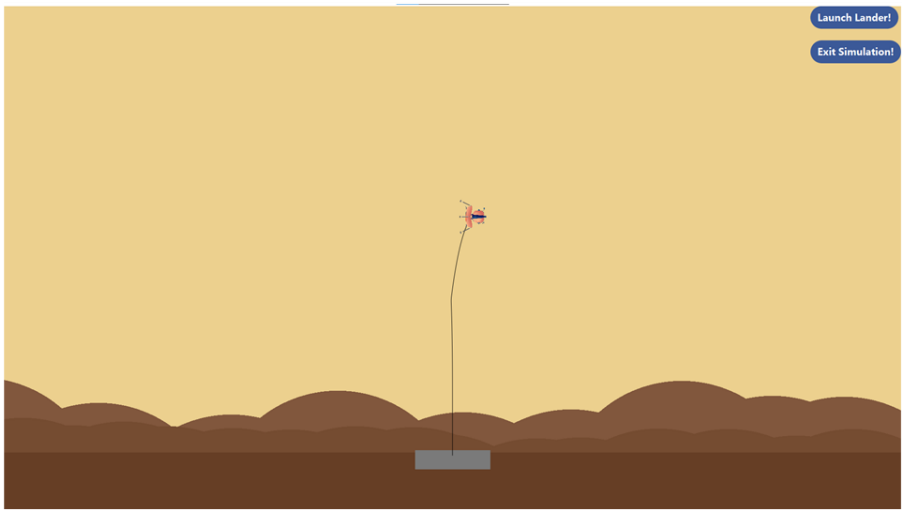



*Figure 4 - Landing Scene*

## 2.3 Trajectory to Titan

The main part of this space mission is getting to and into a geostationary orbit around titan. This is achieved by implementing an engine for the shuttle. This engine can perform a series of burns. During a burn, a force is applied to the shuttle for a given amount of time in a specified direction. This way we can steer the shuttle into the right direction. To get the burns that we need to get to titan we completed the journey in steps. We first wanted to leave earth and get in an orbit around the sun, then we wanted to perform the transfer burn to get to titan, when we are at titan we slow down to get into orbit. When in orbit we can perform some engine burns to correct our orbit. Our last burn is our return from titan orbit back to earth. During the transfer to titan and back we perform some

corrections to be more precise. One big correction we had to perform was at our burn to get in orbit around titan. Here we cancelled all our velocity on the z-axis. To prevent a non-equatorial orbit to make it easier for us to return to earth. For each step we used an iterative process:

- first plan a burn with a guess on the direction and force.
- then use the relative coordinates of the lander compared to different objects in the solar system to make plots like the ones in figure 5 and 6.
- Based on the information shown in these plots we would figure out what to change about our burns, for example if were too far left or right we change our angle, if were over/undershooting our target we change the force we use.
- Then we repeat these steps until we get the desired result for this burn and continue to the next one.

This process was made easier by using the previously mentioned correction burns during the transfer stages of the trajectory. By using a second burn later in the process we noticed that we do not have the need for a high accuracy for the first burn. Resulting in us needing less iterations in our process to get the desired results.

We did not have to worry much about the timing of our burns. Most of the time our engine was strong enough to produce enough force for one timestep to get the velocity that we needed.

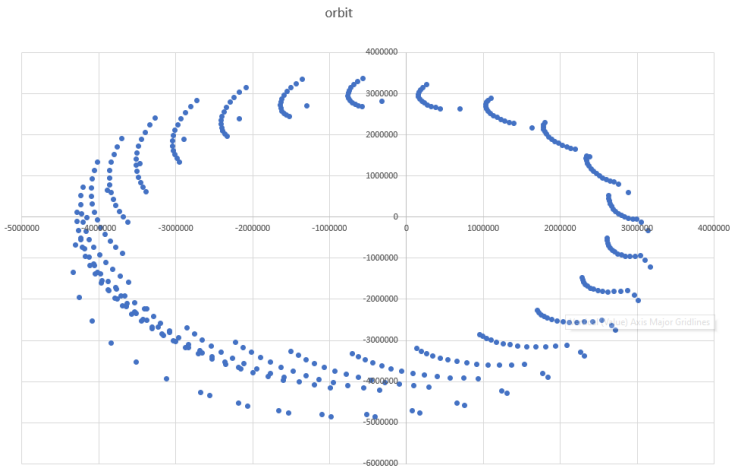



*Figure 5 – Orbit around titan*

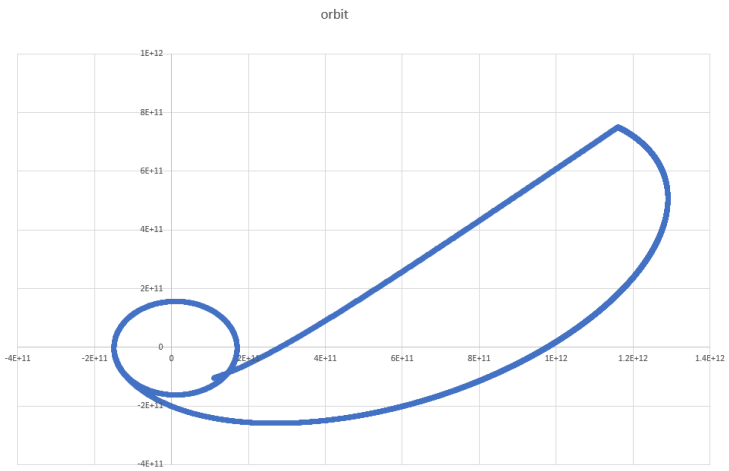



*Figure 6 – Total trajectory relative to the sun*

## 2.4 Landing on Titan

In this part of the program there is a simulation of the solar system, which includes the Sun, Titan and the eight planets. The simulation is based on the gravitational attraction between the planets, as described in Newton's law of universal gravitation. The acceleration of the planets is obtained by numerical integration, specifically by the Runge-Kutta fourth-order method. To land the module on titan two controllers are implemented.

### 2.4.1 Open-loop controller

The open-loop controller works with a pre-existing input list of when the thrusters should burn (to get it into position). This means that it does not operate in real time. To start we establish an initial position and velocity and adjust our functions accordingly so that we can reach the landing site.

### 2.4.2 Closed-loop controller

The second controller is a feedback controller, meaning that it does not take predefined input for the engine burns and that is the main difference between it and the open-loop controller. It depends on feedback to adjust the module's acceleration and which direction it should be headed. It repeats this cycle of feedback and adjustment to get closer and closer to the target landing site.

### 2.4.3 Stochastic model for the wind

To simulate the effect of the wind on our spacecraft, we first had to make an approximation of the wind on titan at different altitudes. We used the data from figure 5 in [6] to approximate a function which is a combination of linear functions.

The approximation we use is:

$$f(x) = 0.00053 * x, x < 65000$$

$$f(x) = -0.00375 * x + 278.75, \quad 65000 \leq x < 73000$$

$$f(x) = 0.0020 * x - 142.553, \quad 73000 \leq x < 120000$$

$$f(x) = 100, 120000 \leq x$$

Which will create the graph in figure 5:

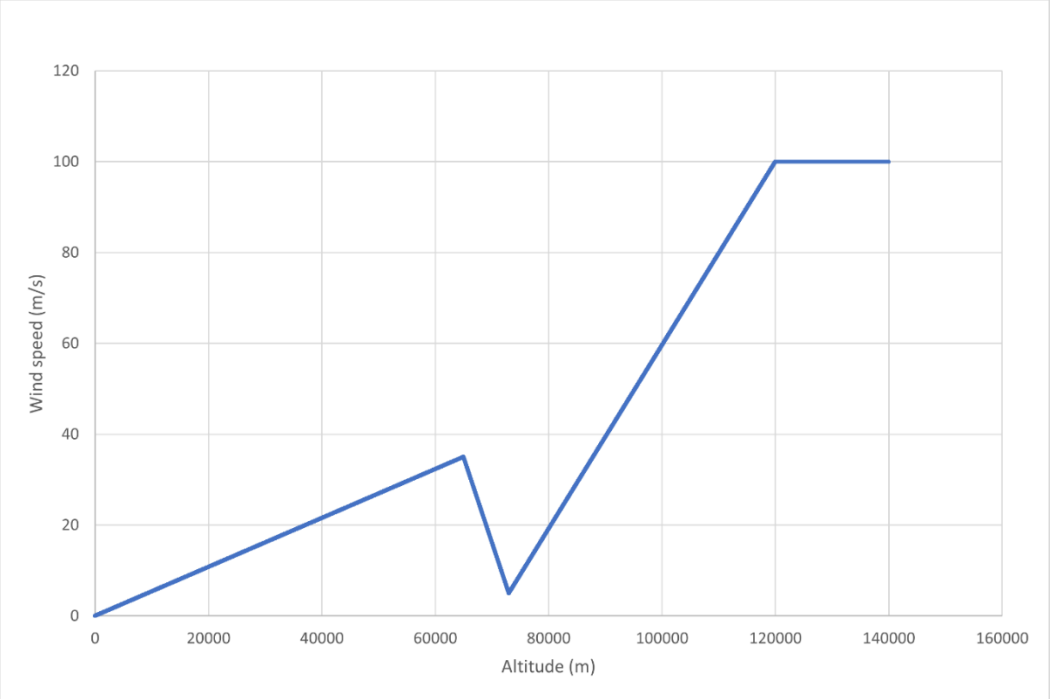



*Figure 7 - The approximated speed of the wind at altitudes between 0 and 140,000 meters*

To make our wind model stochastic we added three random elements:

- Random starting direction
- Random variation in absolute wind speed
- Two random heights at which the wind reverses

For the random starting direction, we have a 50% chance for the wind to be pointing in the positive x direction and 50% chance to be pointing to the negative x direction.

To get the random variation of absolute wind speed we multiply our windspeed by a random factor between $1 - d$ and $1 + d$ where $d$ is the maximum percentage the wind can differ from the original value from our approximated function. To get these results we first made a Perlin noise function that takes the altitude as input. This function is made by choosing random values between -1 and 1 for each 1000 meters. To get the value of the noise value at an altitude $x$ we get the two random values of the two altitudes that $x$ is between. After which we use cosine interpolation to interpolate the value of the function at the point $x$, creating a graph such as in figure 6. Then we multiply the Noise value by our maximum difference percentage $d$ to get our random percentage. After which we increase or decrease the windspeed by this percentage.

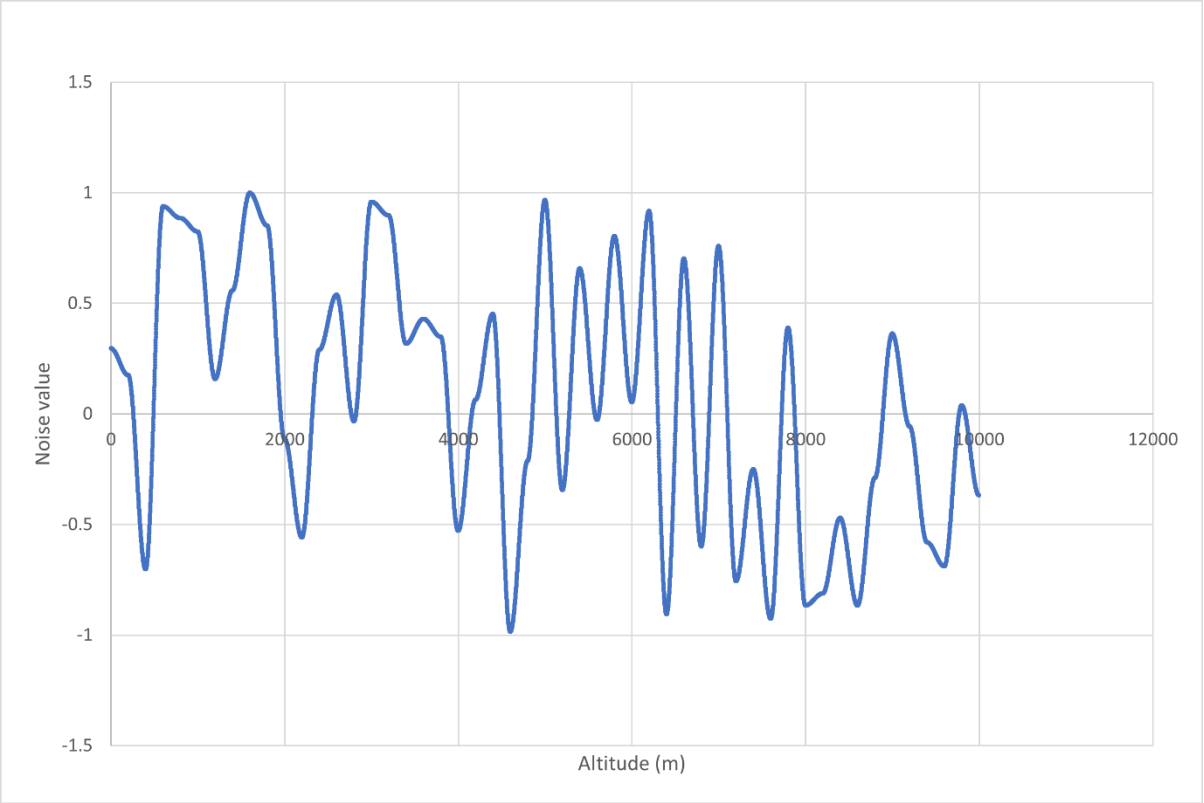



*Figure 8 - Noise value at the first 10,000 meters*

During the descent of the Huygens mission to titan the wind reversed direction twice [7]. To simulate this, we choose two random altitudes at which the wind reverses, and if the altitude is between these two values, we reverse the wind.

Applying all these methods to our original function we get results like the graph shown in figure 7.

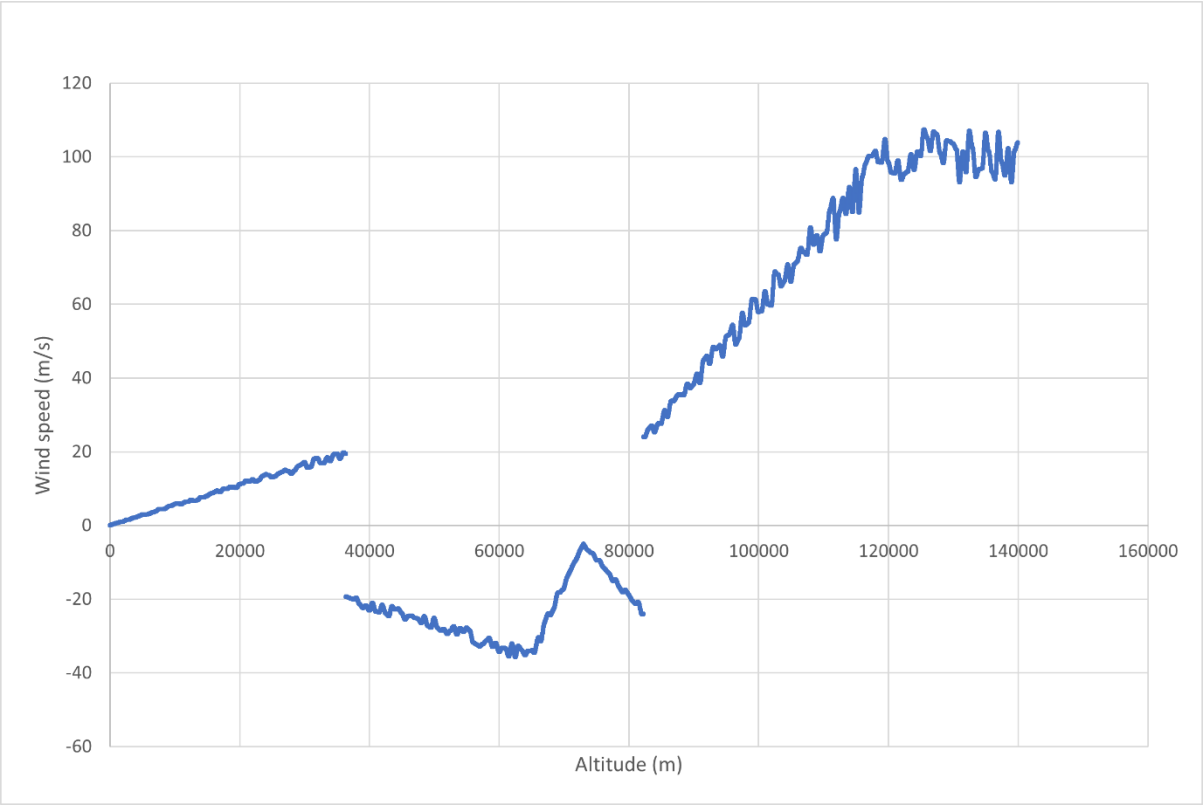



*Figure 9 - Approximated wind speed with random elements between 0 and 140,000 meters*

# 3. Implementation

## 3.1 Solvers

### 3.1.1 Euler's method

At the start of the method, the final time is divided by the step size to acquire the amount of states. The first state is set equal to the initial state, and after that the rest of the states is being calculated using the formula described in **section 2.1.1**. In order to calculate the new state, the acceleration is needed. A function class was implemented for this. This class calculates the accelerations of all celestial bodies in the solar system. These accelerations are stored and returned in a Rate object at the end of the calculation and then used to calculate the new position.

### 3.1.2 Runge-Kutta solver

The general implementation of the Runge-Kutta solver is similar to the one of Euler's method, except for the extra intermediate evaluations described in **section 2.1.2**. Because we use a Singleton design pattern for our Solar System implementation, the position of the planets are updated after every intermediate calculation. Therefore, the planets need to be reset to their original positions after every evaluation. This was achieved by making a copy of the positions at the start of the method, and setting the positions back after every calculation.

### 3.1.3 Verlet solver

The Verlet solver implementation starts with calculating the amount of states, based on the final time and the time step, and creating an array of the according size. The first state is set equal to the initial state, given in the method input. The second state is bootstrapped using the Runge-Kutta method described in **section 3.1.2**. After this, the rest of the states are calculated using the formula described in **section 2.1.3**. Again, the Function and Rate classes are used to acquire the acceleration needed for the calculations as described in **section 3.1.1**.

## 3.2 Engine

### 3.2.1 Trajectory engine

To be able to leave earth and get to titan's orbit and back we added an engine to apply force to the rocket. To calculate everything for the rocket engine we used the Rocket thrust equation [8]. In this research we use the following data for the engine: a maximum force of 30MN and an effective exhaust velocity of 20 km/s.

To fly to Titan and back we have multiple pre-planned burns we execute. For these burns we specify the force of the engine, the starting time, the end time, the effective exhaust velocity (which is constant in our case) and the direction of the force.

When we need the acceleration in our simulation, we check using the current time If there is a burn being performed. If that is the case, we divide the Force by the total mass of the rocket to get the acceleration and return this. To calculate the mass of the rocket we first calculate the fuel mass. We start our journey with 1800t of fuel.

To calculate the total burned fuel we use the previously mentioned rocket equation to calculate the fuel consumption using the force and effective exhaust velocity for each planned burn. Per burn we multiply this fuel consumption by the total time this burn is being or has been executed to get the total fuel consumed by that burn. Subtract the fuel consumed by the engine burns from the starting fuel mass and we get the current fuel mass. To get the total mass of the rocket we must add the dry mass of the rocket (78000 kg) and the lander (6000 kg) to the fuel mass. Now we have the acceleration we just apply this acceleration in the direction specified for the burn being executed.

### 3.2.2 Lander engine

For the lander we use methods that returns acceleration instead of force. At the start of the process, we decided on a maximum acceleration of 2,7 m/s^2. Which is 90% of the maximum acceleration of the apollo lander divided by its maximum mass [9][10]. If we do not exceed this maximum acceleration, we do not have to worry about the landers mass. At each timestep the current state and the current time are passed to the controller. This controller will decide what accelerations to apply at each timestep. Next to the main engine we have an engine that can apply a rotational force, and just like the other engine we use acceleration in our calculations instead of force. This rotation is used to point our main engine in the correct direction because in this simulation we can not just simply input a desired direction.

## 3.3 Wind

To calculate the force caused by the wind we use the force of drag on the lander. We calculate this by using the drag equation: $F_D = \frac{1}{2}\rho v^2 C_D \mathrm{A}$. In this formula $F_D$ is the force we want to calculate, ρ is the density of the fluid we are moving through (in this case the atmosphere of titan), $v$ is the relative velocity at which we are moving, $C_D$ is the drag coefficient of the lander and $\mathrm{A}$ is the cross-sectional area of the object. We assume the lander to be a perfect sphere with an assumed radius of 3.49 meter which is about half of the width of the lander used in the Apollo 11 mission [11]. We also assume the lander to have a constant drag coefficient of 0.47. This way we do not have to worry about the shape of our lander that changes based on the angle and can just use a constant $\mathrm{A}$ and $C_D$ instead. To calculate the density of the atmosphere we approximate the density using an exponential function based on the data from the recommended model from [12]. Resulting in the function for the density: $\rho(x) = 1.5743\, e^{-0.000034x}$ for the altitude $x$. For $v$ we use the relative velocity of the lander with respect to the wind. This system also works without any wind speed, which will result in air drag in our simulation. Therefore, we add this force calculation to the simulation even without the wind.

## 3.4 Controllers

### 3.4.1 Open-loop controller

The working of the open-loop controller is relatively the same as our implementation of the Trajectory engine. There are multiple pre-planned actions to be performed at specific times. However, for the lander we do not return the force using this implementation but the acceleration on the lander. So instead of storing the force and the effective exhaust velocity we store the acceleration. Since our lander uses rotation, we cannot use a direction vector to point our engine, but we have to move our lander using a different acceleration that applies a rotational effect. This acceleration is also pre-planned together with the engine acceleration. In the end we have to specify the two different accelerations and the times at which to apply these to land our spacecraft.

To land we first try to get above $x = 0$ with no horizontal velocity. After we accomplished this we use our engines at the last possible moment to decelerate such that we reach a vertical velocity of aero when we touch the ground.

### 3.4.2 Closed-loop controller

The closed loop controller uses the current state of the lander to decide on what accelerations to apply at each timestep. We try to do the same process as the open-loop controller. The first part of the descent we try to get to $x = 0$, and after we drop below a height of 22215 meter we try to land no matter what x-position the lander has. To get above the centre of the landing pad the controller uses the state to determine where the lander needs to move. If were to the right of the centre we want to move left, and if we are to the left of the centre we want to move right. Depending on the current velocity and rotation of the lander the controller will apply acceleration to perform this movement. When the lander is approaching the centre it will try to approach it slower while also using a lower acceleration. This way we try to be more accurate when landing to prevent overshooting the centre. Then when below the height threshold it will point the craft upwards and then based on the distance to the ground and its current vertical velocity it will decide to slow down or not. Trying to decelerate enough to not be moving when touching the ground.

# 4. Experiments

To answer the problem statement, a set of experiments is being done. The research question and the specifics of each of the experiments are discussed next.

1) ***What impact does a variation of wind speed have on the landing success rate of both the open-loop and the closed-loop controller?***

In this first experiment wind, generated by the stochastic model described in **section 2.4.2**, is added to the simulation. Using different wind speeds, 10.000 simulations are run per controller too see the landing success rate for each of the deviation percentages. We determine the success of a landing by comparing the state of the lander when touching the ground to the error bounds described in the project manual.

2) ***What are the effects of the percentage of deviation on wind speed in our Stochastic wind model?***

To answer this question, a number of different percentages of deviation are used in the stochastic model. For these percentages, the paths of 50 different simulations are displayed in a graph.

3) ***Which of the implemented solvers give more accurate results?***

In this experiment the implemented differential equation solvers described in **section 2.1.2 and section 2.1.3** are being tested on their accuracy. The Euler's method described in **section 2.1.1** will not be tested in this experiment, since by definition of the first- and fourth-order methods, the first-order method will be less accurate than the fourth-order method. To draw conclusions about the accuracy, the calculated positions from the differential equation solvers (using different timesteps) are being compared to the actual positions of the celestial bodies acquired from the NASA Horizons website [13].

# 5. Results

1) Percentage of runs where the variable ends within the error bounds for both controllers tested for all deviation percentages. See **Appendix B** for the excel file with the raw data of our results.

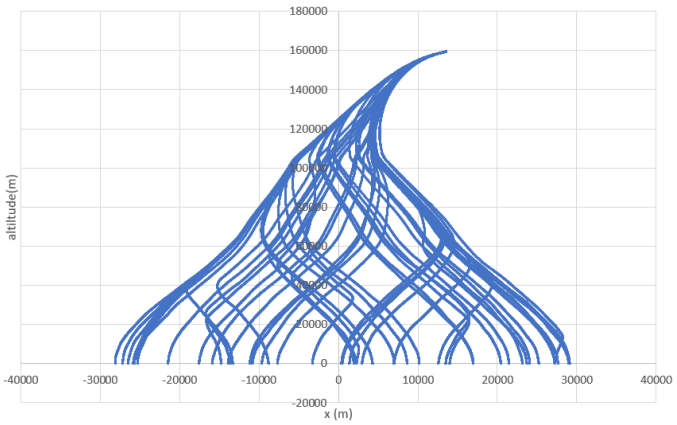
| OPEN LOOP: | X-POS | ANGLE | X-VEL | Y-VEL | ANGLE-VEL |
|---|---|---|---|---|---|
| **50% DEVIATION** | 0.00% | 100.00% | 0.48% | 0.00% | 100.00% |
| **25% DEVIATION** | 0.00% | 100.00% | 0.38% | 0.00% | 100.00% |
| **10% DEVIATION** | 0.00% | 100.00% | 0.48% | 0.00% | 100.00% |
| **0% DEVIATION** | 0.00% | 100.00% | 0.43% | 0.00% | 100.00% |
| **AVERAGE** | 0.00% | 100.00% | 0.44% | 0.00% | 100.00% |

*Table 1 – success percentages for the open-loop controller*

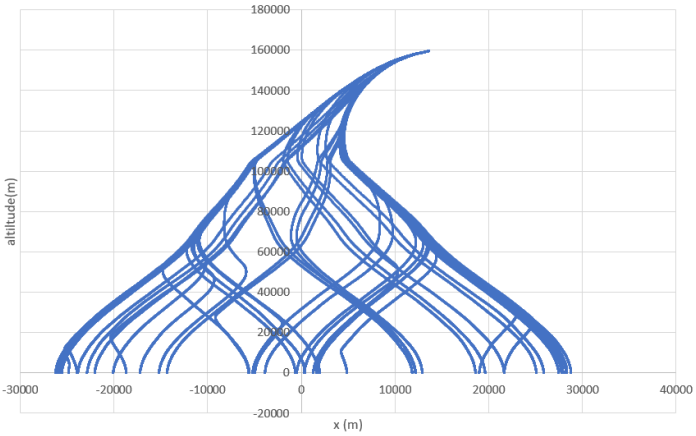
| FEEDBACK: | X-POS | ANGLE | X-VEL | Y-VEL | ANGLE-VEL |
|---|---|---|---|---|---|
| **50% DEVIATION** | 0.00% | 100.00% | 17.94% | 80.42% | 100.00% |
| **25% DEVIATION** | 0.00% | 100.00% | 21.20% | 80.64% | 100.00% |
| **10% DEVIATION** | 0.00% | 100.00% | 25.81% | 82.35% | 100.00% |
| **0% DEVIATION** | 0.00% | 100.00% | 33.75% | 85.58% | 100.00% |
| **AVERAGE** | 0.00% | 100.00% | 24.68% | 82.25% | 100.00% |

*Table 2 – success percentages for the feedback- controller*

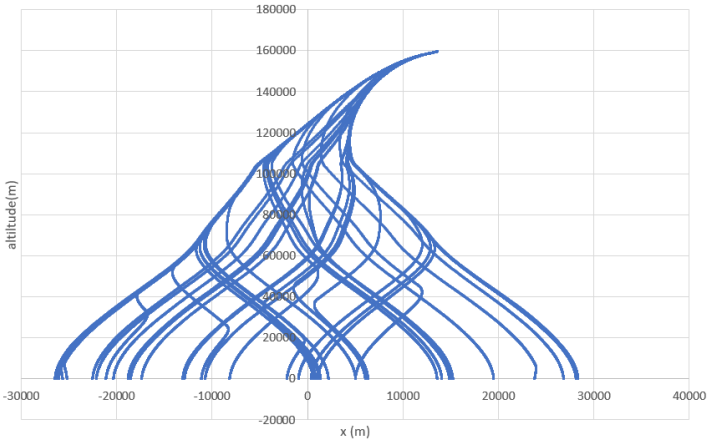
2) The following graphs show the paths of the lander from 50 runs using our open-loop controller using different percentages of maximum deviation.



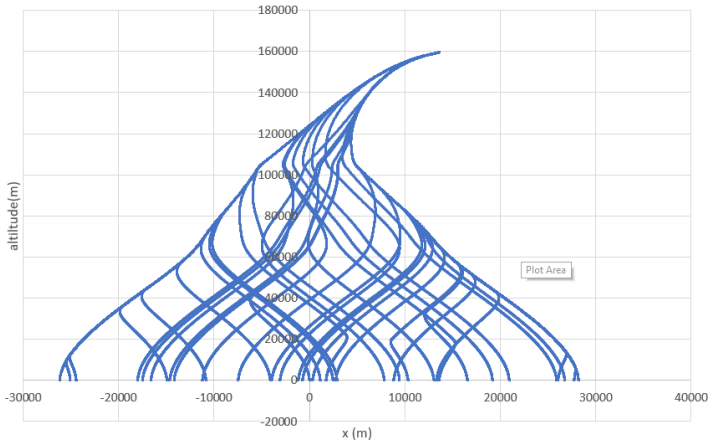

*Graph 1: Paths of the lander using 50% deviation in wind speed*




*Graph 2: Paths of the lander using 25% deviation in wind speed*

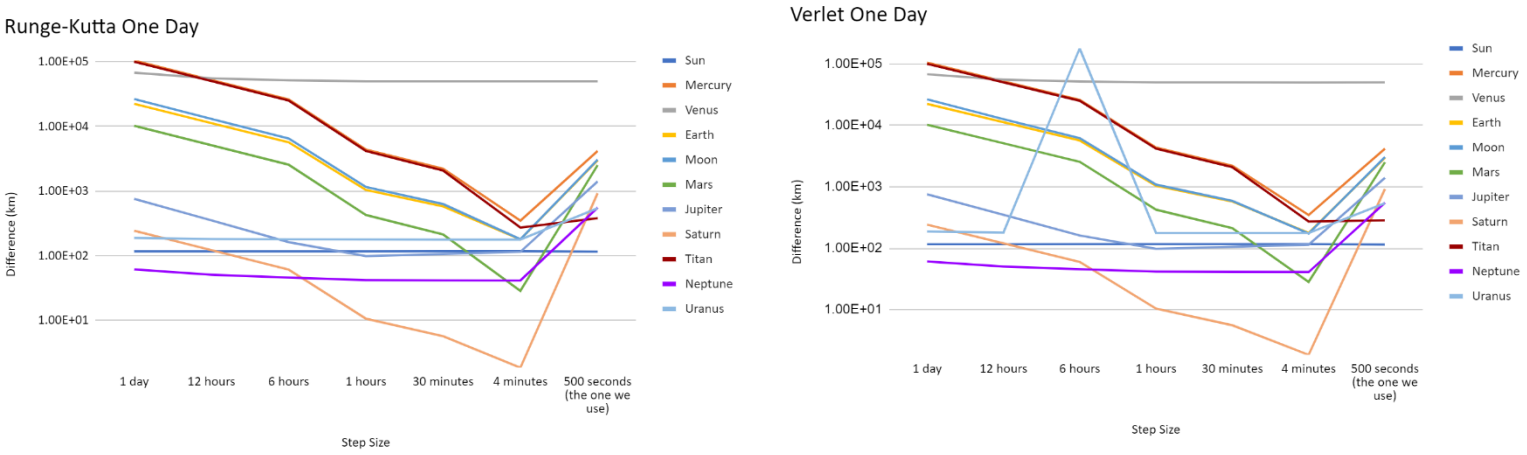



*Graph 3: Paths of the lander using 10% deviation in wind speed*

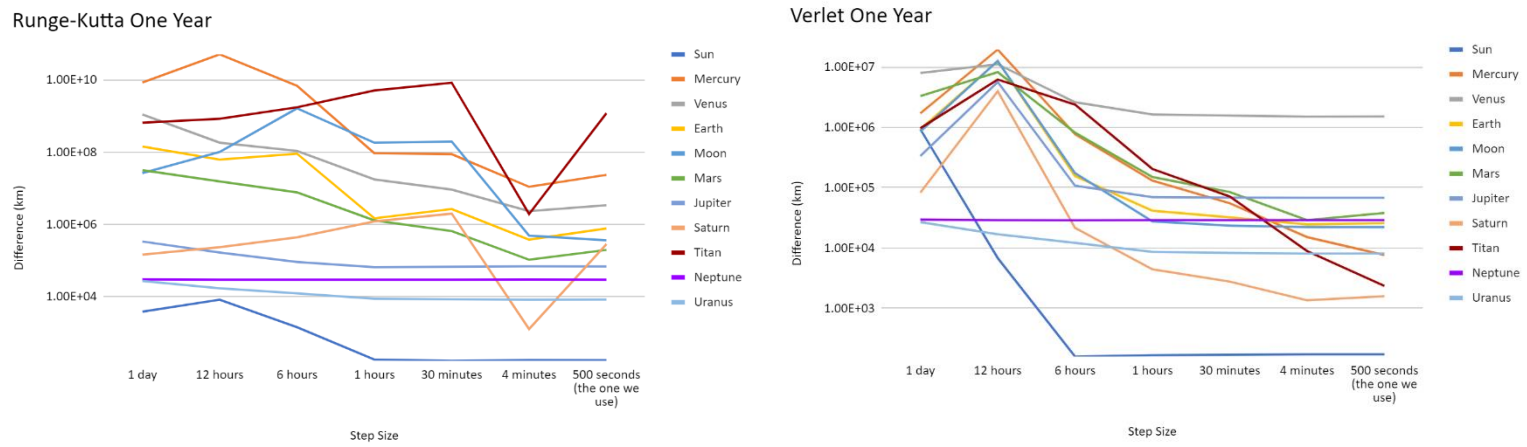



*Graph 4: Paths of the lander using 0% deviation in wind speed*

3) Below are the graphs showing the difference between the positions calculated by the solvers after 1 day and 1 year and the actual positions. The tables with the raw data are displayed in **Appendix A**.




***Graph 5:*** *Difference in km between calculated positions after 1 day and the actual planet positions according to NASA.*




***Graph 6:*** *Difference in km between calculated positions after 1 year and the actual planet positions according to NASA.*

## 6. Discussion

To answer the research questions proposed in **section 1.2.1**, 3 experiments were performed. In this section the results are discussed.

Starting with the results from experiment 1 (**see table 1 and 2, section 5**).
For both controllers we have noticed that they share a common problem, the horizontal position (x-position). Both controllers when subjected to the wind are rarely able to get close to $x = 0$. None of the runs performed by both controllers landed within the error bounds of the x-position. The feedback controller can get close to the centre point more often than the open-loop controller but still, most of the time, is not able to land within the error bounds.
However, even though horizontal velocity (x-velocity) is handled better than the x-position, it also is a problem. The open loop controller is on average only able to land with correct x-velocity 0.44% of the time. Our implementation of the open loop controller uses air drag to slow down and land

exactly on the correct x-position, if there is any wind it is not able to correct itself and will be moving faster than expected. The few runs where we land within the error bounds for the x-velocity we expect that the total velocity the lander has gathered adds up to near zero due to the wind cancelling its own effect by changing direction. Resulting in a horizontal velocity low enough to be within error bounds. The feedback controller is more successful in landing with low enough horizontal velocity, it stays within the error bound on average 24.68% of the time. We think this value is higher than the open loop controller since the feedback controller tries to limit its maximum velocity when moving. This will result in a lower speed when trying to land and the lander will reach a lower speed due to air resistance sooner because of this lower speed.

Next to that we noticed that our controllers are good at handling their rotation. Since rotation is not affected by the wind in our simulation the open-loop controller with its pre-programmed rotational burns is able to land with the exact same angle every time, which is within the error bounds. The feedback controller is not pre-programmed but when landing we are accurate enough with our rotation to stay within the error bounds in all our test runs.

For the vertical velocity (y-velocity) of the lander there is a big difference between the two controllers. The open loop controller was not able to land with a y-velocity within the error bounds in all our test runs. The feedback controller, however, can land within error bounds on average 82.25% of the time. We think this is the case due to how the final landing burn is performed. The open loop controller has a pre-programmed time at which it will perform the landing burn, however we have noticed that due to the wind it will reach the surface slower and thus will always burn its engine too soon to be able to land. There are a few times where the lander was close to landing with the desired speed, but it was not able to accomplish this. The feedback controller can determine the point at which it will fire its engine based on the current velocity and the distance to the surface, resulting in a reasonably accurate timing of the engines which makes it possible for the controller to land safely. During our testing we noticed that the times the feedback controller is not able to land it is because it did not fire its engine on time because it was struggling with its rotation. The controller only fires its engine when pointed in the correct direction and if it takes too long to rotate the lander will be too late to slow down and will not be able to land safely. The reason why the feedback controller struggles with the rotation is unknown to us.

Secondly, the results from experiment 2 (**see table 1 and 2 and see graph 1 to 4, section 5**). We noticed that the deviation on the wind speed does not affect the resulting landing spot of the open loop controller a lot. All of the test performed stay within the 30000 meter distance to the centre. However, one can see more variation in the paths for the 50% deviation compared to the 0 % deviation. The large difference between these paths is mostly caused by the difference in wind direction of the different path. The deviation does seem to have an effect but its not as strong as the change in wind direction. Where the deviation does have an impact is in the percentage of landings where the use of the feedback controller results in a velocity within our error bounds. The lower the deviation the higher the number of landings with a desired velocity. The exact reason why this effect occurs is unknown, but we have a suspicion that our feedback controller cannot handle higher speeds of the lander very well. If the deviation increases, the maximum windspeed also increases and thus the lander can move faster.

Finally, the results from experiment 3 (**see graph 5 and 6, section 5**) revealed that the Verlet solver is more accurate when it comes to calculate positions than the fourth-order Runge-Kutta solver. Both for one day and one year we can see that a decrease in time step results in an increase in accuracy. After one day, the solvers produce very similar results. Except for one outlier in the Verlet calculations, the error of both solvers is almost identical. When we look at the error after one

year, a clearer difference can be seen. This time the Verlet solver is a lot more accurate than the Runge-Kutta solver. Where the Verlet solver can calculate all positions with an error of maximum 1E+7, the Runge-Kutta solver can only do this with a maximum error of 1E+10.

# 7. Conclusions

This report gives an analysis of all the work that was done throughout Project 1-2. It explained the implementation of a program that simulates a space mission to one of Saturn's moons Titan. During the mission, the probe has to enter an orbit around Titan, and a landing module has to be deployed. This landing module has to land on Titan's surface using an open- or a closed-loop controller. After deploying the landing module, the probe has to return safely back to earth. To achieve this task several differential equations, a graphical user interface, an open- and closed-loop controller and a stochastic wind model were implemented.

Several conclusions can be drawn from the experiments discussed in this report. Regarding the controllers, the open-loop controller does not work with wind, the only way that it could work is if the wind forces would cancel themselves throughout the whole path down of the lander. The closed-loop on the other hand, is more effective, being able to give much better results for velocities. Both controllers are not completely able to successfully land, but if we would not pay attention to the x coordinate, or have a bigger error range, the closed-loop controller would be able to land in more cases. Secondly, it has been observed that a decrease in deviation percentage in wind speed in our stochastic wind model leads to an increase in accuracy for the x and y velocity for the closed-loop controller, but it does not affect the open-loop controller, the changes that exist being from the randomness of wind. Furthermore, for the calculation of planet positions, the Verlet solver produces the most accurate results and is therefore used in the final simulation.

Overall, this program was able to simulate a trajectory where the probe enters a geostationary orbit around Titan. The landing was achieved using the open-loop controller. Both the trajectory, as well as the landing is visualised in a user interface. The solvers and the controllers were tested in a set of experiments. For further research, we would look into the feedback controller to see if we can improve the problems with the x-position, x-velocity and rotation. On top of that, the trajectory from Titan back to Earth could be improved. At the moment the probe will use all fuel left to launch itself straight back to Earth, making the trajectory time-efficient. Using a different technique could lead to a more fuel-effective return.

# References

[1] International Astronomical Union (2009). Surface features on Titan form like Earth's, but with a frigid twist. Retrieved June 25th, 2019 from https://www.iau.org/news/pressreleases/detail/iau0915/ .

[2] ESA (2009). Titan and the Origin of Life on Earth Retrieved June 25th, 2019 from https://www.esa.int/esapub/bulletin/bullet92/b92owen.html

[3] Brown, H., Lebreton, J., Waite, J. (2009). Titan from Cassini-Huygens.

[4] Nurujjaman, Md. (2020). Enhanced Euler's Method to Solve First Order Ordinary Differential Equations with Better Accuracy. 10.5281/zenodo.3731020.

[5] Tan, D., Chen, Z., On a general formula of fourth order Runge-Kutta method. Journal of Mathematical Sciences & Mathematics Education (2012) Vol. 7 No. 2.

[6] Bird, M., Allison, M., Asmar, S. *et al.* The vertical profile of winds on Titan. *Nature* **438,** 800–802 (2005). https://doi-org.ezproxy.ub.unimaas.nl/10.1038/nature04060

[7] ESA (2009). The way the wind blows on Titan Retrieved June 15th, 2021 from https://www.esa.int/Science_Exploration/Space_Science/Cassini-Huygens/The_way_the_wind_blows_on_Titan

[8] Rocket Thrust Equation, Glenn Research Centre, NASA http://www.grc.nasa.gov/WWW/K12/airplane/rockth.html

[9] Fisher, S. C., & Rahman, S. A. (2009). *Remembering the giants: Apollo rocket propulsion development*. National Aeronautics and Space Administration, NASA History Division, Office of External Relations.

[10] *NSSDCA Master Catalog*, NASA, Apollo 11 Lunar Module / EA SEP, **NSSDCA/COSPAR ID:** 1969-059C, https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1969-059C , 27 April 2021, Accessed 20/06/2021

[11] Orloff, R. W. (2001). *Apollo by the numbers: a statistical reference*. Government Reprints Press.

[12] Yelle, R. V., Strobel, D. F., Lellouch, E. & Gautier, D. Engineering Models for Titan's Atmosphere 243–-256 (ESA SP-1177, European Space Agency, Noordwijk, 1997).

[13] *HORIZONS: Solar System Dynamics*, NASA Jet Propulsion Laboratory, https://ssd.jpl.nasa.gov/horizons.cgi

# Appendix

**Appendix A:** Tables with position data from the solvers

| Step Size | 1 day | 12 hours | 6 hours | 1 hours | 30 minutes | 4 minutes | 500 seconds (the one we use) |
|---|---|---|---|---|---|---|---|
| *Sun* | 1.16E+02 | 1.16E+02 | 1.16E+02 | 1.16E+02 | 1.16E+02 | 1.16E+02 | 1.15E+02 |
| *Mercury* | 1.04E+05 | *5.20E+04* | *2.60E+04* | *4.36E+03* | *2.20E+03* | *3.46E+02* | *4.14E+03* |
| *Venus* | 6.74E+04 | 5.52E+04 | 5.14E+04 | 4.98E+04 | 4.97E+04 | 4.96E+04 | 4.98E+04 |
| *Earth* | 2.22E+04 | 1.12E+04 | 5.65E+03 | 1.04E+03 | 5.76E+02 | 1.77E+02 | 2.96E+03 |
| *Moon* | 2.64E+04 | 1.30E+04 | 6.48E+03 | 1.15E+03 | 6.25E+02 | 1.79E+02 | 3.04E+03 |
| *Mars* | 1.02E+04 | 5.09E+03 | 2.55E+03 | 4.24E+02 | 2.12E+02 | 2.84E+01 | 2.50E+03 |
| *Jupiter* | 7.54E+02 | 3.50E+02 | 1.61E+02 | 9.80E+01 | 1.06E+02 | 1.14E+02 | 1.40E+03 |
| *Saturn* | 2.42E+02 | 1.21E+02 | 6.06E+01 | 1.06E+01 | 5.64E+00 | 1.84E+00 | 9.19E+02 |
| *Titan* | 9.96E+04 | 5.02E+04 | 2.50E+04 | 4.14E+03 | 2.07E+03 | 2.71E+02 | 3.78E+02 |
| *Neptune* | 6.10E+01 | 5.04E+01 | 4.55E+01 | 4.16E+01 | 4.13E+01 | 4.10E+01 | 5.49E+02 |
| *Uranus* | 1.88E+02 | 1.80E+02 | 1.78E+02 | 1.77E+02 | 1.77E+02 | 1.77E+02 | 5.40E+02 |

***Table 1:*** *Difference in km between calculated position after 1 day using Runge-Kutta solver and NASA Horizons actual positions.*

| Step Size | 1 day | 12 hours | 6 hours | 1 hours | 30 minutes | 4 minutes | 500 seconds (the one we use) |
|---|---|---|---|---|---|---|---|
| *Sun* | 1.16E+02 | 1.16E+02 | 1.16E+02 | 1.16E+02 | 1.16E+02 | 1.16E+02 | 1.15E+02 |
| *Mercury* | 1.04E+05 | *5.18E+04* | *2.59E+04* | *4.34E+03* | *2.20E+03* | *3.48E+02* | *4.16E+03* |
| *Venus* | 6.74E+04 | 5.52E+04 | 5.14E+04 | 4.98E+04 | 4.97E+04 | 4.96E+04 | 4.98E+04 |
| *Earth* | 2.22E+04 | 1.12E+04 | 5.66E+03 | 1.04E+03 | 5.76E+02 | 1.77E+02 | 2.97E+03 |
| *Moon* | 2.64E+04 | 1.26E+04 | 6.16E+03 | 1.08E+03 | 5.90E+02 | 1.73E+02 | 3.04E+03 |
| *Mars* | 1.02E+04 | 5.09E+03 | 2.54E+03 | 4.24E+02 | 2.12E+02 | 2.84E+01 | 2.50E+03 |
| *Jupiter* | 7.54E+02 | 3.50E+02 | 1.61E+02 | 9.80E+01 | 1.06E+02 | 1.14E+02 | 1.40E+03 |
| *Saturn* | 2.42E+02 | 1.20E+02 | 6.00E+01 | 1.04E+01 | 5.59E+00 | 1.85E+00 | 9.19E+02 |
| *Titan* | 9.96E+04 | 4.99E+04 | 2.50E+04 | 4.16E+03 | 2.08E+03 | 2.72E+02 | 2.85E+02 |
| *Neptune* | 6.10E+01 | 5.04E+01 | 4.55E+01 | 4.16E+01 | 4.13E+01 | 4.10E+01 | 5.49E+02 |
| *Uranus* | 1.88E+02 | 1.80E+02 | 1.78E+05 | 1.77E+02 | 1.77E+02 | 1.77E+02 | 5.40E+02 |

***Table 2:*** *Difference in km between calculated position after 1 day using Verlet solver and NASA Horizons actual positions.*

| Planet | 1 day | 12 hours | 6 hours | 1 hours | 30 minutes | 4 minutes | 500 seconds (the one we use) |
|---|---|---|---|---|---|---|---|
| *Sun* | 3.75E+03 | 8.01E+03 | 1.39E+03 | 1.76E+02 | 1.63E+02 | 1.71E+02 | 1.69E+02 |
| *Mercury* | 8.37E+09 | 5.03E+10 | 6.79E+09 | 9.20E+07 | 8.67E+07 | 1.08E+07 | 2.31E+07 |
| *Venus* | 1.09E+09 | 1.80E+08 | 1.06E+08 | 1.73E+07 | 9.04E+06 | 2.28E+06 | 3.32E+06 |
| *Earth* | 1.41E+08 | 6.14E+07 | 8.91E+07 | 1.45E+06 | 2.61E+06 | 3.72E+05 | 7.57E+05 |
| *Moon* | 2.58E+07 | 9.96E+07 | 1.61E+09 | 1.79E+08 | 1.93E+08 | 4.80E+05 | 3.56E+05 |
| *Mars* | 3.13E+07 | 1.53E+07 | 7.57E+06 | 1.26E+06 | 6.42E+05 | 1.03E+05 | 1.93E+05 |
| *Jupiter* | 3.30E+05 | 1.63E+05 | 8.95E+04 | 6.43E+04 | 6.58E+04 | 6.77E+04 | 6.73E+04 |
| *Saturn* | 1.42E+05 | 2.30E+05 | 4.30E+05 | 1.19E+06 | 1.95E+06 | 1.23E+03 | 2.78E+05 |
| *Titan* | 6.44E+08 | 8.27E+08 | 1.72E+09 | 5.04E+09 | 8.24E+09 | 1.90E+06 | 1.18E+09 |
| *Neptune* | 2.94E+04 | 2.88E+04 | 2.88E+04 | 2.89E+04 | 2.89E+04 | 2.90E+04 | 2.90E+04 |
| *Uranus* | 2.66E+04 | 1.67E+04 | 1.20E+04 | 8.59E+03 | 8.30E+03 | 8.05E+03 | 8.09E+03 |

***Table 3:*** *Difference in km between calculated position after 1 year using Runge-Kutta solver and NASA Horizons actual positions.*

| Planet | 1 day | 12 hours | 6 hours | 1 hours | 30 minutes | 4 minutes | 500 seconds (the one we use) |
|---|---|---|---|---|---|---|---|
| *Sun* | 9.29E+05 | 6.83E+03 | 1.58E+02 | 1.67E+02 | 1.70E+02 | 1.73E+02 | 1.72E+02 |
| *Mercury* | 1.70E+06 | 1.95E+07 | 7.68E+05 | 1.31E+05 | 5.52E+04 | 1.52E+04 | 7.65E+03 |
| *Venus* | 7.97E+06 | 1.11E+07 | 2.61E+06 | 1.64E+06 | 1.56E+06 | 1.51E+06 | 1.52E+06 |
| *Earth* | 9.29E+05 | 1.25E+07 | 1.56E+05 | 4.14E+04 | 3.22E+04 | 2.46E+04 | 2.58E+04 |
| *Moon* | 8.58E+05 | 1.26E+07 | 1.74E+05 | 2.75E+04 | 2.35E+04 | 2.22E+04 | 2.22E+04 |
| *Mars* | 3.30E+06 | 8.22E+06 | 8.15E+05 | 1.50E+05 | 8.49E+04 | 2.90E+04 | 3.81E+04 |
| *Jupiter* | 3.35E+05 | 5.65E+06 | 1.08E+05 | 6.98E+04 | 6.86E+04 | 6.81E+04 | 6.81E+04 |
| *Saturn* | 8.30E+04 | 3.98E+06 | 2.15E+04 | 4.43E+03 | 2.77E+03 | 1.35E+03 | 1.58E+03 |
| *Titan* | 9.69E+05 | 6.19E+06 | 2.38E+06 | 2.04E+05 | 7.15E+04 | 8.89E+03 | 2.35E+03 |
| *Neptune* | 2.96E+04 | 2.89E+04 | 2.89E+04 | 2.89E+04 | 2.89E+04 | 2.90E+04 | 2.90E+04 |
| *Uranus* | 2.68E+04 | 1.69E+04 | 1.21E+04 | 8.63E+03 | 8.32E+03 | 8.06E+03 | 8.10E+03 |

***Table 4:*** *Difference in km between calculated position after 1 year using Verlet solver and NASA Horizons actual positions.*

**Appendix B:** link to github with the excel sheets with our results;

https://github.com/Willem-P1/WindExperimentsResults.git