Group 02 – Project1.2 – Final Report

Adrien Bersia, Ignacio Cadarso Quevedo, Bianca Caissotti di Chiusano, Yana Chernyatyeva, Artur Czaban

Abstract

This report describes the simulation of a space mission from Earth to Titan, one of Saturn's moons. The objective of the mission is to launch a spaceship from Earth, orbit around and land on Titan, then return back to Earth. The broader purpose of this mission is to use higher-order differential equation solvers, simulate wind and implement controllers, while considering the ideas and concepts of Software Engineering. As follows, this paper will tackle the implementation of the spaceship, including the 4th-Order Runge-Kutta and Verlet Solvers for its trajectory, Newton Raphson for the calculation of its initial velocity, its Fuel system and Orbit calculation around Titan. For the landing module, both open and closed loop controllers will be implemented, taking into consideration the gravity and wind of Titan. This report, moreover, discusses the conclusions of our findings. After implementations and experiments, the results show that with a fuel mass of 1149.16 tons, using the Velocity Verlet solver, Newton Raphson method, open and closed controllers, as well as a wind model for Titan, a realistic simulation of a mission from Earth to Titan, and back, can be realised.

Table of Contents

Introduction

Modern Space exploration has been brought out by humans only since the 20th century. Before then, space had been explored through the use of ancient astronomy. According to NASA, Space exploration can be divided into: astronomy, unmanned probes, and manned probes. Through the intricate use of astronomy and technology, this type of exploration has led to many benefits and achievements [1]. More importantly, it is interesting to look behind the reasons for exploring space. The understanding of our universe can have many benefits whether financial, political or genetic, nonetheless knowledge has to be acquired in order for resources to be located or dangers to be detected [2].

Recently, NASA has announced a new project, which goes by the name of "Dragonfly". It refers to a rotorcraft that will be launched in 2026 and should land on Titan in 2034 [3]. Titan is Saturn's largest moon and the second largest in our solar system [4]. It has a dense atmosphere made up, mostly, of nitrogen, and characterised by several liquid hydrocarbon surfaces. Titan was Saturn's first moon to be discovered, in the 1950's by Dutch astronomer Christriaan Huygens. "Dragonfly's" aim is to look for prebiotic chemical processes, common both on Earth and on Titan. This, however, is not the first exploration of Titan. It was first explored in 1979 by NASA's Pioneer 11, followed by Voyager 1 in 1980 and Voyager 2 in 1981.

This report aims to recreate similar missions, through outlining the methodology and implementation for simulating orbital positions, velocities, gravity, on both cosmic objects and a rocket. Runge-Kutta, Verlet and Euler solvers will be firstly described, and then compared, with the means of calculating trajectories. Moreover, this report will discuss the implementation of Newton Raphson method to calculate initial velocities , and the intricacies of orbital physics. Finally, it will conclude with an implementation of open and closed loop controllers for the landing module, in order to land on Titan.

Problem Statement

This report will investigate, analyse and test numerous algorithms with the aim of answering the following problem statement: "*Can we simulate a realistic mission to Titan, within a reasonable amount of time and mass fuel, using and testing different solvers and controllers?*"

Research Questions

To fully tackle the problem statement of this report, the following research questions will be investigated and answered:

1. How to mathematically model the solar system
2. How to approximate a solution to a differential equation
3. What is the effect on accuracy when decreasing the step size?
4. What is the minimum step size needed for a reasonable result?
5. What are the advantages of each particular solver?
6. What amount of fuel is needed to get to Titan and back?
7. What are the pros and cons of an open or closed loop controller?

Methods

To simplify the notation in the remaining of this report, we will use a vector $\overrightarrow{x_t}$ to denote the position of an object at a time t. For phase II this vector is 3 dimensional, and 2 dimensional for phase III. On the same principle, the velocity will be denoted as $\overrightarrow{v_t}$.
A component i of a vector $\overrightarrow{u_t}$ is denoted u_t^i and of a vector $\vec{u}$ u_i. By convention we assume that the components of $\overrightarrow{x_t}$ are $x(t)$, $y(t)$ and $z(t)$.

Space Travel

The goal of this part is to be able to compute accurate trajectories of the solar system and find optimum launch parameters to simulate a rocket traveling from Earth toward Titan, get into an orbit around Titan and come back.

To achieve this we need to simulate a realistic solar system. This is done by solving the differential equation of movements given by the law of gravitation and Newton's second law. This problem is tackled by implementing an ODE Solver. To be able to get a realistic result we implemented different kind of solvers, here under described.

Solvers

4th Order Runge-Kutta Solver

One of the solvers that we implemented is the classic 4th order Runge-Kutta method denoted by the formula $x(t_0 + \Delta t) = x(t_0) + \frac{k_1+2k_2+2k_3+k_4}{6}$
Where each k_i is defined as:

$$k_1 = f(x_0, t_0)$$

$$k_2 = f\left(x_0 + k_1 \times \frac{\Delta t}{2}\ ,\ t_0 + \frac{\Delta t}{2}\right)$$

$$k_3 = f\left(x_0 + k_2 \times \frac{\Delta t}{2}\ ,\ t_0 + \frac{\Delta t}{2}\right)$$

$$k_4 = f(x_0 + k_3 \times \Delta t\ ,\ t_0 + \Delta t)$$[5]

Verlet Solver

The Verlet method is a widely used integrator for finding a solution to the kinematic equation for the motion of an object:

$$\vec{x} = \overrightarrow{x_0} + \overrightarrow{v_0}t + \frac{1}{2}\vec{a}t^2 + \frac{1}{6}bt^3 + \cdots$$

With $\vec{x}$ - the position, $\vec{v}$ - the velocity, $\vec{a}$ - the acceleration, b - the jerk term, t - time.

The original idea by Verlet considers a Taylor Series Expansion of the position coordinate in two different directions of time:

$$\vec{x}(t + \Delta t) = \vec{x}(t) + \vec{v}(t)\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2 + \frac{1}{6}b(t)\Delta t^3 + O(\Delta t^4)$$

$$\vec{x}(t - \Delta t) = \vec{x}(t) - \vec{v}(t)\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2 - \frac{1}{6}b(t)\Delta t^3 + O(\Delta t^4)$$

Adding two equations together and solve for $x(t + \Delta t)$ we get the Verlet integration step for the position:

$$\vec{x}(t+\Delta t) = 2\vec{x}(t) - \vec{x}(t-\Delta t) + \vec{a}(t)\Delta t^2 + O(\Delta t^4) \quad [6]$$

The Verlet integrator gives greater stability than the Euler method and more importantly it preserves properties of the system. It has to be noted, that the classic Verlet integration requires an initialization of step $\vec{x}(t-\Delta t)$ made by another algorithm.

The Velocity Verlet algorithm is more commonly used and it is mathematically equivalent to the classic Verlet integration but it incorporates velocity explicitly:

$$\vec{v}\left(t+\frac{1}{2}\Delta t\right) = \vec{v}(t) + \frac{1}{2}\vec{a}(t)\,\Delta t$$

$$\vec{x}(t+\Delta t) = \vec{x}(t) + \vec{v}\left(t+\frac{1}{2}\Delta t\right)\Delta t$$

$$\vec{a}(t+\Delta t) = f(\vec{x}(t+\Delta t))$$

$$\vec{v}(t+\Delta t) = \vec{v}\left(t+\tfrac{1}{2}\Delta t\right) + \tfrac{1}{2}\vec{a}(t+\Delta t)\,\Delta t \quad [7]$$

4-Step Adams-Bashforth Method

A common method to increase the accuracy of a solver is to take into account several steps. This can be done using the Adams-Bashforth method.

Firstly we bootstrap this method with our previously mentioned 4th Order Runge-Kutta method.

Then we can proceed to calculate:

$$W_{i+1} = W_i + \frac{\Delta t}{24}(55f(t_i, w_i) - 59f(t_{i-1}, w_{i-1}) + 37f(t_{i-2}, w_{i-2}) - 9f(t_{i-3}, w_{i-3}))$$

This method is further used in our Predictor-Corrector method

Predictor-Corrector Method (4th order three-step Adams-Moulton implicit method)

We can also use a previously calculated value and increase its accuracy. This process is described as the predictor-corrector method.

Firstly we calculate the predictor of W_{i+1} denoted as $\widetilde{W_{i+1}}$ using the previously mentioned Adams-Bashforth method. Then we correct our predictor with the equation

$$W_{i+1} = W_i + \tfrac{\Delta t}{24}(9f(t_{i+1}, \widetilde{W_{i+1}}) + 19f(t_i, w_i) - 5f(t_{i-1}, w_{i-1}) + f(t_{i-2}, w_{i-2})) \quad [8]$$

Newton-Raphson Method

Once the trajectories of the planets calculated remains the problem of finding a realistic trajectory for a probe-like object. This can be simplified as finding an initial position and velocity $\overrightarrow{x_0}$ and $\overrightarrow{v_0}$ such that at the final time f the probe is at the position $\overrightarrow{x_f}$.

To bring this problem to a 3 unknowns system we need to decide on the initial position arbitrarily. We decided to take $\overrightarrow{x_0}$ as the closest point to the target on our starting object (the earth at the beginning of the travel for example). If $\overrightarrow{x_{earth,0}}$ is the position of the earth during the launch and $\overrightarrow{x_{titan,f}}$ the position of the titan at the time f then the initial position of our probe in Earth's repository is given by

$$\overrightarrow{x_0} = R_{earth} \times \frac{\overrightarrow{x_{titan,f}} - \overrightarrow{x_{earth,0}}}{\left\|\overrightarrow{x_{titan,f}} - \overrightarrow{x_{earth,0}}\right\|}$$

With R_{earth} the radius of the earth.

We can then define a function $\vec{g}$ such that, given an initial velocity $\overrightarrow{v_0}$, $\vec{g}(\overrightarrow{v_0})$ gives us the distance to the target at the time f, as the vector separating the two objects. Our goal is to find $\overrightarrow{v_0}$ such that $\vec{g}(\overrightarrow{v_0}) = \vec{0}$. This can be done iteratively using the Newton-Raphson method. As an iterative solution, we need to define ε the accuracy of our result, and so we want to find $\overrightarrow{v_0}$ such that $\|\vec{g}(\overrightarrow{v_0})\| < \varepsilon$.

We need a first approximation of $\overrightarrow{v_0}$ to initialize the process. A fair assumption is to take $\|\overrightarrow{v_0}\|$ equal to the escape velocity of the starting planet, and in the direction of the target (colinear with the starting position in earth's repository). This velocity is defined as:

$$v = \sqrt{\frac{2GM}{R+h}} \text{ [9]}$$

With G the universal gravitational constant (6,674 30(15) × 10^{-11} m^3 kg^{-1} s^{-2}), M the mass of the planet, R its radius, and h the initial altitude (in case of an orbit for example).

We can then improve this initial velocity with the Newton-Raphson formula:

$$\overrightarrow{v_{0,k+1}} = \overrightarrow{v_{0,k}} - J^{-1}.g\left(\overrightarrow{v_{0,k}}\right) \text{ [10]}$$

Where k denotes the number of iterations, and J the Jacobian matrix of g:

$$J = \begin{pmatrix} \frac{\partial g^x}{\partial v_k^x} & \frac{\partial g^x}{\partial v_k^y} & \frac{\partial g^x}{\partial v_k^z} \\ \frac{\partial g^y}{\partial v_k^x} & \frac{\partial g^y}{\partial v_k^y} & \frac{\partial g^y}{\partial v_k^z} \\ \frac{\partial g^z}{\partial v_k^x} & \frac{\partial g^z}{\partial v_k^y} & \frac{\partial g^z}{\partial v_k^z} \end{pmatrix}$$

Every partial derivative of g can be approximated by the 3 point centered difference method:

$$\frac{\partial g^i}{\partial v_k^j} = \frac{g^i\left(v_k^j + h\right) - g^i\left(v_k^j - h\right)}{2 \times h}$$

Fuel mass calculation

Once the starting velocity $\overrightarrow{v_0}$ known, we can use it to estimate the amount of fuel needed in case of the use of a rocket instead of a simple probe.

We know the initial velocity of our rocket, for example equal to the velocity of the earth at $t = 0$, and the velocity $\overrightarrow{v_0}$ we want to reach at $t \approx 0$. Ideally we would like to get from $\overrightarrow{v_{0,Earth}}$ to $\overrightarrow{v_0}$ in a very small amount of time, that we can take as our resolution, the step size δt. The acceleration needed is then given by:

$$\vec{a} = \frac{\overrightarrow{v_0} - \overrightarrow{v_{Earth}}}{\delta t}$$

And Newton's second law gives us:

$$\sum \vec{F} = m\vec{a} \Leftrightarrow \overrightarrow{F_T} + \overrightarrow{F_G} = m\vec{a} \Leftrightarrow \overrightarrow{F_T} = m\vec{a} - \overrightarrow{F_G} = m\frac{\overrightarrow{v_0} - \overrightarrow{v_{Earth}}}{\delta t} - \overrightarrow{F_G}$$

With $\overrightarrow{F_T}$ the force induced by the thrusters, $\overrightarrow{F_G}$ the pull of gravity, and m the mass of the rocket at this step (the mass of fuel burned at that step is ignored).

The force $\overrightarrow{F_T}$ is induced by the thruster, consuming fuel. The rate of this fuel consumption is given by:

$$\dot{m} = \frac{\overrightarrow{F_T}}{V_E}$$

With V_E the exhaust velocity of the thrusters. This rate however is limited by the physical capabilities of the rocket, and so we have:

$$\dot{m}_{max} = \frac{\overrightarrow{F_{max}}}{V_E}$$

We can then deduce:

$$t_{combustion} = \frac{\dot{m}}{\dot{m}_{max}} \times \delta t$$

And so find the amount of fuel needed:

$$m = t_{combustion} \times \dot{m}$$

This method however only works if we know the mass of the rocket, this means we can't find the amount of fuel needed to go to titan without knowing the mass of the rocket at titan, charged with the fuel needed to be put in orbit and to come back. This means that assuming an empty fuel tank when reaching earth, we can calculate back the total amount of fuel needed by summing the mass needed from the last usage to the first.

Orbit

In order to get to Titan we need our rocket to enter and stay in orbit around Saturn's satellite. We need for that to compute the orbital velocity, the velocity sufficient to cause a natural or artificial satellite to remain in orbit [11].

The vector of this velocity is tangent to the path and has constant magnitude [12].

Therefore, the probe must have a velocity perpendicular to the radius vector of Titan. The radius vector is defined as

$$\vec{r} = \overrightarrow{x_{probe}} - \overrightarrow{x_{titan}}$$

We can pick arbitrary an orthogonal basis vectors to $\vec{r}$, for example:

$$\overrightarrow{e_1} = (-r_y, r_x, 0)$$

$$\overrightarrow{e_1} = (0, -r_z, r_y)$$

Finally, the velocity vector is any combination of those:

$$v_x = e_{1,x} + e_{2,x}$$

$$\mathrm{v_y} = e_{1,y} + e_{2,y}$$

$$v_z = e_{1,z} + e_{2,z}$$

$$\vec{v} = (v_x, v_y, v_z)$$

And then we can multiply it by the module of the orbital velocity.

$$v = \sqrt{\frac{G * M_{Titan}}{R_{Titan}}}$$

[13]

Landing on Titan

GUI

For the landing, we have opted to build a simple 2 dimensional GUI with coordinate in kilometers, which demonstrates the landing point (0,0) as a red rectangle, the trajectory as yellow rectangles, the current position of the lander as a black rectangle and a red line representing the Y axis. The GUI also displays the Velocity of the lander and wind as well as an arrow that points to the direction that the wind blows in. Pictures of the GUI can be seen as appendices 2 and 3.

Physics Engine

Our physics engine uses the Velocity Verlet method as integrator to derive the angle of rotation, horizontal and vertical positions of the lander from the differential equations:

$$\ddot{x} = \vec{u}(\sin\theta)$$

$$\ddot{y} = \vec{u}(\cos\theta) - \vec{g}$$

$$\ddot{\theta} = {}^{\tau}/_{I}$$

Where $\vec{x}$ denotes the horizontal position, $\vec{y}$ the vertical position, $\vec{u}$ the acceleration provided by the main thruster, θ the angle of rotation, τ the torque provided by the side thrusters, I the moment of inertia.

First it calculates the free fall trajectory of the lander (assuming the lander falls in vacuum), then it applies the wind influence and accelerations provided by thrusters and recalculates the trajectory.

Wind model

For our wind model we generate random velocity vertices based on the altitude, in order to do so we divided the altitude into ranges which have different max and min speeds of wind (Wind speed is the norm of a velocity vector) [14].

These ranges are:

- Altitude > 60 000 m generates vertices with speed between 432 km/h and 160km/h
- Altitude between 60 000 m and 45 000 m generates vertices with speed between 150 km/h and 120 km/h
- Altitude between 44 999 m and 30 000 m generates vertices with speed between 150 km/h and 120 km/h
- Altitude between 29 999 m and 15 000 m generates vertices with speed between 119 km/h and 70 km/h

- Altitude between 14 999 m and 10 000 m generates vertices with speed between 29 km/h and 13 km/h
- Altitude between 9 999 m and 4 000 m generates vertices with speed between 12 km/h and 4 km/h
- Altitude between 3 999 m and 0 m generates vertices with speed between 3 km/h and 0 km/h

We also fallow some assumptions based European space agency such as:

- The wind goes west to east and reverses direction twice, at the 6 km mark and the 700 m mark
- The closer the lander is to the ground the slower the wind is
- The wind speed goes consistently down after the 60 km mark (it is turbulent above it)
- The wind blows from West to East (reverses between 6000 and 700)

We then proceed to use the generated vertices to calculate the force of the wind on an object using the equation

$$F = A \times P \times D$$

Where:

Since our lander is a sphere we calculate A (Area of the object) as the half of a sphere

$$A = \frac{2\pi r^3}{3}$$

P is the Air pressure, we use V (Wind speed) and 0.613 to calculate it, since 0.613 is a number unique to earth we correct for it by multiplying it by titan's air pressure

$$P = 1.47 \times 0.613 \times V^2$$

D stands for the Drag Coefficient and since A is half of a sphere we chose 0.47 as our drag coefficient

$$D = 0.47$$

Open loop controller

The first controller that will be described is the Open-loop (non-feedback) controller. This type of controller performs computations on the provided input, while only using the system's initial or current state. Unlike the feedback controller (Section.) is does receive any feedback regarding whether the inputs given have reached the desired goal. On one hand, this characteristic greatly impacts the inefficiency of the open-loop, however, it makes the controller simple and less computationally expensive. [15]

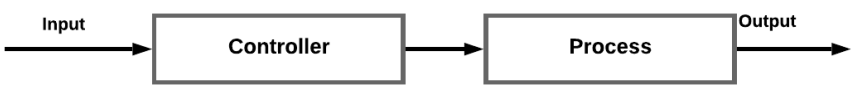


Figure 01

As it can be observed from the implementation (Section.), the open-loop controller takes in two inputs: The Lander's initial state and its distance from the landing pad's x-coordinate. The methodology used for this implementation was to split the landing into separate phases, below described as "Rotations" and "Vertical Landing". Finally, the controller outputs a set of states of the lander at every time t, from which is possible to derive data, including all positions and velocities. The sub methods previously stated are described in the following paragraphs.

To rotate the lander, we need to activate the side thrusters to provide the necessary torque. The way we do this is by using the maximum thrust for a certain amount of time and then activating them again with opposite torque to stop the lander from rotating passed the desired angle. The amount of time necessary to rotate is calculated by:

$$t = \sqrt{\frac{\Delta\theta}{\dot{V}(t)}}$$

Which is derived from the kinematic equation:

$$\theta = \omega_0 t + \frac{1}{2}\alpha t^2$$

[16]

To prevent the lander from falling we make a 45º rotation. Then we set our main accelerations to produce an acceleration:

$$a = \frac{1.352}{cos(\theta)}$$

That way we can counter the acceleration in the y axis and still move in the x axis towards the final position.

Once the x position is reached, we make a 90º rotation and then apply the thrusters until $v_x = 0$.

Then we make a last rotation to make the angle 0º.

Now that the lander has stabilized on the desired x-coordinate, we can start the vertical landing toward y = 0. The first quarter of the vertical landing will be a free fall. This free fall aims to assure that we land within a reasonable time and will decrease the usage of the thrusters. In order to get the point until which the lander can free fall, we subtract the current y position of the lander, divided by 4 from the current y position itself. Hence, until the lander reaches that point we keep all thrusters deactivated.

In the second part of the vertical landing, the main thruster is activated in order to decelerate and safely land on the surface of titan. The acceleration *a*, needed to decrease the velocity of the lander, is found using the *Bisection* method, also known as *Binary Search* method. This iterative method is commonly used to approximately find a root f(x) = 0, of a function f, on an interval $[a, b]$

Overall, the Bisection method can be described in the following way [17]:

Calculate p_n the midpoint of the interval $[a_n, b_n]$

$$p_n = \frac{a_n + b_n}{2}$$

If p_n makes the lander not to reach the ground $a_{n+1} = p_n$

If p_n makes the lander to reach the ground too fast $b_{n+1} = p_n$

To find the acceleration needed for landing, the two endpoints of the initial interval provided to the bisection method will be:

$$a = g + 0.7\,, \qquad b = g$$

Making the interval $[g, g + 0.7]$

The midpoint of the interval will be set as the acceleration. This acceleration can then be tested by checking at the velocity of the lander as it reaches y = 0. Once this velocity satisfies the landing conditions, the Bisection method stops and the required acceleration for a safe landing is found.

Feedback controller

The feedback controller on the opposite of the open loop controller works as a reaction to its environment, sensed by measurements. The goal is to use the present data to decide on a behavior. The measurable data available is the position and velocity of our lander as well as its inclination and angular velocity.

We want to get to the target point [0;0]. Our process will be to get closer and closer to the x=0 axis, and go down at a velocity depending on the altitude. This process works in several steps:

First we want to compensate the gravity and the current velocity to slow down our descent. We need for that to define our goal velocity, the vertical velocity we want to have depending on the altitude. The key elements in choosing this velocity is to ensure that it gets close to 0 when y tends to 0. The initial maximum velocity can be define as a velocity reached after a certain time of free fall or as an arbitrary value. We then need to choose the shape of our velocity function. Some testing showed us that with a linear function, a maximum velocity v_{max} of 100ms^{-1} was big enough to have a reasonable descent time and still be able to get to 0 at the surface, starting from an orbit between 100 and 300 km. So we can define the goal velocity as

$$v_{goal}(y) = -\frac{v_{max}}{y_0} \times y$$

Then, knowing our current velocity and the wished velocity we need to find the value of the thrust to be applied. v_{goal} is the vertical component of our velocity, and so is used to calculate the vertical component of our main thruster, $\vec{u}$. For that we use Newton's second law stipulating that a force on a mass provides us an acceleration :

$$u_y = \frac{v_{goal}(y) - v_y}{\delta t} \times m$$

With δt our time step, m the mass of the lander, v_y the vertical component of the current velocity.

The x component of $\vec{u}$ can then be found knowing the current inclination θ of our lander:

$$u_x = u_y \tan\theta$$

$$\text{As } u_y = \vec{u}\cos\theta \text{ and } u_x = \vec{u}\sin\theta \text{ and so } u_x = u_y\frac{\sin\theta}{\cos\theta}$$

We finally need to keep in mind that we are simulating a real life situation, and so our thruster is limited by its physical properties, so we need to scale down the norm of our vector if it is bigger than F_{max} the maximum force of our thruster.

We are now able to reach the surface at a controlled speed. We also want to reach the axis $x = 0$ before touching the surface to reach our target. This is done by applying a torque τ to our lander to induce a rotation around its z axis (z conventionally defined as $x \times y$, $\times$ denoting here the cross product). The torque is defined by

$$\tau = r \times F_{side}$$

With F_{side} the maximum force deployed by the side thrusters of the lander and r the distance to the center of mass of our lander. Our lander was assumed spherical, and of homogeneous mass, hence a center of mass centered on the lander. r is then the radius of our lander. We designed our lander as similar as the moon lander Eagle used during the Apollo 11 mission [18]. and so F_{side} is either null or at full capacity. This means we can apply a torque of $r \times F_{side}$, $-r \times F_{side}$ or 0. Our goal is to oscillate around the $x = 0$ with a damping factor to reduce our amplitude. The following formulas are an open end and weren't implemented for time reasons:

The equation of a damped oscillation is given by:

$$m\frac{d^2x}{dt^2} + b\frac{dx}{dt} + kx = 0$$

With solutions of the form

$$x(t) = A_0 e^{-\frac{b}{2m}t} sin(\omega t + \phi)$$

This means we want to keep the $x(t)$ values within $\pm A_0 e^{-\frac{b}{2m}t}$

Moreover, the damp can be more or less efficient depending on the value of the b factor:

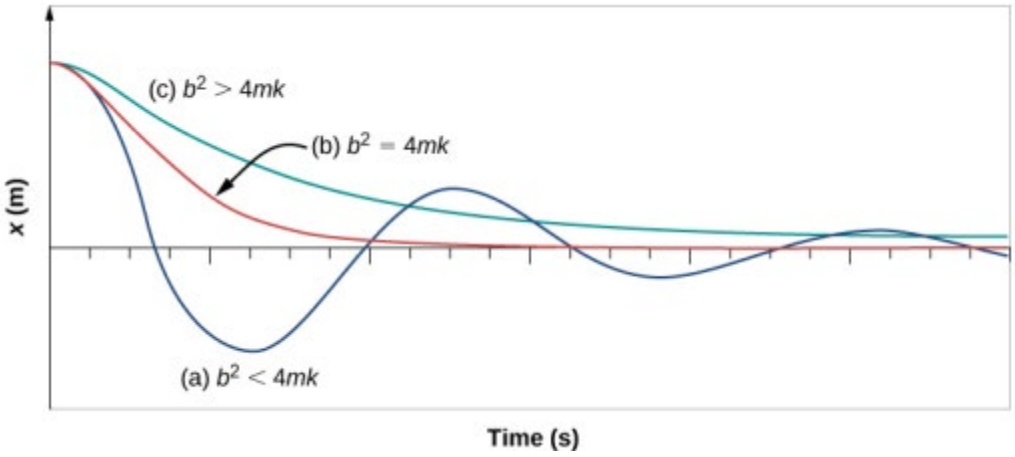


Fig 02 : The position versus time for three systems in a damped oscillation motion. [19]

With k the stiffness of the system. This value can't be calculated, this means we need to experiment several values of b to find an optimum scenario.

We now need to find when to apply a torque and of what value to get an appropriate motion. This can be done by solving the system

$$\begin{cases} x(t) = A_0 e^{-\frac{b}{2m}t} sin(\theta) \\ \ddot{x}(t) = u \sin\theta \\ \ddot{\theta} = \frac{\tau}{I} \end{cases}$$

With I the moment of inertia of the lander. As a sphere we have $I = \frac{2}{5}mr^2$

Implementation

Space Travel

Solvers

4th Order Runge-Kutta Solver

Inputs: States an array containing the states (at first it only contains the first state), Time an array containing the time coresponding to each state, Delta the step Size
Output: array of states (trajectory)
Start:

for i = 1 until Final time

K1 = f(time[i],states[i])

yOfK1 = states[i] + (K1 $\times \frac{\text{Delta}}{2}$)

K2 = f(time[i] + $\frac{\text{Delta}}{2}$, yOfK1)

yOfK2 = states[i] + (K2 $\times \frac{\text{Delta}}{2}$)

K3 = f(time[i] + $\frac{\text{Delta}}{2}$, yOfK2)

yOfK3 = states[i] + (K3 $\times$ Delta)

K4 = f(time[i] + Delta , yOfK3)

States[i+1] = states[i] + (Delta $\times$ $\frac{\text{K1} \times (2 \times \text{K2}) \times (2 \times \text{K3}) \times \text{K4}}{6}$)

End

Classic Verlet

Inputs: function of the differential equation, initial state, array of time steps
Output: array of states (trajectory)
Start:

State [0] = Initial state

State [1] = State Euler [1]

for i=2 ⟶ timeSteps.length

$position_i = 2position_{i-1} - position_{i-2} + a_{i-1} * step^2$

$v_i = v_{i-1} + a_{i-1} * step$

State [i] = new State { $postion_i, v_i$ }

End

Velocity Verlet

Inputs: function of the differential equation, initial state, acceleration, array of time steps
Output: array of states (trajectory)
Start:

State [0] = Initial state
for i=1 ⟶ timeSteps.length
$v_{int} = v_{i-1} + \frac{1}{2}a_{i-1} * step$
$position_i = position_{i-1} + v_{int} * step$
$a_i = function(position_i)$
$v_i = v_{int} + \frac{1}{2}a_i * step$
State [i] = new State { $position_i, v_i$ }
end

4-Step Adams-Bashforth Method Implementation

Inputs: States an array containing the states (at first it only contains the first state and the next three are bootstrapped by 4th order Runge-Kutta), Time an array containing the time corresponding to each state, Delta the step Size
Output: array of states (trajectory)
Start:
for i = 4 until Final time
nextState = States[i]
f1 = f(Time[i], States[i])
$\text{nextState} = \text{nextState} + (\frac{\text{Delta}}{24}) \times 55 \times \text{f1}$
f2 = f(Time[i-1], States[i-1])
$\text{nextState} = \text{nextState} - (\frac{\text{Delta}}{24}) \times 59 \times \text{f2}$
f3 = f(Time[i-2], States[i-2])
$\text{nextState} = \text{nextState} + (\frac{\text{Delta}}{24}) \times 37 \times \text{f3}$
f4 = f(Time[i-3], States[i-3])
$\text{States[i+1]} = \text{nextState} - (\frac{\text{Delta}}{24}) \times 9 \times \text{f4}$
End

Predictor-Corrector Method Implementation

Inputs: States an array containing the states (at first it only contains the first state and the next three are bootstrapped by 4th order Runge-Kutta), Time an array containing the time corresponding to each state, Delta the step Size
Output: array of states (trajectory)
Start:
for i = 4 until Final time
Predictor = calculated with the above mentioned 4-Step Adams-Bashforth Method
corrector = States[i]
f1 = f(Time[i+1], Predictor)
$\text{corrector} = \text{corrector} + (\frac{\text{Delta}}{24}) \times 9 \times \text{f1}$
f2 = f(Time[i], States[i])
$\text{corrector} = \text{corrector} + (\frac{\text{Delta}}{24}) \times 19 \times \text{f2}$
f3 = f(Time[i-1], States[i-1])
$\text{corrector} = \text{corrector} - (\frac{\text{Delta}}{24}) \times 5 \times \text{f3}$
f4 = f(Time[i-2], States[i-2])

States[i+1] = corrector + $(\frac{\text{Delta}}{24}) \times 1 \times \text{f4}$

End

Newton-Raphson Method

Function: Calculate optimum initial velocity
Input: starting planet start, target the trajectory of the target, timeToReach the approximated time to get there, altitude the initial altitude (0 if on the surface)
Output: the optimal trajectory of the probe
Parameters: Precision, accuracy of the result, VEL_STEP the step of improvements
Start:

```
%find the optimum point on the starting planet to go to the endpoint
    direction = last position of target – start position
%normalize
    direction = direction/norm of direction
%go on the surface on the planet in its repository
    startPosition = direction*radius of start
    Move the starting position to the earth repository
%calculate escape velocity
    v0 = direction * sqrt(2*G*mass of start/(radius of start+altitude))
%try out
    do
        Calculate trajectory
        g0 = closest distance between trajectory of the probe and target
        if(norm of g0 < Precision)
            break
        hx = (VEL_STEP, 0, 0)
        traject = calculate trajectory with (startPosition, v0+hx)
        g1x = closest distance between trajectory of the probe and target

        hy = (0, VEL_STEP, 0)
        traject = calculate trajectory with (startPosition, v0+hy)
        g1y = closest distance between trajectory of the probe and target

        hz = (0, 0, VEL_STEP)
        traject = calculate trajectory with (startPosition, v0+hz)
        g1z = closest distance between trajectory of the probe and target

        traject = calculate trajectory with (startPosition, v0-hx)
        g1mx = closest distance between trajectory of the probe and target

        traject = calculate trajectory with (startPosition, v0-hy)
        g1my = closest distance between trajectory of the probe and target

        traject = calculate trajectory with (startPosition, v0-hz)
        g1mz = closest distance between trajectory of the probe and target

        jacobian = calculate the derivatives to form the Jacobian matrix
```

inverse = inverse of jacobian
v0 = v0-inverse*g0
while(norm of g0 != 0);
End

Fuel mass calculation

Function: Calculate fuel mass
Input: launch velocity v(x,y,z), initial velocity v0(x,y,z), step the time step, fuelLeft the amount of fuel left after the process
Output: m the mass of fuel needed to get to v0
Parameters: maxThrust the physical limits of the thruster, vE the exhaust velocity, M the mass of the rocket
Start:
forceNeeded = (v-v0)*(M+fuelLeft)/step
mRateNeeded = norm of forceNeeded/vE
maxRate = maxThrust/vE
timeOfCombustionNeeded = mRateNeeded/maxRate*step
fuelMass = maxRate*timeOfCombustionNeeded
End

Orbit

Inputs: positions of the probe and titan
Output: the orbit velocity
Start:
Vector r = position(probe)-position(Titan)
Vector e1 = (-r.getYComponent, r. getXComponent,0)
Vector e2 = crossProduct(r, e1)
e1 =e1 / norm(e1)
e2 =e2 / norm(e2)
vx = (e1.getXComponent + e2.getXComponent)
vy = (e1.getYComponent + e2.getYComponent)
Vector v = (vx, vy, 0)
vNorm = sqrt((G * titanMass)/(titanRadius))
v = vNorm*v + Titan.velocity
End

Landing on Titan

Lander Specifications

To simplify the problem we assumed the shape of the lander as a sphere, and used the same characteristics as the moon lander of the Apollo 11 mission for the main and side thrusters as for the radius of the module. [18]

Physics Engine

Inputs: initial lander state (position, velocity, acceleration, angular velocity, angle, torque)
Output: array of lander's states (trajectory)
Start:
State [0] = Initial lander state

while (altitude > 0)

[new theta, new angular velocity, new torque] = integrator(current theta, current angular velocity, current torque, next torque)

[new position, new velocity, new acceleration] = integrator(current position, current velocity, current acceleration, next acceleration)

State.add(new theta, new angular velocity, new torque, new position, new velocity, new acceleration)

End

Generating velocity vectors of the wind

Inputs: MinSpeed the minimum speed of the wind, MaxSpeed the maximum speed of the wind
Output: Wind vector
Start:

Speed = random number between MinSpeed and MaxSpeed

SpeedSquared = Speed to the Second Power

GeneratedX = random number between SpeedSquared/2 and SpeedSquared (in order to skew in favor of the X axis since the wind mainly moves from west to east)

x = Square Root of GeneratedX

y = Square Root of (SpeedSquared - GeneratedX)

WindVector = velocity vector with coordinates (X, Y)

End

Open Loop Controller

Vertical Landing (after free fall phase)

The Bisection method takes in the Lander's current state, and will return the required acceleration for a safe landing on Titan.

Function: openLoop
Inputs: finalX, initialState
Output: Trajectory
Start:

trajectory = **rotate**(initialSate,PI/4)

acceleration = G*cos(angle)

while(trajectory.positionX<finalX)

trajectory.addAcceleration(acceleration)

trajectory = **rotate**(trajectory.lastState,-PI/2)

acceleration = -(lastState.velocityY^2/2*lastState.position)

while(trajectory.velocityX>0)

trajectory.addAcceleration(acceleration)

rotate(lastState, PI/4)

freeFall

acceleration = **Bisection**(lastState)

calculateTrajectory(acceleration)

End

Function: rotate

Inputs: state, angle

Output: trajectory

Start:

```
acceleration = maxTorque/mass;
halftime = sqrt(angle/acceleration)*14
for i<=halftime
        set Torque
for i>=halftime
        set -Torque
calculateTrajectory
```

End

Function: bisection

Inputs: state

Output: root

Start:

```
a = g + 0.7
b = g
While true
        c = (a+b)/2
        if(Check(state, c) == 0)
                a = c
        else if(Check(state, c) == 2)
                b = c
        else return c
end while
```

End

The Bisection method calls the Check() method to check whether the acceleration passed will make the lander diverge (returns 0), crash (returns 0) or converge satisfying the landing conditions (returns 1). In order to check this result we first need to generate the states of the lander using the calculateNextState() method from the physics engine.

Function: Check
Inputs: state, acceleration
Output: whether it diverges or converges
Start:

```
While state.getPos().getY() > 0
        CalculateNextState() using current state and acceleration
        If state.getPos().getY() > 100000 return 0
        If abs(state.getVelocity().getY()) > 0.02 return 2
        return 1
```

```
        end while
End
```

Feedback Controller

Function: Calculate next thrusts
Input: initial position x0 y0, position x y, velocity vx vy, angle theta, angle velocity omega, step the time step
Output: u a vector of the main thruster depending on theta, v a torque
Parameters: maxVel the maximum vertical velocity, maxThrust the physical limits of the thruster
Start:
%Calculate torque:

```
        if(x<0)
                if(theta<=0 and omega<=0)
                        v = maximum torque
                else if(theta> absolute value of (x/x0)* PI/4 and omega>0)
                        v = - maximum torque
        else
                if(theta>=0 and omega>=0)
                        v = - maximum torque
                else if(theta<-absolute value of (x/x0)* PI/4 and omega<0)
                        v = maximum torque
%Calculate goal velocity such that v=0 at y=0
        goalVelocity = - (maxVel/y0)*y
% Calculate u to get there:
        uy=(goalVelocity - vy)/step
        ux=uy*tan(theta)

        if(norm of u > maxThrust)
                u = u*maxThrust /norm of u
End
```

Experiments

Solvers

Once our solvers were implemented, we conducted some experiments to compare the results and make some conclusions regarding the precision of different solvers. For that we used the following steps:

1) Extract the trajectories of celestial bodies with different step sizes (1 day, 12 hours, 1 hour, 1 minute) from Nasa Horizons website;
2) Calculate the trajectories with relative step sizes by our solvers;
3) Build the graphs and calculate errors using the Root Sum Squared method;
4) Compare the results and conclude.

Fuel mass step size

By the formula of our fuel mass calculation we notice a high dependency on the time step. We want to know how accurate our result is by estimating it's variation with the variation of the step size. This is done by plotting the mass found for the space trip for different time step and find an approximated interpolating function of our error.

Orbit

We can plot the positions of the Probe and Titan for a period of time to see how stable is the orbit.

Correctness of the feedback controller's assumptions

(Experience still to be conducted) The value of v_{max} needs to be tested to confirm a reasonable enough value allowing a realistic landing. This means that the descent must be slow enough to allow the stabilization of the $x = 0$ axis. This needs to be done by testing a landing without wind, with $y_0 = 100000$ as it is the smallest initial altitude defined in the requirements and thus the worst case scenario in case of a high maximum velocity.

(Experience still to be conducted) We also need to find an appropriate damping coefficient b for our x velocity. This needs to be done by measuring and plotting the absolute value of x over the time with different values of b, the goal is to find the coefficient b such that the mean value of $|x|$ is minimized, this means that we have an important damping effect, and so that we reach the $x = 0$ fast.

(Experience still to be conducted) Once the optimum parameters found we can test the influence of the wind on our controller by measuring the success rate of our landing over several iterations and measure the offset induced by the wind. These results can be used to compare the feedback controller with the open loop controller.

Results

Solvers

First of all, our experiments showed that all our solvers work correctly as the graphs of trajectories are in line with the real data.

Secondly, we observed that the 4-stage Runge-Kutta and the Verlet solvers work much better than the Euler method (cf Appendix).

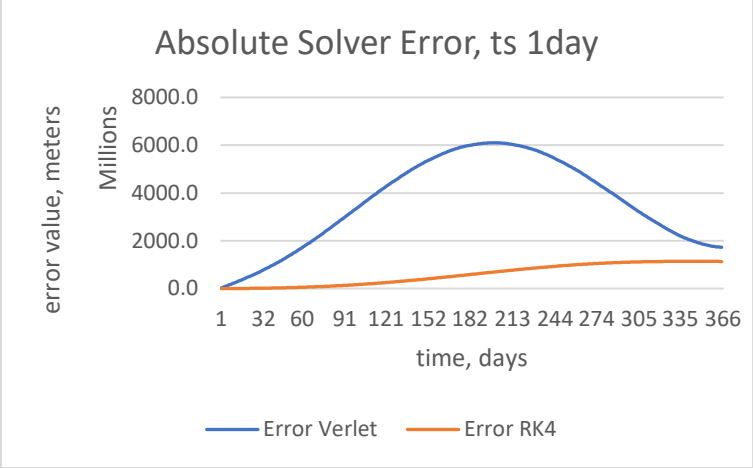


Figure 03

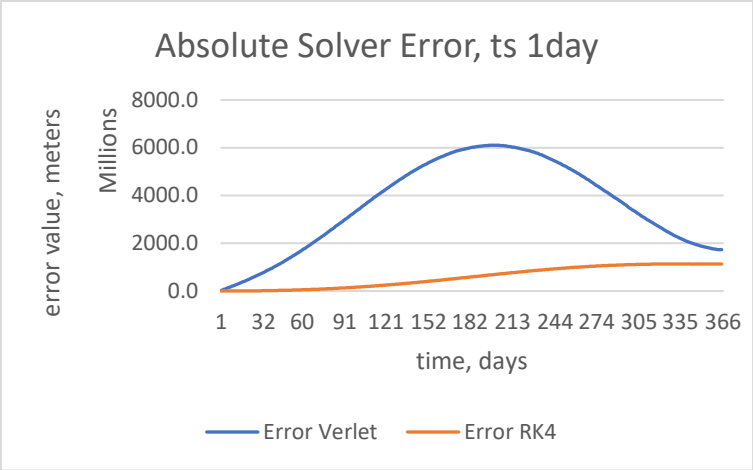


Figure 04

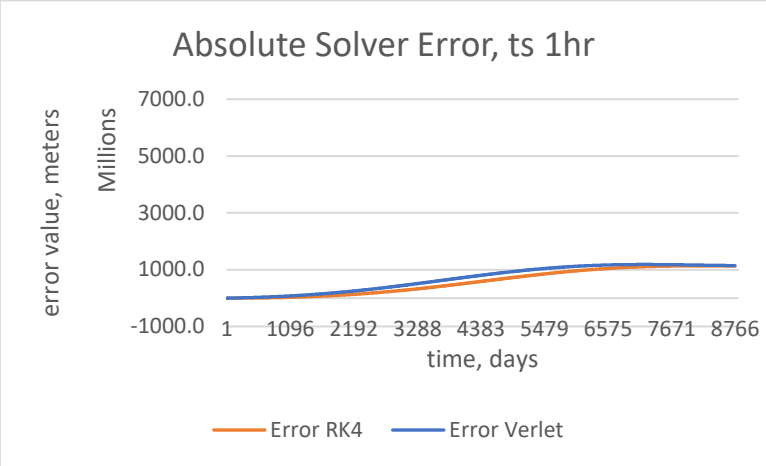


Figure 05

We also discovered that the smaller the step size was the better the precision, however the sensitivity is different. The Verlet solver has higher sensitivity of the precision to the step size than the 4-stage Runge-Kutta method. Furthermore, we detected that the Velocity Verlet solver works better than the Classic Verlet one, and with small step sizes such as 1 hour and 1 minute gives greater precision than Runge-Kutta method.

#ts	RK4 error	Velocity Verlet error	error diff
ts=1d, tf=366			
1	0.00	0.00	0.00
2	13,296.44	128,766.90	115,470.47
366	487,721,578.28	574,203,036.04	86,481,457.76
ts=12hrs, tf=731			
1	0.00	0.00	0.00
2 (1 day)	3,319.60	16,216.87	12,897.27
3	13,274.27	33,733.59	20,459.32
4 (2 days)	29,861.11	54,349.80	24,488.69
730	1,131,354,203.53	1,153,027,321.57	21,673,118.04
731 (365 days)	1,131,234,624.54	1,152,976,346.86	21,741,722.32
ts=1hr, tf=8761			
24 (1 day)	12,189.99	12,170.62	-19.37
48 (2 days)	50,887.61	50,846.64	-40.96
8761 (365 days)	1,131,220,530.13	1,131,372,717.11	152,186.98
ts=1min, tf = 2881			
1440 (1 day)	13,254.40	13,254.39	-0.0053
2881 (2 days)	53,075.56	53,075.55	-0.0120

Figure 06

Where error diff = (Velocity Verlet error) – (RK4 error), i.e. if error diff < 0, Velocity Verlet performs better and vice versa

In addition to this, the Verlet solver is a symplectic integrator which is an integrator whose solution resides on a symplectic manifold. This characteristic is important for physics, and thus, for the given problem because Hamiltonian's equations have solutions residing on a symplectic manifold. Symplectic integrators also tend to capture long-time patterns better than non-symplectic integrators.

Additional Solvers

Firstly, we can see that the new solvers are accurate when compared to the actual NASA data and RK4, With RK4 only having an advantage visible in the graph with the z coordinate, the rest requires the calculation of errors to clearly distinguish which one is more accurate

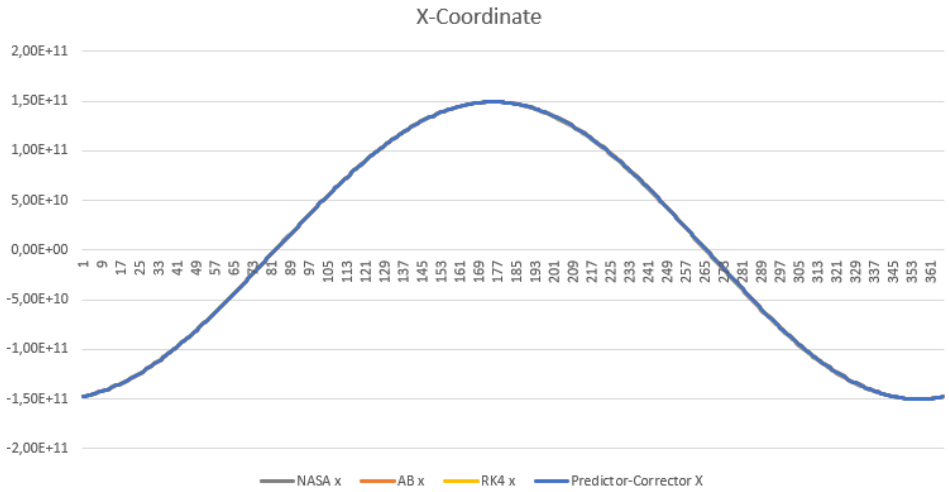


Figure 07

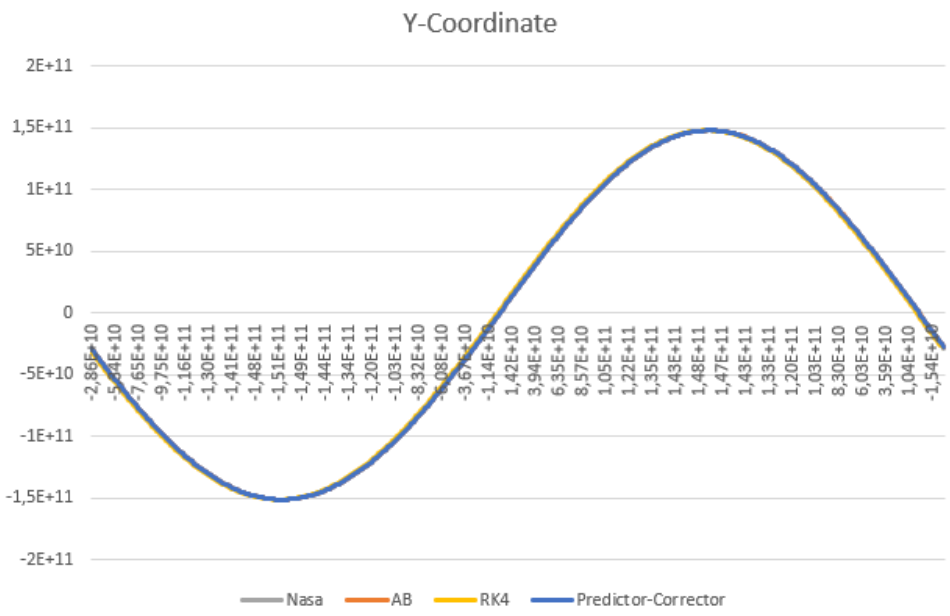


Figure 08

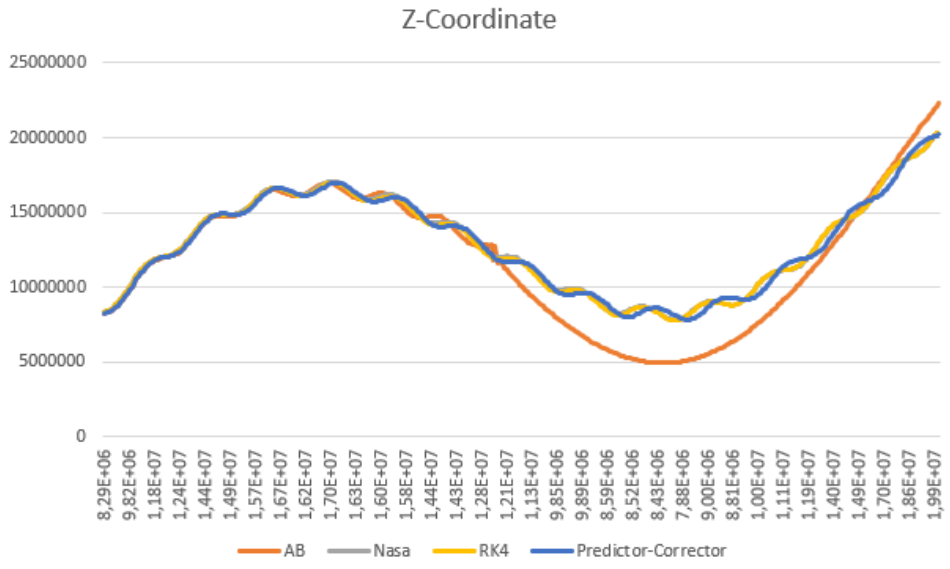


Figure 09

After calculating the relative errors of the Predictor-Corrector Method and Adams-Bashforth, although not visible on a graph (Except for the z coordinate), by looking at the numbers we can clearly see that the Predictor-Corrector Method is the more accurate one

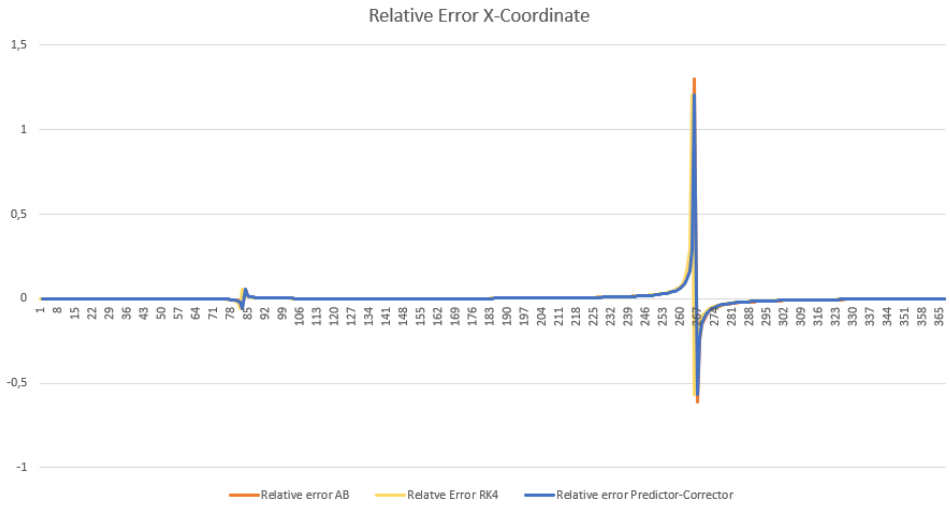


Figure 10

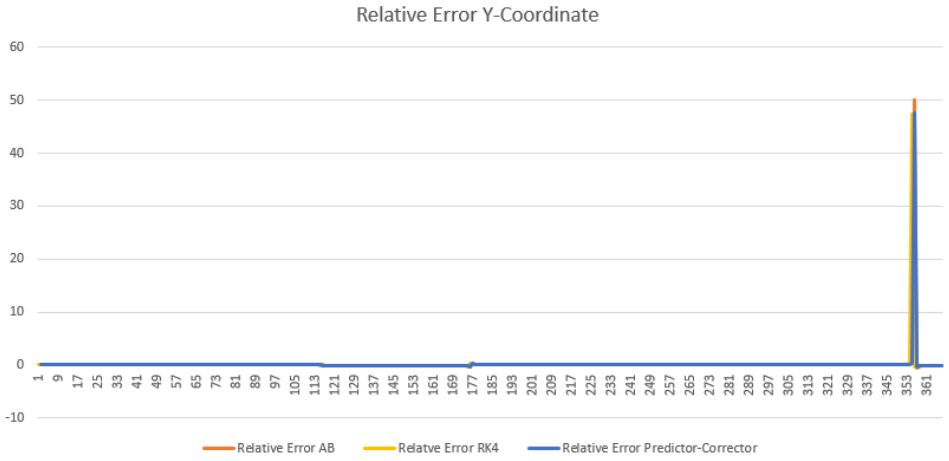


Figure 11

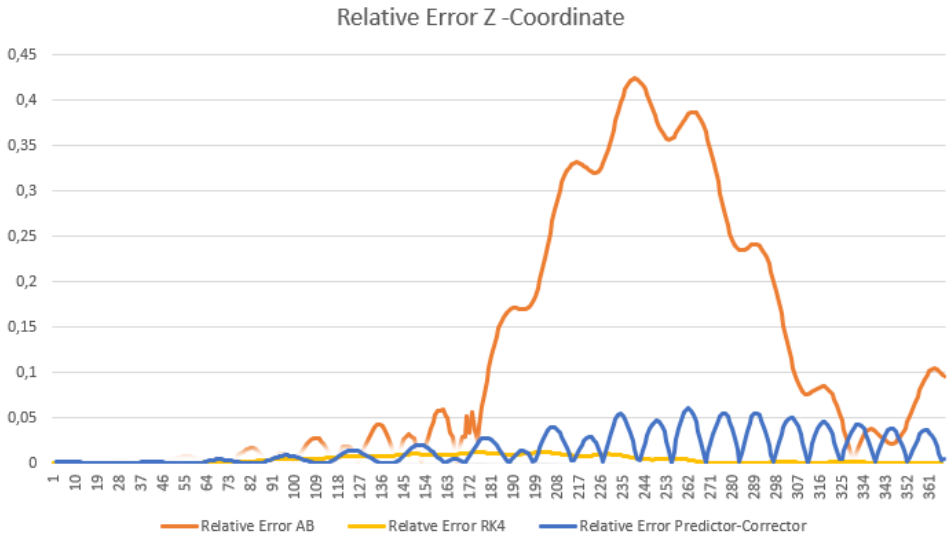


Figure 12

By looking at the comparison of the relative errors we can see that Adams-Bashforth is the worst out of these methods, while the Predictor-Corrector Method provides a good middle ground between AB and RK4. I'm assuming that the Predictor-Corrector Method might be a good alternative to RK4, although it is worse on the z coordinate, it is easier to implement.

Fuel mass step size

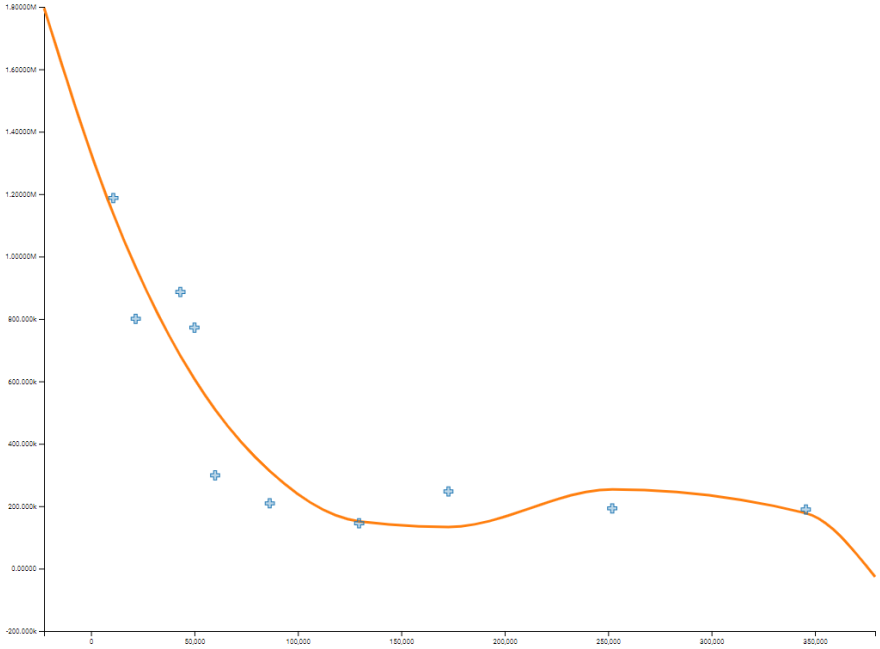


Figure 13

Mass fuel needed depending on the time step, plotted with a cubic regression

We can approximate the amount of fuel needed by the function $m(\delta t) = -0.0000h^3 + 0.0001h^2 - 18.5933h + 1326241.9786$

We notice that the important term of this approximation is, as suspected from the formula, the h factor, so we have an error of $O(h)$.

Orbit

We found that with an orbit of 200km above Titan´s surface we have an orbital velocity of 1800m/s. Here are the positions of the probe and Titan, with a time step of 3 minutes:

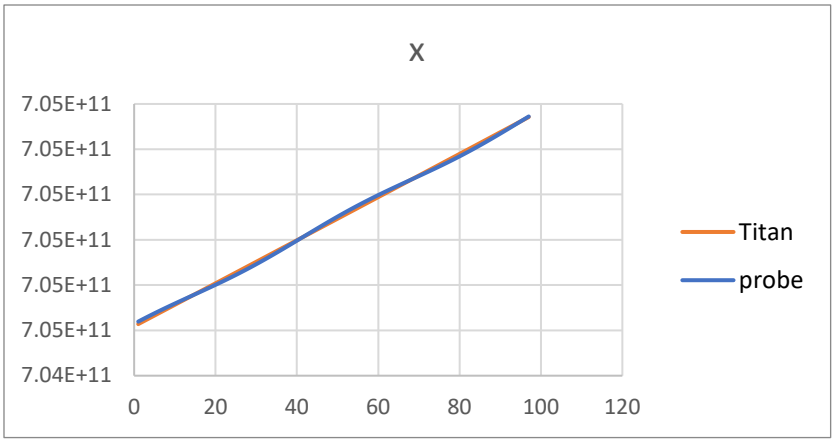


Figure 14

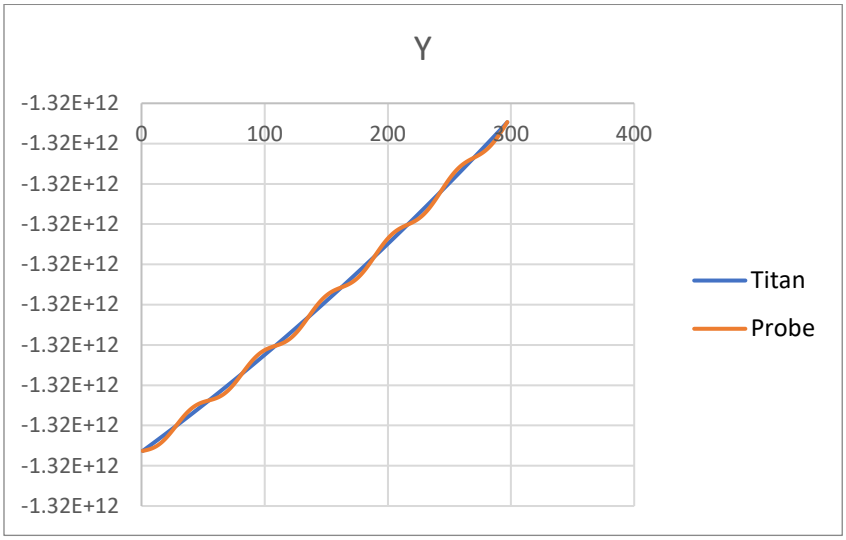


Figure 15

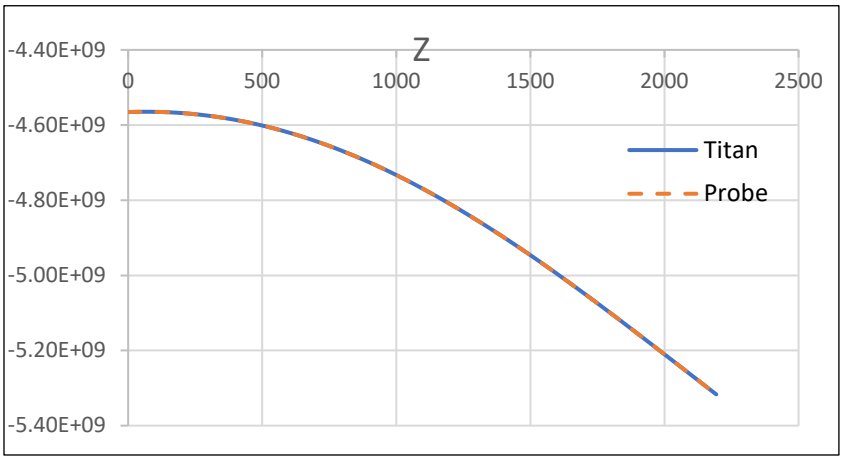


Figure 16

Correctness of the feedback controller's assumptions

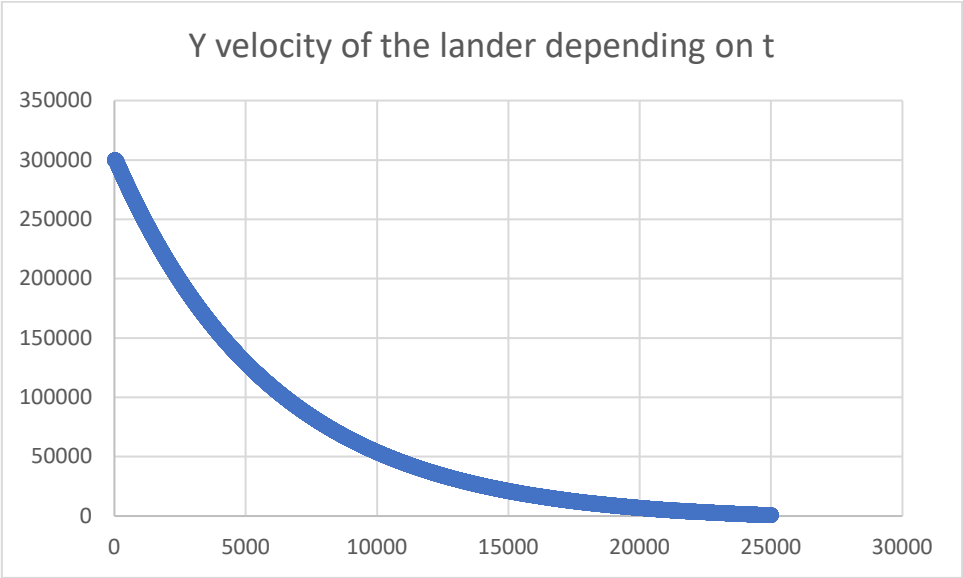


Figure 17

We notice that with a $v_{max} = 100$ our resulting velocities are dropping down following a parabolic shape. This is a good thing as it means that the lander doesn't suffer abrupt decelerations that might fragilize its integrity.

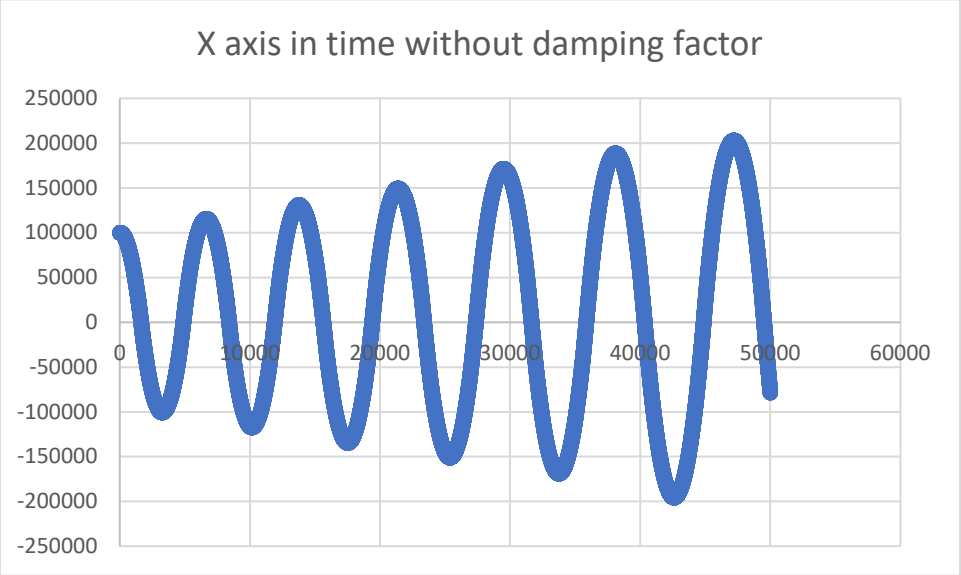


Figure 18

We notice that without a damping factor on our oscillation our horizontal position is defined by a growing oscillation, this is something to avoid as our goal is to get a stable position at x=0.

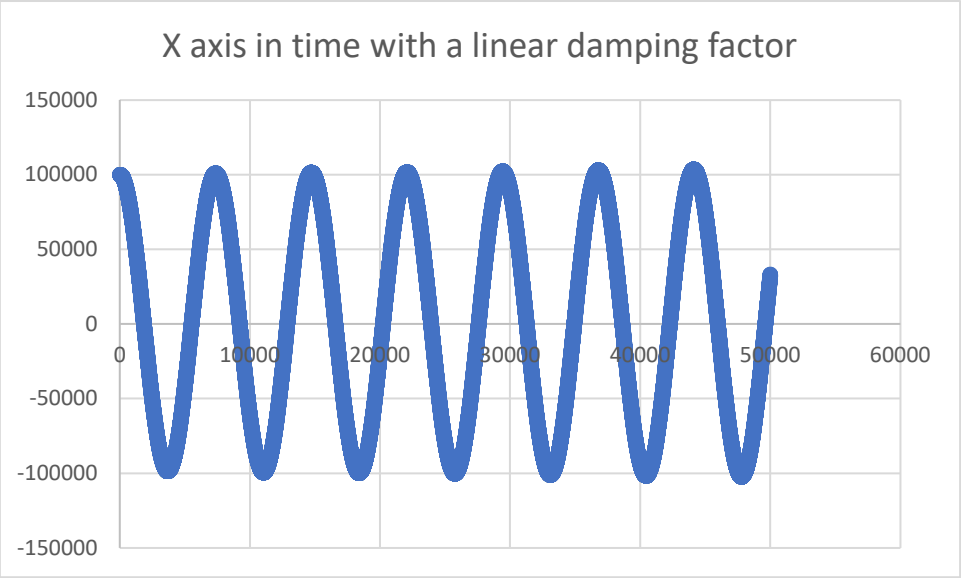


Figure 19

If we add a damping factor linearly dependant on the x position we notice a stabilization of the oscillation. This is however still not enough to get a stable position.

Discussion

Orbit

The only way we can reach a stable orbit is if we keep the step size small. Otherwise, the errors start to accumulate and it goes off.

Solvers

A Mathematical model of the solar system can be built using a set of differential equations which include celestial bodies' positions, velocities and accelerations. Accelerations can be calculated using the gravitational forces. The approximation of a solution of a differential equation can be done using numerical methods.

For this project we used various methods and compared their efficiency. The smaller the step size we use for the solvers, the better precision they give. To get a rough trajectory, *1 day* time step is enough. To get closer to the real numbers it's better to use step size equal or smaller than 1 minute but it will require much more computational power.

Euler method is the simplest integrator to use and implement. The classic Verlet and the Velocity Verlet methods, being symplectic integrators, preserve the properties of the system which is very important for physics. The Velocity Verlet doesn't need pre-previous positions for calculation.

After analysing the relative errors, we find that the predictor-corrector method, although less accurate might be a viable alternative to RK4 due to its easier implementation.

We believe that it would be beneficial to calculate and compare how computationally expensive the predictor-corrector method is to RK4 to see if it really is worth to sacrifice the slightly better accuracy of RK4 for the easier implementation of predictor-corrector method.

Wind

The number 0.613 used in the calculation of air pressure is unique to earth and we could not find how it was calculated, so we multiplied it by 1.47 to correct for titan's atmospheric pressure. We believe that if given more time, we could try other multipliers than 1.47 to test how the multiplication affects this equation.

Fuel mass calculation

The low accuracy of our method makes difficult a good approximation of the amount of fuel needed. However our results tend to turn around the amount of fuel actually used during the Saturn V mission to Titan. This real life scenario however takes into account more parameters than we do, such as the air resistance for example, so we can argue with the correctness of our method.

Correctness of the feedback controller's assumptions

A higher v_{max} is more interesting, as the landing will be faster and so more energy efficient, however the success of the landing is more important. There is moreover a maximum boundary on the reached velocity during a free fall (as the descent is not damped this boundary is the final velocity). There is no point in choosing a higher velocity. A reasonable maximum boundary could be the velocity at the half of the trajectory in free fall for example.

The function of v_{goal} as it is implemented is linear. A parabolic function could also be considered to shorten the descent time, or on the opposite a square root function to have a smaller velocity and so have more time to increase the x axis accuracy.

The stabilization of the x axis wasn't achieved within the time provided.

Controllers general comparison

In theory an open loop controller should be just as efficient as a feedback controller assuming that all parameters are known and taken into account. This condition in a real life problem is unreachable because of the amount of parameters and their often stochastic characteristics. The only suitable solution is then to use a feedback controller, able to correct the errors encountered without the need of understanding their origin.

In a complex and real life situation however this can mean a long processing time, going against the concept of a prompt reaction to a measured error. A good solution would then be to preprocess a solution using an open loop controller and then bring small corrections to it using a feedback controller after the start of the process.

Conclusion

In this section we will answer our main research question and do an overall discussion of how we could improve our investigation. We will conclude by doing a summary.

This report served as an overall analysis of the work done throughout Project 1-2. It explained, outlined and implemented several algorithms with the aim of investigating and providing an answer to our ***Problem Statement (Section.)***. This exploration was guided by 7 research questions, that were used to conduct experiments and create tables and graphs displaying the corresponding results. Restating the methods, in order to complete a mission from Earth to Titan, the trajectories were calculated using the Velocity Verlet solver, the initial velocities and orbit using the Newton-Raphson method, and finally the landing was brought out using a physics engine, controllers and by simulating wind.

Several conclusions can be drawn from the experiments conducted within this report. Regarding the trajectories from Earth to Titan and back, all implemented solvers work correctly. Nonetheless the Velocity Verlet solver showed a greater precision with small step sizes, compared to the other solvers, and was selected for the calculation of the trajectories. Secondly, through research, implementation and experiments we were able to calculate the fuel mass needed for the mission, about 1149.16 tons, and successfully orbit around Titan at 200km above its surface. Lastly, landing on Titan winded up being, to a certain extent, successful. Using the open loop controller the lander reaches the surface Titan at a reasonable velocity for a safe landing, however does not land on the correct x-coordinate of the landing pad. On the other hand, when using the feedback controller, the lander continuously oscillates in the horizontal position but manages to land on the surface without enduring any abrupt decelerations. Overall, the feedback controller provides a more accurate representation of landing, also due to the fact that it considers the impact of the wind on the lander.

In conclusion, to answer this report's problem statement, we are able to simulate a realistic mission to Titan within a reasonable time of 489 days using a working fuel system, solver and controller. For further research, we can look at ways to improve our controllers to land on titan and find more accurate ways of calculating the amount of fuel needed.

References

[1] Web.archive.org. 2021. *Space and its Exploration: How Space is Explored*. [online] Available at: <https://web.archive.org/web/20090702153058/http://adc.gsfc.nasa.gov/adc/education/space_ex/exploration.html> [Accessed 14 June 2021].

[2] Web.archive.org. 2021. *Space and its Exploration: Why Do We Explore Space?*. [online] Available at: <https://web.archive.org/web/20090702153943/http://adc.gsfc.nasa.gov/adc/education/space_ex/essay1.html> [Accessed 14 June 2021].

[3] NASA. 2021. *NASA's Dragonfly Mission to Titan Will Look for Origins, Signs of Life*. [online] Available at: <https://www.nasa.gov/press-release/nasas-dragonfly-will-fly-around-titan-looking-for-origins-signs-of-life> [Accessed 14 June 2021].

[4] NASA Solar System Exploration. 2021. *Titan - Overview*. [online] Available at: <https://solarsystem.nasa.gov/moons/saturn-moons/titan/overview/> [Accessed 14 June 2021].

[5] "4th-Order Runge Kutta Method for ODEs", *Youtube.com*, 2015. [Online]. Available: https://www.youtube.com/watch?v=1YZnic1Ug9g&t=410s. [Accessed: 14- Jun- 2021]

[6] [C. Holm, "Simulation Methods in Physics," 2013.]

[7] "Verlet Integration · Arcane Algorithm Archive", *Algorithm-archive.org*, 2021. [Online]. Available: https://www.algorithm-archive.org/contents/verlet_integration/verlet_integration.html. [Accessed: 22- Jun- 2021].

[8] Www3.nd.edu. [Online]. Available: https://www3.nd.edu/~zxu2/acms40390F15/Lec-5.6.pdf. [Accessed: 16- Jun- 2021].

[9] R. R. Bate, D. D. Mueller, and J. E. White, "Fundamentals of Astrodynamics," in *Fundamentals of Astrodynamics*, 1971, p. 35.

[10] A. Ben-Israel, “A modified newton-raphson method for the solution of systems of equations,” *Isr. J. Math.*, vol. 3, no. 2, pp. 94–98, 1965, doi: 10.1007/BF02760034.

[11] Encyclopedia Britannica. 1998. Orbital velocity | physics. [online] Available at:<https://www.britannica.com/science/orbital-velocity> [Accessed 13 June 2021

[12] Moebs, W., Ling, S. and Sanny, J., 2016. *4.4 Uniform Circular Motion*. [online] Opentextbc.ca. Available at: <https://opentextbc.ca/universityphysicsv1openstax/chapter/4-4-uniform-circular-motion/> [Accessed 13 June 2021].

[13] Physicsclassroom.com. 2021. *Mathematics of Satellite Motion*. [online] Available at: <https://www.physicsclassroom.com/class/circles/Lesson-4/Mathematics-of-Satellite-Motion> [Accessed 15 June 2021]

[“14] The way the wind blows on Titan", www.esa.int, 2007. [Online]. Available: https://www.esa.int/Science_Exploration/Space_Science/Cassini-Huygens/The_way_the_wind_blows_on_Titan. [Accessed: 10- Jun- 2021]

[15] Hashmi, S., Batalha, G., Van Tyne, C. and Yilbas, B., 2014. *Comprehensive Materials Processing*.

[16] Openstax.org. 2021. *10.2 Kinematics of Rotational Motion - College Physics | OpenStax*. [online] Available at: <https://openstax.org/books/college-physics/pages/10-2-kinematics-of-rotational-motion> [Accessed 22 June 2021].

[17] Faires, J. and Burden, R., 2003. *Numerical methods*. Pacific Grove, CA: Thomson/Brooks/Cole.

[18] “Apollo Lunar Module,” *Wikipedia*, 18-Jun-2021. [Online]. Available: https://en.wikipedia.org/wiki/Apollo_Lunar_Module. [Accessed: 22-Jun-2021]

[19] Damped Oscillations. (2020, November 5). Retrieved June 22, 2021, from https://phys.libretexts.org/@go/page/4066

Appendices

Annex 1: Trajectory of Earth over the period of 1 year, time step = 1 day, with Euler, RK4 and Classic Verlet

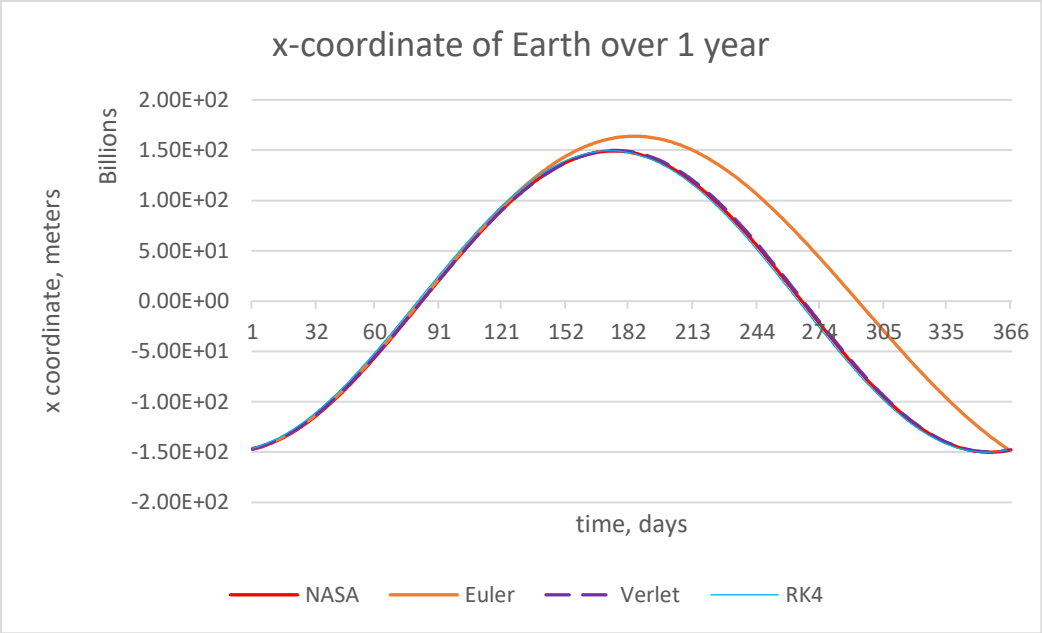


Figure 20

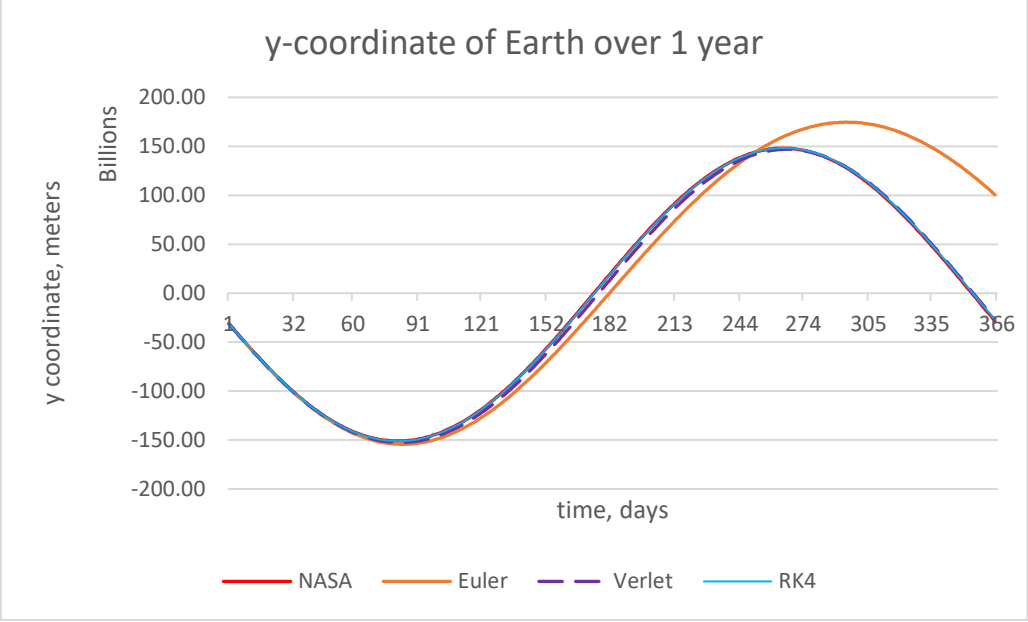


Figure 21

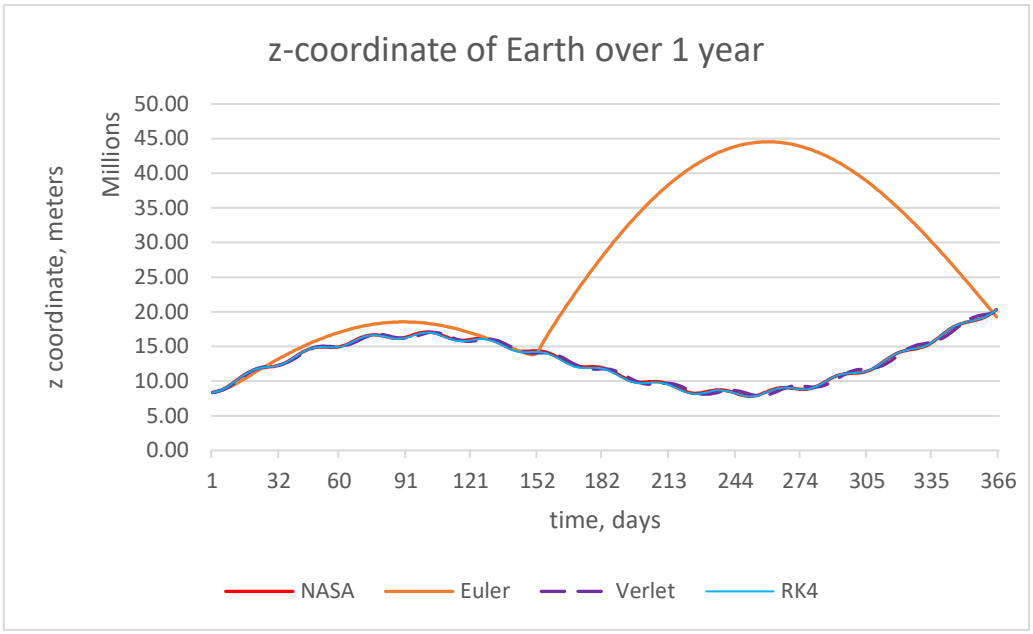


Figure 22

Annex 2: Trajectory of Earth over the period of 1 year, time step = 1 day, RK4 and Velocity Verlet

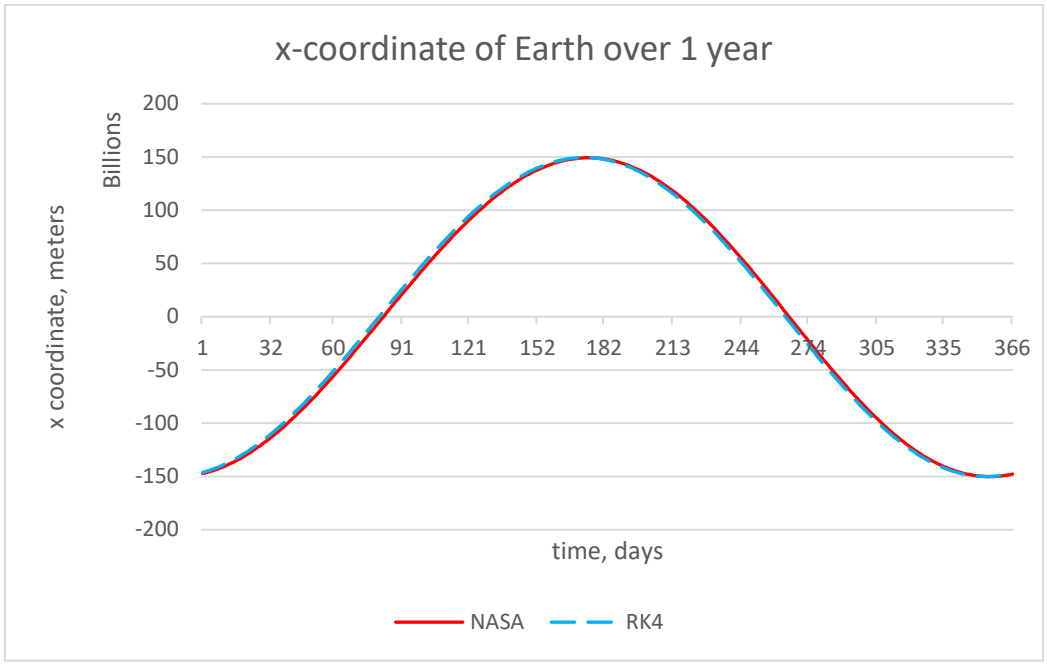


Figure 23

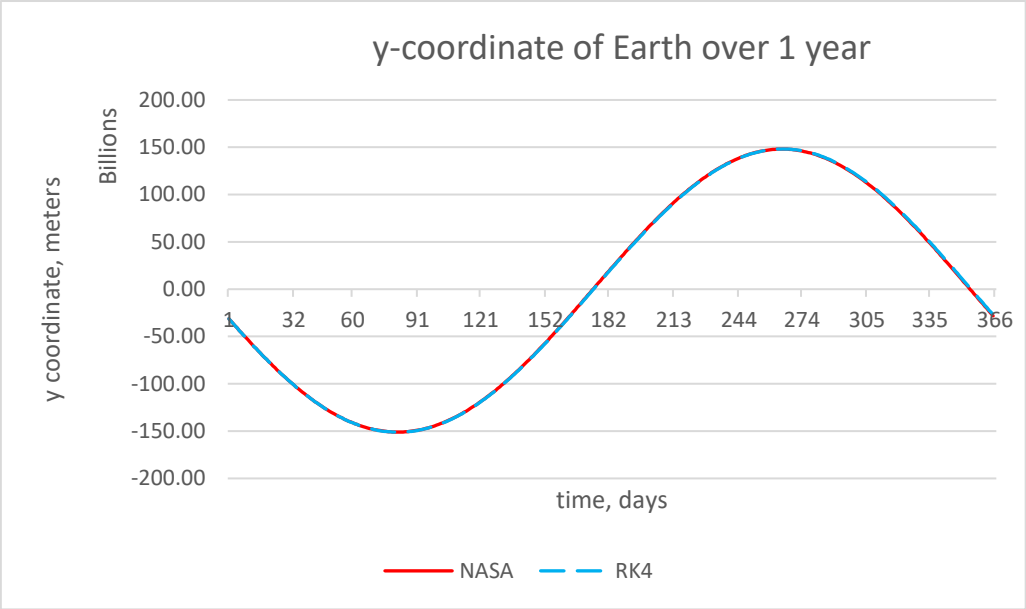


Figure 24

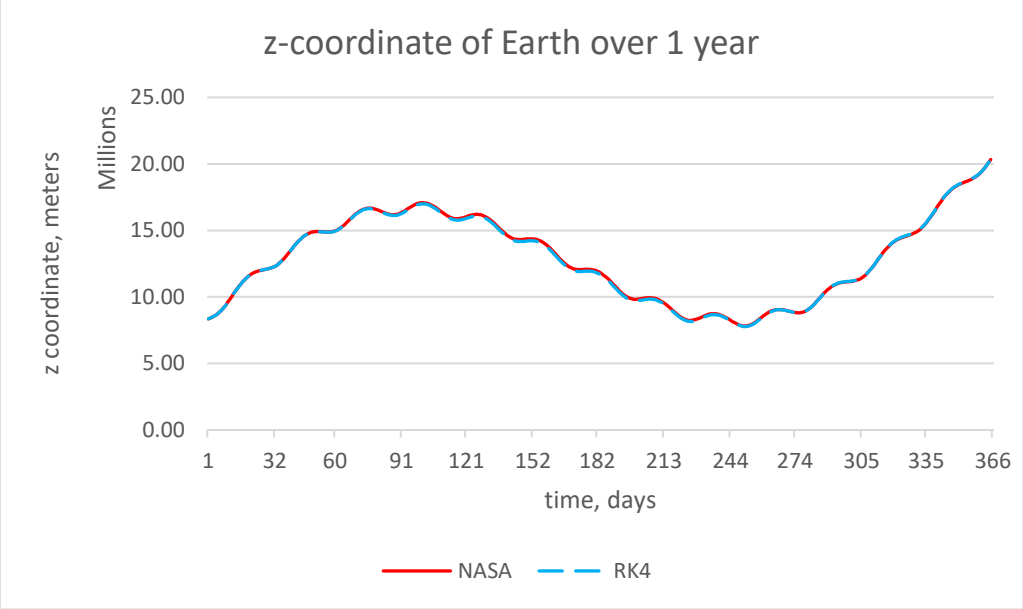


Figure 25

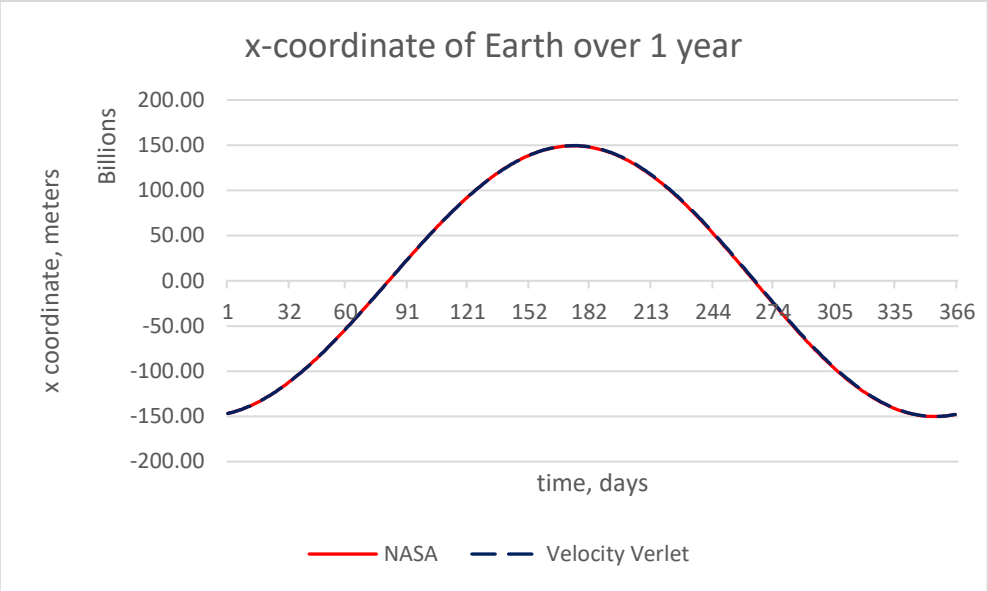


Figure 26

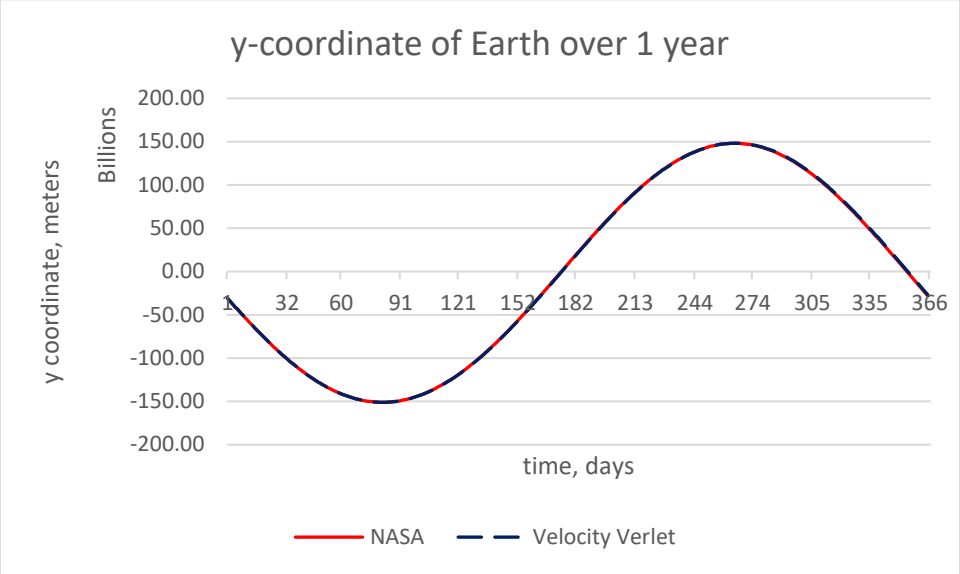


Figure 27

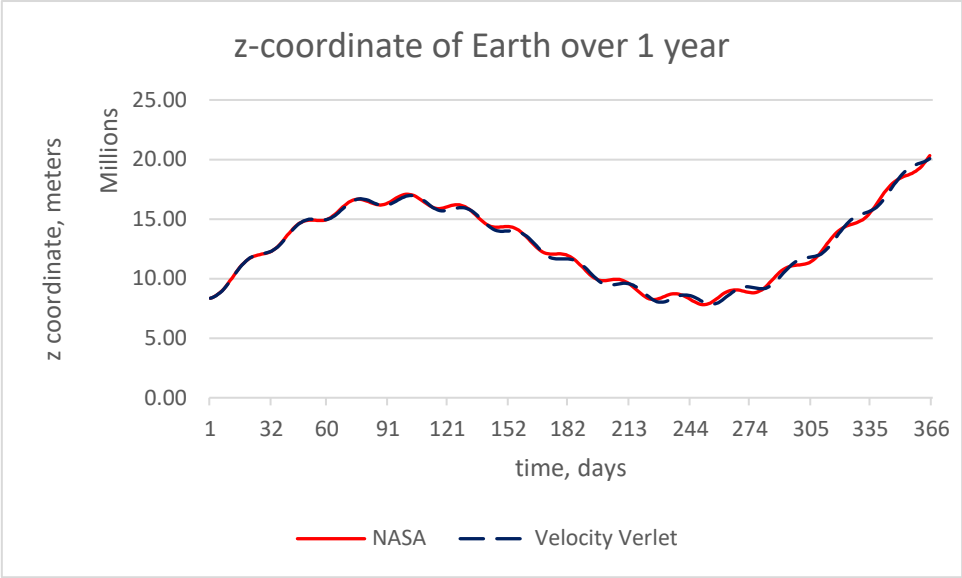
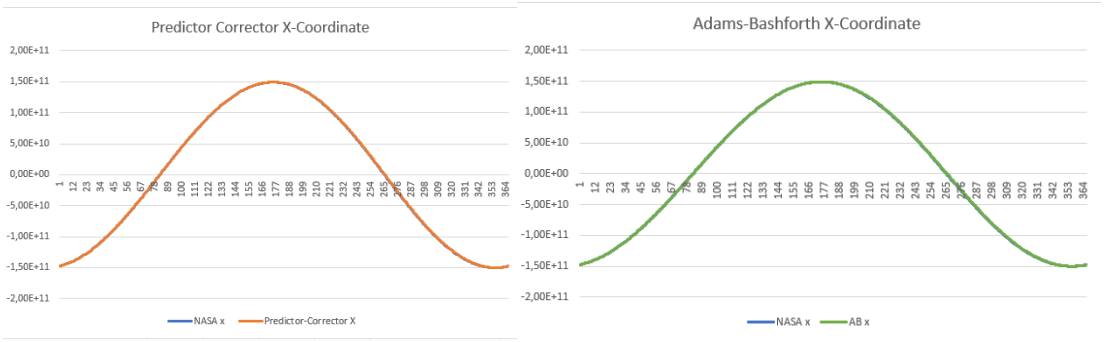


Figure 28

Annex 3: Solvers comparison: RK4 vs Velocity Verlet

Figure 29

Annex 4: Trajectory of the Earth over the period of 1 year, time step = 1 day, Predictor Corrector and Adams-Bashforth



Figures 30 and 31

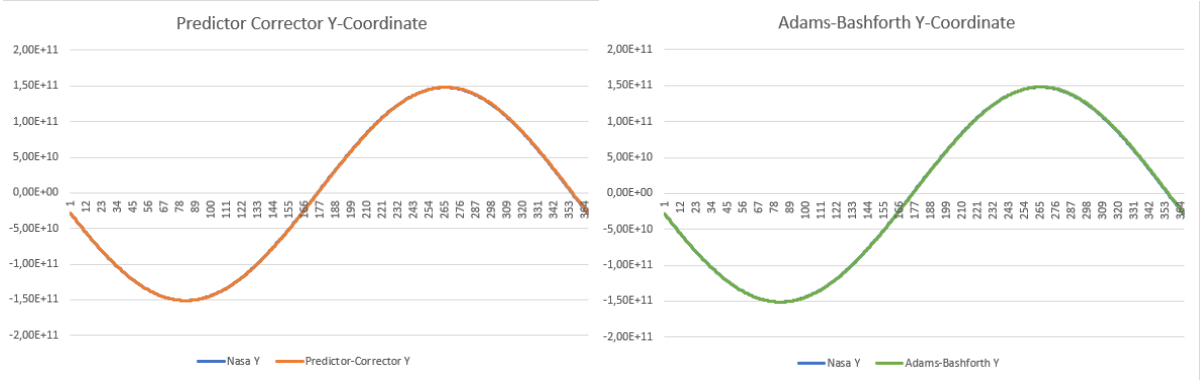
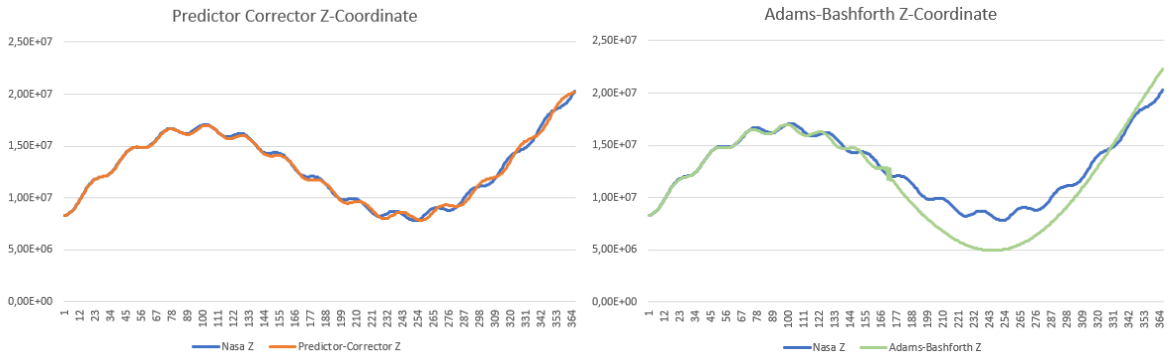


Figure 32 and 33



Annex 5: Lander GUI – Feedback controller

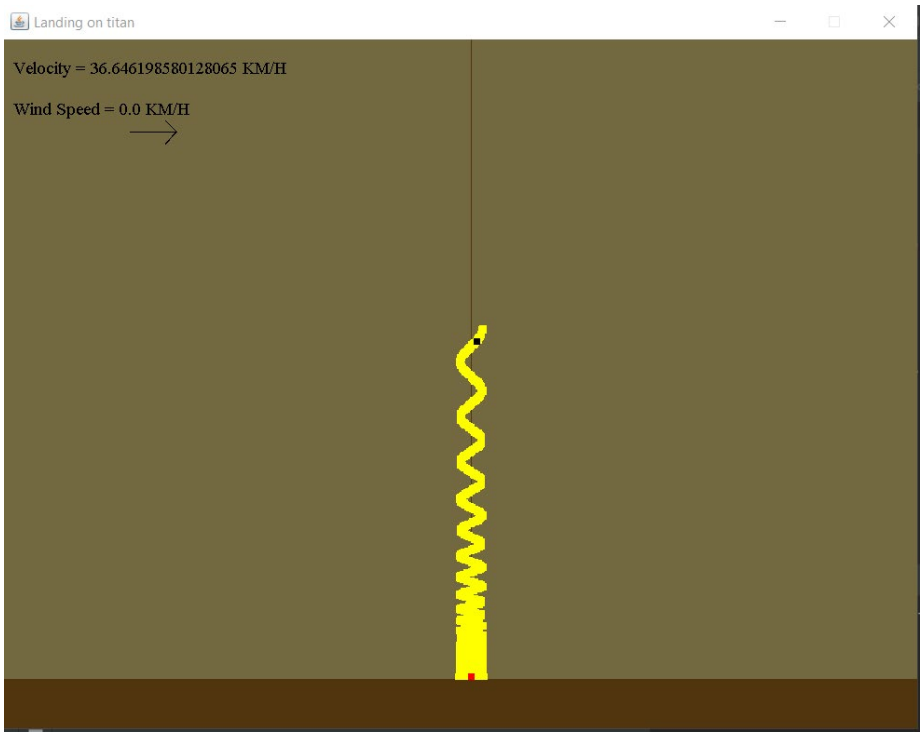


Figure 36