MAASTRICHT UNIVERSITY

DEPARTMENT OF DATA SCIENCE AND KNOWLEDGE ENGINEERING

PROJECT 1.1 – BLOCK 3

A Titan Space Odyssey

GROUP 10

Daniel Oliveira i6219581	Maria Thurn i6263844
Dumitru Versebeniuc i6257660	Britt Schmitz i6235053
Gaspar Kuper i6253413	

June 22, 2021

Project coordinator: Katharina Schüller

Overview

1. Abstract
2. Introduction
3. Methods
    - 3.1. Overview
    - 3.2. Ordinary Differential Equation (ODE) Solvers
        - 3.2.1. Euler Solver: Implicit/Symplectic
        - 3.2.2. Verlet Velocity
        - 3.2.3. Runge Kutta:Brute Force algorithm
    - 3.3. Newton Raphson
    - 3.4. Stochastic Wind Model
    - 3.5. Controllers
    - 3.6. Fuel Calculations
    - 3.7. Assumptions
4. Implementation
5. Experiments
6. Results
7. Discussion
8. Conclusion
9. References

1. Abstract

Space exploration has been a fascination to humankind for many years. Reaching distant planets and exploring them. This paper tries to respond to the questions, to what extent can it be claimed that an ODE solver is more accurate than others? Which algorithm is best suitable to optimize the flight trajectory and what are the ideal parameters for the probe? How does the wind influence the landing accuracy? A focus is set in investigating three different areas: physics of the model, calculation of trajectory, and calculation of landing. For each part of the investigation different algorithms were used, four Ordinary Differential Equation (ODE) solvers were used for the physics model, two algorithms for the trajectory, and two controllers for the landing. With the investigation one is able to conclude that the Verlet-Velocity is the more accurate ODE solver, a brute force algorithm was the only successful algorithm for finding trajectory, and the closed loop controller has a better success rate on achieving landing.

2. INTRODUCTION

With the recent advancements in aerospace engineering the exploration of other planets has become accessible. Without these advancements many services/goods like weather tracking, global position system (GPS), etc would not exist. One of the problems that come with the advancements/exploration of outer space is the planning for these missions. This task is faced by all space agencies and in order to overcome these problems, an extensive amount of simulations/experiments are necessary. It is quite important that nothing goes wrong in these missions as millions of dollars tend to be involved and a simple mistake can be detrimental. This is what that scope of the investigation tries to foresee, the planning of an expedition to Titan and back.

In this report three main Ordinary Differential Equation (ODE) solvers are used, Euler Method, Verlet, and Runge-Kutta. The difference between these solvers is the structure of their equations. When analysing/comparing ODE solvers, what will classify one as superior is how close one is from the real result. The results that are used to compare are taken from the Nasa Horizons[1].

When thinking about the trajectory, many factors should be taken into consideration. Two distinct algorithms are used for the calculation of trajectory: brute force algorithm and Newton's Raphson. The brute force algorithm has a more exhaustive approach to the problem while Newton's Raphson is more of an iterative approach to the problem. The comparison to be made between both algorithms is primarily on their runtime.

For the landing mission two controllers are used: open-loop controller and closed-loop controller. The investigation is simplified by only taking into account two dimensions (third dimension would not play a role necessarily). The main difference between both controllers is the input that they receive prior to their execution. For the open loop controller the only input it takes in is the amount of thrust that needs to be used, while the closed loop controller takes an additional input of the state of the probe.

To investigate a possible manned mission to Titan one must successfully implement a physics engine, an algorithm for the calculation of the trajectory of the probe, and a controller for a successful landing. The questions that are investigated in this paper are the following. To what extent can it be claimed that an ODE solver is more accurate than others? Which algorithm is best suitable to optimize the flight trajectory and what are the ideal parameters for the probe? How does the wind influence the landing accuracy?

3. METHODS

3.2 Ordinary Differential Equation (ODE) Solvers

Many physical phenomena can be modeled by differential equations, that is, equations in which the unknown is not a number, but a function, and include one or more derivatives of this function. However, in practice, for a general differential equation, it is often difficult or even impossible to find a formula for an exact solution. Therefore, it is necessary to use a numerical integrator, which is an algorithmic method for calculating an approximate solution. There exists a wide range of algorithms for solving an ordinary differential equation and now we present a few examples of such methods.

3.2.1 Euler: Implicit/Symplectic Euler method

Implicit Euler method or backward Euler method is a numerical method that solves first an ordinary differential equation $\frac{dy}{dt} = f(t, y)$ where $y(t_0) = y_0$ is the initial state at time t_0. It is one of the simplest and basic ODE-solvers. The implicit Euler method computes the derivation using:

$$\frac{dy}{dt} = f(t, y)$$

$$y_{k+1} = y_k + f(t_{k+1}, y_{k+1})\Delta t$$

The implicit Euler method has order one. This means that the error at a specific time t is $O(h)$. For implementation of this method, requires only one function evaluation at each iteration, which makes it easy to implement.

Symplectic Euler is a modification of the Euler method. It is a symplectic integrator, which is better for calculating trajectories objects in the solar system, hence it calculates the approximations better than the implicit Euler method. The symplectic Euler method produces the approximations using:

$$v_{k+1} = v_k + f(t_k, y_k)\ \Delta t$$

$$x_{k+1} = x_n + v_{k+1}\Delta t$$

Where v_k is the velocity of the state and x_k is the position of the state.

The semi-implicit Euler is a first-order integrator, therefore it has the same error with the implicit Euler method. However, the symplectic integrator gives more accurate results due to it almost conserving the energy. In addition, this method leads to the Verlet-Velocity (leapfrog integration), which increases the accuracy of calculations.

3.2.2 Verlet-Velocity (Leapfrog integration)

In leapfrog integration, similar to velocity Verlet, the synchronised form of the equations for updating position and velocity are:

$$v_{k+1/2} = v_k + \frac{\Delta t}{2} a_k$$

$$x_{k+1} = x_k + v_{k+1/2} \Delta t$$

$$v_{k+1} = v_{k+1/2} + \frac{\Delta t}{2} a_{k+1}$$

Where v_k is the velocity and x_k is the position of the object

a_k is the acceleration which was calculated:

$$a = \frac{F}{m}$$

Where F is Newton's law of universal gravity:

$$F = \sum_{j \neq i} G \times m_i \times m_j \times \frac{x_j - x_i}{||x_i - x_j||^3}$$

Leapfrog integration is a second-order method. Unlike Euler integration, the main advantage of this method is its symplectic nature, therefore it is stable for oscillatory motion, which implies that it conserves the energy of dynamical systems.

3.2.3 4th order Runge-Kutta

The fourth order runge kutta method is another iterative solution to an ordinary differential equation $\frac{dy}{dt} = f(t, y)$ where $y(t_0) = y_0$ is the initial state at time t_0. Four function evaluations k_i, where $i \in [1, 4]$, are required at each time t, making it more difficult to implement. The results are however expected to be more precise. Each respective k is calculated using the following formulae, including the current step size h, the current time t_n, and the current state y_n:

$$k_1 = f\left(t_n,\ y_n\right)$$

$$k_2 = f(t_n + \frac{h}{2},\ y_n + h\frac{k_1}{2})$$

$$k_3 = f(t_n + \frac{h}{2},\ y_n + h\frac{k_2}{2})$$

$$k_4 = f(t_n + h,\ y_n + hk_3)$$

Once we have these function approximations, the next state y_{n+1} at the following time t_{n+1} can be approximated using:

$$y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4)$$

As mentioned before, this is a fourth order method meaning that it has a local truncation error at any time t of at most $O(h^5)$.

3.3 Algorithms

The correct algorithms for the different tasks are crucial for a successful mission. We chose the respective algorithms based on their runtime and accuracy. In the following chapter we describe the motivation and usage of the most important algorithms.

3.3.1 Brute force algorithm

To find a good trajectory for the probe and to reach a specific position, a brute force algorithm can be used. First, it generates a vector at a specific distance of the object we are aiming at. This is useful to reach the surface of a planet, or to go into orbit. Then, it computes the trajectory of the probe repeatedly. Between each computation, based on the final position of the probe, x, y and z values of the initial velocity vector are increased or decreased by the current step size value. When no progress is made, the value of the step size is divided by two, until it reaches 1E-15. At this point, the brute force algorithm has a good initial velocity.

3.3.2 Newton Raphson

Newton Raphson method is a root finding algorithm and is an iterative method. a 3-dimensional model is used, so a multivariate Newton-Raphson method must be used. This method is used to calculate an initial velocity.

To find the initial velocity, the calculation is repeated , using the formula below, until the result is accurate enough:

$$V_{k+1} = V_k - Dg(V_k)^{-1} * g(V_k)$$

V_k represents the velocity, $Dg(V_k)$ is the jacobian matrix and $g(V_k)$ is the distance between titan and the probe.

To compute the Jacobian, we need to find the partial derivatives of the velocity vector using the formula below:

$$f'(x) = (f(x + h) - f(x - h))/2h$$

3.4 Stochastic Wind Model

The wind is the most important part of the landing as wind can influence the displacement of the trajectory of the lander. Titan has a dense atmosphere that extends approximately 600 kilometers from the surface. Although the model of the wind is stochastic, NASA data makes it more realistic. The Huygens spacecraft landed on Titan on January 12, 2005, in the process of landing, it encountered its maximum wind speed about ten minutes after beginning its descent. The speed was roughly 120 meters per second and was measured at an altitude of about 120 kilometers. As the probe dropped below 60 kilometers, the wind speed dropped too. During the final seven kilometers of the descent, Huygens encountered wind speeds of just a few meters per second, allowing it to drop in an almost straight line. At the surface of Titan, there was nothing but a gentle breeze of just 0.3 meters per second. [3]

During the landing, the direction of the wind can be changed, since, in the NASA mission, the direction of the wind changed only twice, the stochastic model has a 1% chance to change wind direction. In addition, the maximum speed on Titan was recorded at a height of about 120 kilometers, and it was 120 meters per second, and on the surface of Titan, 0.3 meters per second. The stochastic model uses the assumption that 120 kilometers equals 120 meters per second and every kilometer down, the speed of the wind decreases by proportion 1 kilometers minus 1 meter

per seconds from the maximum speed. However, it's unrealistic that the speed of the wind is increased by the height above, where the maximum speed of the wind was fixed, so the formula 1 kilometer minus 1 meter per second from the maximum speed is also effective here. For the simulation of the stochastic model, the wind speed is deviated by 20% from the calculations. After this the wind force is calculated, it is multiplied on the area of the lander, and divided on the lander mass.

3.5 Controller

After the probe has reached the orbit of Titan, it needs to safely land the module onto the surface. A landing is considered safe when $y = 0$ within the following tolerances for the horizontal position x, the angle of rotation θ, the horizontal velocity $\dot{x}$, the vertical velocity $\dot{y}$, and the angular velocity $\dot{\theta}$:

$$|x| \leq 0.1$$

$$|\theta \; mod \; 2\pi| \leq 0.02$$

$$|\dot{x}| \leq 0.1$$

$$|\dot{y}| \leq 0.1$$

$$|\dot{\theta}| \leq 0.1$$

In order to increase the chances for a safe landing, the landing is split into four distinct phases. First, we activate torque to position the lander in a horizontal state such that $\theta = -90$ degrees or $\theta = 90$degrees, depending on how the horizontal position of the lander should be corrected. Once this is the case, the main thruster is activated to obtain a horizontal position $x = 0$. Then the torque is again activated to rotate the lander back to a vertical position $\theta = 0$and finally, the main thruster is activated once more in order to reach the surface $y = 0$ with a vertical velocity within the specified range to make a safe landing.

For calculation the position and velocity were calculated with the following formula, where u and $v$respectively are the power of the thrust and the torque:

$$x_{t+1} = x_t + v_t \times timestep + v_{acc} \times \frac{timestep^2}{2}$$

$$v_{t+1} = v_t + v_{acc} \times timestep$$

Where v_{acc} equals:

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

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

$$\ddot{\theta} = v \times I$$

I is the moment of inertia of the lander. Assuming the moment of inertia I to be constant, though in practice it will decrease slightly due to fuel loss, the mathematics are not affected.

3.5.1 Open-loop controller

An open-loop or feedforward controller is the simplest approach to achieve a landing on Titan. However, since it's actions are independent of the desired output or result, it is not the best solution to the problem. In order to use this controller to make a safe landing on Titan, many attempts may be necessary before a successful landing is found without crashing the module. The input values for both thrust and torque are predetermined and thus cannot be altered during the landing.

3.5.2 Feedback controller

A closed-loop or feedback controller takes into account the current state of the module. Meaning, that the algorithm is significantly less simple than the one previously described. After each timestep between the release of the lander from the probe and its landing, the algorithm determines how much thrust and torque should be applied to achieve a safe landing on the surface of Titan. This allows for a safe landing with a single attempt because the thrust and torque are determined based on how the current state at each step compares to the desired final values.

3.6 Fuel Calculation

Thrust was calculated with the following formula:

$$F = m \times a$$

where F is the thrust and a is the acceleration

$$a = \frac{V_{req} - V_{initial}}{\Delta t}$$

Mass rate was calculated with the following formula:

$$\dot{m} = \frac{F}{V_{exhaust}} \qquad \dot{m}_{max} = \frac{F_{max}}{V_{exhaust}}$$

Where $V_{exhaust}$ depends on the type of fuel and F_{max} is the maximum thrust

Then the time needed t_{req} to get to the right velocity with the maximum thrust was calculated with the following formula:

$$t_{req} = \frac{\dot{m}}{\dot{m}_{max}} \times \Delta t$$

Finally, the mass of fuel burned was calculated using:

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

3.7 Assumptions

Some simplifications were made in the experiment in order to reduce the span. These changes do not affect the general approach to the problem, only takes away potential additional steps. It is critical to take these point through and analyze their effect:

1. **Planet Bodies:** The model of the Solar System is limited to eleven bodies composed of Sun, Mercury, Venus, Earth, Moon(earth's), Mars, Jupiter, Saturn, Uranus, Titan, and probe.
2. **Shape of lander module:** The lander has a sphere shape for the constant area in each state, which is useful for calculation how the wind speed influences the lander .
3. **Landing in 2D:** Only consider the X and the Y dimensions.
4. **Flat surface:** It is implied that the position in which the thruster tries to land on is flat with no obstacles (caters, rivers, lakes , etc...) or incline gradient.

4. IMPLEMENTATION

4.1 Trajectory Algorithms

Pseudocode 1. Algorithm 1: Calculate the initial velocity for the probe

```
Input:None
Output: The optimal velocity to reach the aimed object
Begin
SET StepSize = 10000
while StepSize>1E-15 Do
    call function trajectory to find lastPositionOfAimedObject and lastPositionOfProbe

    if lastPositionOfProbe.getX <lastPositionOfAimedObject.getX
        add StepSize to X of velocityVector
    else if lastPositionOfProbe.getX >lastPositionOfAimedObject.getX
        subtract StepSize to X of velocityVector
    end
    if lastPositionOfProbe.getY <lastPositionOfAimedObject.getY
        add StepSize to Y of velocityVector
    else if lastPositionOfProbe.getY >lastPositionOfAimedObject.getY
        subtract StepSize to Y of velocityVector
    end
    if lastPositionOfProbe.getZ <lastPositionOfAimedObject.getZ
        add StepSize to Z of velocityVector
    else if lastPositionOfProbe.getZ >lastPositionOfAimedObject.getZ
        subtract StepSize to Z of velocityVector
    end

    if velocity is Repeating
        StepSize = StepSize/2
    end
end
```

4.2 Controllers

The second algorithm compiles all states of the lander during the different phases into one list.

Pseudocode 2. **Algorithm 2:** calculatePhase(state, duration)

```
Input: state of a lander, duration to accelerate in order to reach desired velocity
Output: a list of states of a lander
Begin
SET resultList
IF state.position =/= 0
        CALL correctX with state, duration RETURNING landerStates
        add landerStates to resultList
ENDIF
determine latest state s
CALL phaseLanding with s, 0.1 RETURNING landerStates
add landerStates to resultList
RETURN resultList
End
```

The third algorithm is called only when the horizontal position of the probe needs to be adjusted, it first rotates the lander by 90 degrees, depending on the type of correction we need to apply. Then the thruster is activated in order to reach horizontal position 0.

Pseudocode 3. **Algorithm 3:** correctX(state, duration)

```
Input: state of a lander, duration to accelerate in order to reach desired velocity
Output: a list of states of a lander
Begin
SET resultList
IF state.position > 0
        THEN angle ← -90
        ELSE angle ← 90
ENDIF
CALL rotationPhase with state, duration, 0.1, angle RETURNING landerStates1
determine latest state s in landerStates1
determine time t needed to reach the maximum velocity of the lander
CALL phaseSpeedUp with s, t, 0.1 RETURNING landerStates2
IF angle = -90
        THEN newAngle -270
        ELSE newAngle 270
ENDIF
determine latest state s in landerStates2
CALL rotationPhase with s, duration, 0.1, newAngle RETURNING landerStates3
determine latest state s in landerStates3
CALL phaseToSlowDownX with s, 0.1 RETURNING landerStates4
IF newAngle = -270
        THEN lastAngle = -360
        ELSE lastAngle = 360
ENDIF
determine latest state s in landerStates 4
CALL rotationPhase with s, duration, 0.1, lastAngle RETURNING landerStates5
add landerStates1 to resultList
add landerStates2 to resultList
add landerStates3 to resultList
add landerStates4 to resultList
add landerStates5 to resultList
RETURN resultList
End
```

The fourth algorithm determines the required thrust to apply to the lander in order to reach a given angular rotation of the module, while also reaching an angular velocity of 0 when this rotation is obtained.

Pseudocode 4. **Algorithm 4:** rotationPhase(state, time, h, angle)

```
Input: state of a lander, time during which we can accelerate, step size h, desired angle
Output: list of states of a lander
Begin
SET resultList
determine maximum angular velocity Umax
torque ← Umax/time
FOR i = 0 to time
        determine latest state s in resultList
        CALL step with s, 0, torque, h RETURNING newState
        add newState to resultList
        update fuel for newState
ENDFOR
determine torque v in opposite direction to stop the rotation at angle
determine latest state s in resultList
WHILE s.velocity > 1e-5
        CALL step with s, 0, v, h RETURNING newState
        add newState to resultList
        s ← newState
        update fuel for newState
ENDWHILE
RETURN result
End
```

The fifth algorithm allows the landing module to reach its maximum velocity within the given time.

Pseudocode 5. **Algorithm 5:** phaseSpeedUp(state, time, h)

```
Input: state of a lander, time to reach maximum velocity, step size h
Output: list of states of a lander
Begin
SET resultList
determine thrust needed to reach horizontal position 0
determine latest state s in resultList
FOR i = 0 to (time – 120)
        CALL step with s, thrust, 0, h RETURNING newState
        add newState to resultList
        update fuel for newState
ENDFOR
RETURN resultList
End
```

The sixth algorithm slows the landing module down in order to reach the desired horizontal position with a horizontal velocity of 0.

Pseudocode 6. **Algorithm 6:** phaseToSlowDownX(state, h)

```
Input: state of a lander, step size h
Output: list of states of a lander
Begin
SET resultList
determine thrust needed to reach horizontal velocity 0 at the time of reaching horizontal position 0
determine latest state s in resultList
WHILE s.horizontalVelocity > 1e-5
        CALL step s, thrust, 0, h RETURNING newState
        add newState to resultList
        s ← newState
        update fuel for newState
ENDWHILE
RETURN resultList
End
```

The seventh algorithm is the last phase of the landing. The landing module goes into free fall towards the surface at first, then we activate the thrust in order to land on the surface with a vertical velocity of 0, this makes sure that the landing module does not crash at landing.

Pseudocode 7. **Algorithm 7:** phaseLanding(state, h)

```
Input: state of a lander, step size h
SET resultList
determine latest state s in resultList
WHILE s.verticalVelocity > 50000
        CALL step with s, 0, 0, h RETURNING newState
        add newState to resultList
        s ← newState
ENDWHILE
determine thrust to reach vertical position 0 with vertical velocity 0
determine the time of landing tl
FOR i = 0 to tl
        CALL step with s, thrust, 0, h RETURNING newState
        add newState to resultList
        s ← newState
        update fuel for newState
ENDFOR
CALL step with s, thrust, 0, tl mod h RETURNING newState
add newState to resultList
RETURN resultList
```

The eight method is a simple step method, which updates the state of the lander at each time t.

Pseudocode 8. **Algorithm 8:** set(state, thrust, torque, h)

```
Input: state of a lander, main thrust for acceleration, torque for rotation, step size h
Output: new state of the lander
Begin
lander ← clone of state
update position of lander to its new position
determine new velocity v of lander
IF wind =/= 0
        THEN add wind to v
ENDIF
update velocity of lander to v
update angle of lander to new angle
update angularRotation of lander to new Angular rotation
RETURN lander
End
```

5. EXPERIMENTS

5.1 ODE Solver

5.1.1 Step sizes

After implementing four different solvers, namely Symplectic Euler, implicit Euler, Verlet velocity, and Runge Kutta, their efficiency and precision must be compared. A focus is set on two parameters: the time efficiency and the accuracy of the results.

To compare the efficiency of the solvers, the coordinates of the NASA HORIZONS system were taken to be the optimal solution. Then a comparison is made between all solvers' calculations and the Nasa coordinates.

5.1.2 Runtime

A comparison is made between the different ODE solvers based on their runtime. The experiment uses a step size of one day (86400 seconds).

5.2 Trajectory: determine runtime of Brute Force

Find the runtime of the Brute force algorithm. The sample size used is only one since there is no randomization in this brute force.

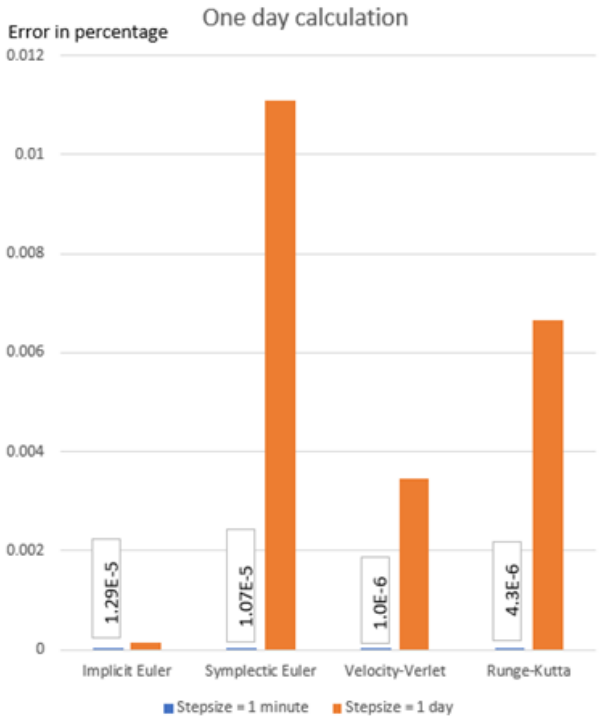
5.3 Landing: Influence of the wind on the landing accuracy

Find the rate of success of the landing module successfully landing on the titan surface with coordinates (0,0) with an acceptance range of x. The experiment is composed of ten trials each in which 100,000 samples will be taken. These calculations are using initial position parameters computed by a brute force algorithm: x position is 531250 meters and y position is 240000 meters.

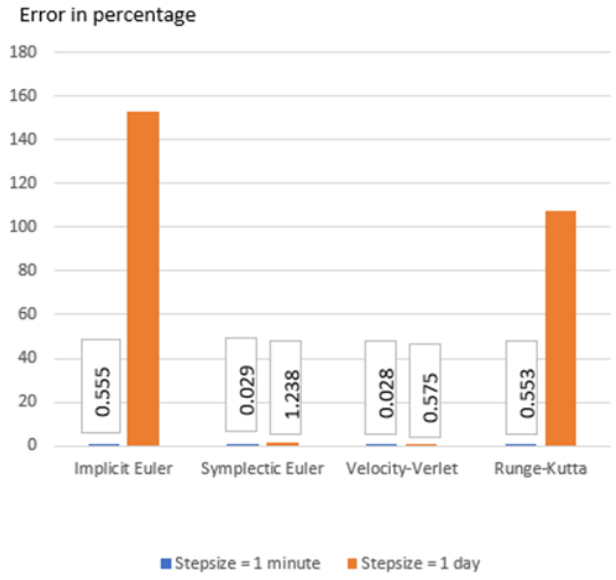
6. Results

6.1 Results of 5.1

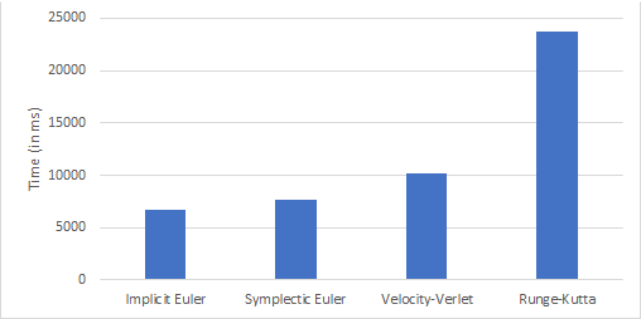
Graph 3. Error percentage of solvers for a day's calculation (5.1.1)



Graph 4. Error percentage of solvers for a day's calculation (5.1.1)



Graph 5. Runtime comparison of solvers (5.1.2)



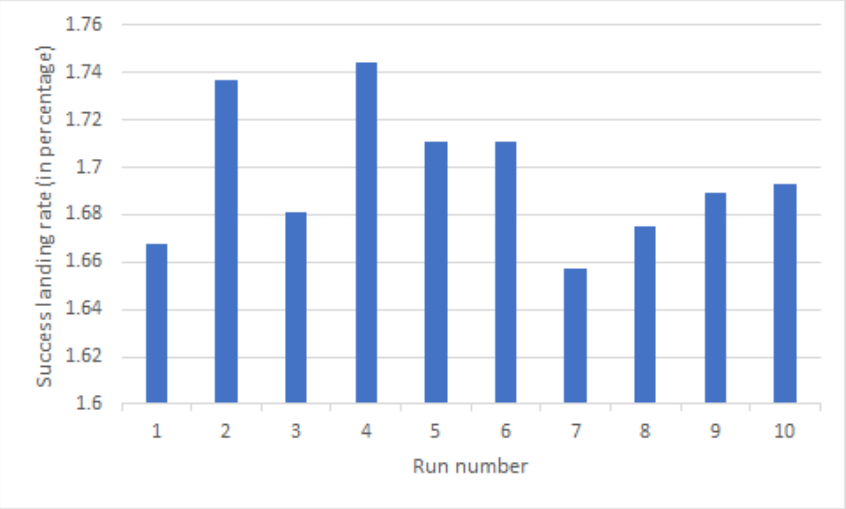
6.2 Results of 5.2

Initial Velocity calculation for mission to Titan: 11 minutes and 15 seconds. Accuracy is 3.66E-4 meters.

Initial Velocity calculation for return mission: takes 13 minutes 58 seconds. Accuracy is 6.0E-5.

6.3 Results of 5.3

Graph 6. Success landing rate of the closed loop controller per 100,000 simulations



1.6966% success rate on average

(without wind, success rate is 100%)

7. Discussion & Conclusion

An algorithm's success is not only defined by how it completes the task. It is also crucial that an algorithm performs under any condition and does not face any limitations. When trying to differentiate different algorithms and their performance, three aspects must be analyzed: runtime, storage, and accuracy. It is important since improving these allows the algorithm to achieve a solution with minimal requirements. From the experiments a focus is set in runtime and accuracy of solutions.

For experiment 5.1, a higher interest is set in the one year calculation as a smaller step size leads to more computational power required. As seen within graph 4, the Velocity-Verlet solver is the most accurate one with either a small or a big step size. We can contribute to the success of the Velocity-Verlet due to it being a symplectic integrator and that it conserves energy. Symplectic Euler

and Velocity Verlet are also relatively efficient. As for the implicit solver it can be concluded that it is highly inaccurate for bigger step sizes. Fourth order Runge-Kutta is the most accurate solver with the small step size, but due to their not efficient time complexity, it can be used as an adaptive solver for launching the probe, entering to the orbit and leaving the orbit.

For the comparison of different calculations of trajectory, a possible improvement that could have been made is to better understand the concept of Newton-Raphson and how to implement it. This would have allowed us to compare and analyze the calculation of the trajectory, and use fewer steps to predict trajectory. We are able to see that it still takes a decent amount of time for the calculation of the brute force algorithm but is always successful. Regarding the parameters it was calculated that the total fuel necessary for the mission would be 55,2014.76 kg and the duration of the trip would be 3 years 362 days 13 hours 5 minutes 16 second (approximately 1457 days).

When comparing both controllers, we can see that the closed-loop controller has a better landing accuracy than the open-loop controller. The reason is that the open loop controller cannot change its trajectory while landing. Considering the varying conditions in Titan's atmosphere, such as the wind, in which the landing module is found, a fixed set of torque and thrust cannot be predetermined such that every landing is safe. Only a percentage actually achieve the landing zone successfully, roughly 1.6% seen in experiment 6.3.

Taking the RQs' into account, to what extent can it be claimed that an ODE solver is more accurate than others? Which algorithm is best suitable to optimize the flight trajectory and what are the ideal parameters for the probe? Can be claimed that the Runge-Kutta is the more accurate solver but if we were to consider runtime the Verlet-Velocity would be the better ODE solver as it loses a bit of accuracy to gain substantially in runtime. As for the trajectory calculation due to time constraints the brute force algorithm is the best/only algorithm for trajectory with the probe using 55,2014.76 kg of fuel. Finally the controller, as for which one is best we can clearly leave it to closed-loop. This is due to the closed loop actually taking the state of the probe into consideration.

9. References

[1] Jet Propulsion Laboratory. (n.d.). *HORIZONS web-interface*. National Aeronautics and Space Administration. https://ssd.jpl.nasa.gov/horizons.cgi

[2] Tokano, T., & Lorenz, R. D. (2007, January 6). *The way the wind blows on Titan*. National Aeronautics and Space Administration. https://www.nasa.gov/mission_pages/cassini/whycassini/cassinif-20070601-05.html

[3] National Aeronautics and Space Administration. (2021, May 13). *Torque (moment)*. https://www.grc.nasa.gov/www/k-12/rocket/torque.html

[4] National Aeronautics and Space Administration. (2018, November 20). *Gimbalded thrust.* https://www.grc.nasa.gov/www/k-12/rocket/gimbaled.html