Rocket Simulation

Erik Dahlström, erida600
Filip Malm-Bägén, filma379
Filip Hamrelius, filha243
Oliver Lundin, olilu316

March 14, 2023

Contents

List of Figures

Chapter 1

Introduction

In this chapter the project is introduced and a description of how to run the simulation is provided.

1.1 Introduction

This report goes through the process of developing and implementing a fundamental rocket simulation. The goal is to simulate the rocket in real time with as realistic parameters as possible, such as the force of drag, the shifting of thrust and the different stages of the rocket. The rocket which is to be simulated is Spacex's *Falcon 9* rocket. The first simulation will be done in *MATLAB*. When this has been managed the group will move on to demonstrate it using *JavaScript*.

1.2 Delimitations

A rocket simulation is a complicated subject which takes countless of parameters into consideration. The project does therefore not have usual delimitations, but focuses on the selected parameters in addition to a movement in a 3D space. The simulation depends on the density of the air, the thrust of the rocket, the loss of mass through time and through the different stages of the rocket, the decrease of gravitational pull and the drag force on the rocket.

This project does not consider wind, weather, humidity, fluid dynamics for the fuel, axial drag coefficient, side force coefficient, launch site, etcetera.

1.3 Run Simulation

The following paragraphs describes how one should pursue to run both the MATLAB code and the JavaScript application locally.

1.3.1 MATLAB

Version 2019 or higher is required in order to run the simulation in MATLAB. The code can be cloned from Rocket-MATLAB. Run the m-file *sim.m* to run the simulation. Run the MLX-file:

sim.mlx to see the numerical result, that is, only the velocity and position over time. It is recommended to have at least 8 GB of RAM to be able to run the simulation smoothly.

1.3.2 JavaScript

To run the JavaScript simulation, node.js run-time environment must be installed. The code can be cloned using git from rocket-simulation. The 3D package *Three.js* must be installed at the project directory in order to start the simulation. Simply write

```
npm install three
```

to configure the simulation and run

```
npm run dev
```

to start the simulation on your machine.

The simulation will also run directly in the web browser by simply clicking here.

Chapter 2

Rocket Model

In this chapter the system, model, numerical implementation and animation implementation is described.

2.1 Mathematical Model

The initial description of the system was derived in three dimensions, meaning a spherical coordinate system was needed. The three characteristics used by the system are the radial distance from the origin r, the angle θ measured from the positive z-axis, and the angle φ measured from the positive x-axis in the xy-plane. A graphical representation of the spherical coordinate system can be viewed in figure 2.1.

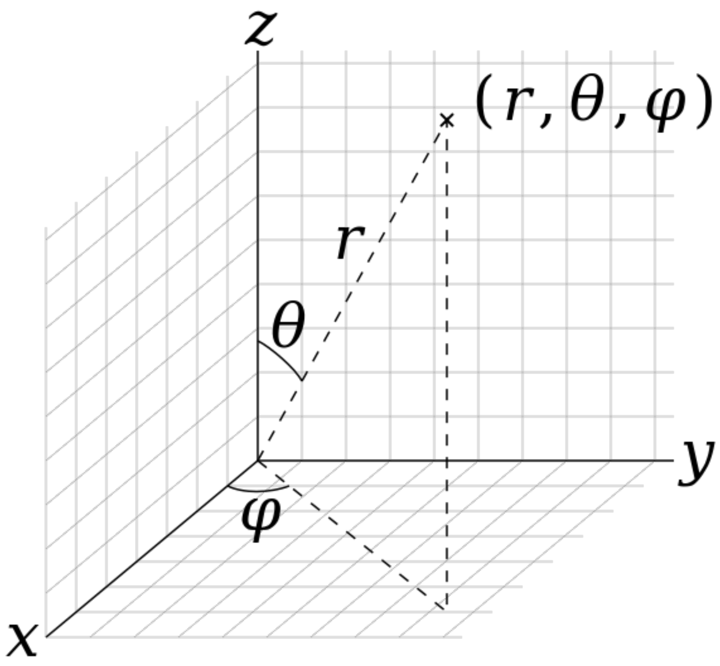

Figure 2.1: Spherical Coordinate System
Wikimedia, 2019

A mathematical description of the motion of the rocket is required in order to simulate it. The final equation is derived from Newton's Second Law of Motion.

$$F(t) = m(t) \cdot a(t) \tag{2.1}$$

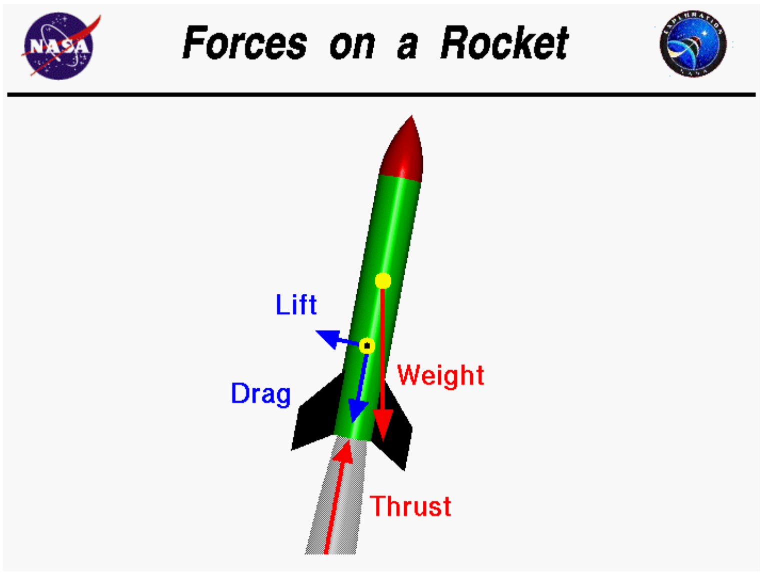

Figure 2.2: The forces acting on the rocket
Nasa, 2021

Thereafter the different directions of the force were taken into consideration. All of the forces acting on the rocket can be seen in figure 2.2.

$$F_{thrust}(p(h)) - F_{drag}(\rho(p(h), T(h), h) - m(t) \cdot g(h) = m(t) \cdot a(t) \tag{2.2}$$

Equation 2.2 describes how the forces acts on the rocket in one dimension where

$$F_{thrust}(p(h)) = M \cdot V_e + (P_e - p(h)) \cdot A_e \tag{2.3}$$

and

$$F_{drag}(\rho(p(h), T(h), h), v) = \frac{\rho v^2 C_d A}{2} \tag{2.4}$$

M represents the mass flow rate, V_e is the velocity of exhaust, $p(h)$ is the atmospheric pressure, P_e is the exhaust pressure and A_e is the area of exhaust, $\rho(p, T, h)$ is the air density, v is the velocity of the rocket, C_d is the drag coefficient of the rocket and A is the reference area of the rocket. The air density ρ is found using

$$\rho(p(h), T(h), h) = \frac{p(h)}{0.2869 \cdot (T(h) + 273.1)} \tag{2.5}$$

where

$$p(h), T(h) = \begin{cases} T(h) = 15.04 - 0.00649 \cdot h & \\ p(h) = 101 \cdot \frac{T(h) \cdot 273.1}{288.08}^{5.256} & \text{if } h < 11000 \text{ (Troposphere)} \\ & \\ T(h) = -56.46 & \\ p(h) = 22.65 \cdot e^{1.73 - 0.000157 \cdot h} & \text{if } 11000 < h < 25000 \text{ (Lower Stratosphere)} \\ & \\ T(h) = -131.21 + 0.00299 \cdot h & \\ p(h) = 2.488 \cdot \frac{T(h) \cdot 273.1}{216.6}^{-11.388} & \text{if } h > 25000 \text{ (Upper Stratosphere)} \end{cases} \tag{2.6}$$

where $T(h)$ represents the temperature in Celsius. The gravitational force on the rocket is found using

$$g(h) = G\frac{R}{(R+h)^2} \tag{2.7}$$

where h is the altitude of the rocket, G is the standard gravitational acceleration and R is the radius of the earth. The mass of the rocket, $m(t)$, is calculated by

$$m(t) = \begin{cases} m_{R1} + m_P + m_f(t) & \text{if } m_f(t) > 0 \\ m_{R2} + m_P & \text{if } m_f(t) = 0 \end{cases} \tag{2.8}$$

where m_{R1} and m_{R2} represents the first stage rocket mass and the second stage rocket mass respectively. m_P represents the payload and $m_f(t)$ represents the fuel left on the rocket which can be described using

$$m_f(t) = m_0 - b_r \cdot t \tag{2.9}$$

where m_0 is the initial mass of the fuel, b_r is the rockets burn rate and t is the time. Equation 2.2 describes the motion of the rocket solely along the z axis, due to the inclusive of gravity. This can be fixed by splitting the equation into three different components, where each component describes each dimension.

$$\begin{aligned} F_{trust_x}(p(h)) - F_{drag_x}(\rho(p(h), T(h), h) &= m(t) \cdot a_x(t) \\ F_{trust_y}(p(h)) - F_{drag_y}(\rho(p(h), T(h), h) &= m(t) \cdot a_y(t) \\ F_{trust_z}(p(h)) - F_{drag_z}(\rho(p(h), T(h), h) - m(t) \cdot g(h) &= m(t) \cdot a_z(t) \end{aligned} \tag{2.10}$$

Thereafter spherical coordinates had to be implemented in order for the model to be simulated in three dimensions.

$$\begin{aligned} F_{thrust_x}(p(h)) \cdot sin(\theta) \cdot cos(\varphi) - F_{drag_x}(\rho(p(h), T(h), h) &= m(t) \cdot a_x(t) \\ F_{thrust_y}(p(h)) \cdot sin(\theta) \cdot sin(\varphi) - F_{drag_y}(\rho(p(h), T(h), h) &= m(t) \cdot a_y(t) \\ F_{thrust_z}(p(h)) \cdot cos(\theta) - F_{drag_z}(\rho(p(h), T(h), h) - m(t) \cdot g(h) &= m(t) \cdot a_z(t) \end{aligned} \tag{2.11}$$

Finally, the acceleration is factored out in order to describe the rockets acceleration.

$$
a_x(t) = \frac{F_{thrust_x}(p(h)) \cdot sin(\theta) \cdot cos(\varphi) - F_{drag_x}(\rho(p(h), T(h), h)}{m(t)}
$$

$$
a_y(t) = \frac{F_{thrust_y}(p(h)) \cdot sin(\theta) \cdot sin(\varphi) - F_{drag_y}(\rho(p(h), T(h), h)}{m(t)} \tag{2.12}
$$

$$
a_z(t) = \frac{F_{thrust_z}(p(h)) \cdot cos(\theta) - F_{drag_z}(\rho(p(h), T(h), h) - m(t) \cdot g(h)}{m(t)}
$$

Equation 2.12 gives the final equation which describes the acceleration of the rocket. The constants used in the equations are revealed in A.1.

2.2 Numerical Method Implementation

The model was first investigated on MATLAB's integrated ODE solver, *ODE45*, during the testing stage. ODE45 is an iterative solver in turn based on *Euler approximations*. The JavaScript simulation uses the numerical method Runge-Kutta order 4, *RK4*. The fourth-order Runge-Kutta method works by using several k values which can be determined by iterating a set and carrying on from the previous point while partially extracting a slope at each step. The last stage determines the next point by averaging the results of the other phases. See equation 2.13.

$$
\begin{aligned}
t_{n+1} &= t_n + h \\
k_1 &= f(y_n) \\
k_2 &= f(y_n + h\frac{k_1}{2}) \\
k_3 &= f(y_n + h\frac{k_2}{2}) \\
k_4 &= f(y_n h_3) \\
y_{n+1} &= y + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h
\end{aligned} \tag{2.13}
$$

The Runge-Kutta 4 function works by taking the current position and velocity of the rocket in every dimension and outputs the next position and velocity in every dimension of the rocket. Thus, the function takes an 1 by 6 array as an input and gives an 1 by 6 array as an output. RK4 allowed the equation to be solved in one iteration for each time step. Using only a Euler approximation the solver would have to be called 3 times for each value. Therefore RK4 helped with simplicity.

The final model, developed in JavaScript, uses an implemented version of RK4 unlike the MATLAB implementation.

Pseudocode 1 Runge Kutta order 4 - JavaScript : Implementation

```
Input: $d_{xyz}$ - array of current position and velocity, $dt$ - seconds since last frame loaded
Output: $d_{xyz+1}$ - array of the following position and velocity

  $k1 = rocketEquation(d_{xyz})$
  $k2 = rocketEquation(d_{xyz} + k_1 \cdot \frac{dt}{2})$
  $k3 = rocketEquation(d_{xyz} + k_2 \cdot \frac{dt}{2})$
  $k4 = rocketEquation(d_{xyz} + k_3 \cdot dt)$

  $d_{xyz+1} = d_{xyz} + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \cdot dt$
```

2.3 Animation Implementation

The following section describes how the model is simulated on the computer, both using MATLAB and by simulating in real time using JavaScript.

2.3.1 MATLAB

To animate the rocket trajectory in MATLAB, a nine grid subplot window was used. The calculations themselves were not time dependent and were therefore calculated as quickly as possible. To represent an actual trajectory the plots were instead animated after the calculations were made using *for loops* to plot each discrete data point, and in turn mimic the change over time.

2.3.2 JavaScript

To animate the rocket in JavaScript a third party library called Three.js was used. The library uses WebGL in order to run 3D graphics in the browser [1]. The library was used to run the simulation using 3D models and in a 3D environment. The simulation runs in real-time, meaning one second in the simulation reflects one second in real life. This is in contrast with the MATLAB simulation which runs the simulations as fast as the computer can calculate. The simulation runs in real-time by calling the RK4-function for each frame and using the time it takes for each frame to run as the step size.

Beyond the necessary components acting on the rocket that were made in Three.js, the 3D model of the landing station that is animated in Three.js was made in *Blender*. The object was exported as a glb-file in order to showcase it in the same way as the rocket when using the web browser.

Pseudocode 2 JavaScript: Simulation Loop

```
  while $true$ do
      updateRocketPosition($dt$)
      render($scene$, $camera$)
```

Chapter 3

Result

This chapter showcases the result found by the MATLAB implementation regarding numerical solvers aswell as the JavaScript version and its graphical implementation. The values used in the simulation can be found in the Appendix, A.1

3.1 MATLAB and Numerical Solver

The MATLAB testing and proof of concept solved the differential equations using MATLAB ODE45 solver. The final implementation in JavaScript used RK4 as its solver. Since the equations are rather elementary, only a small difference was found between the two methods. Figure 3.1 shows the simulation running for the first 100 seconds with ODE45. Figure 3.2 shows the same simulation with RK4. Both simulations ran for 100 seconds and the angles were set to zero.

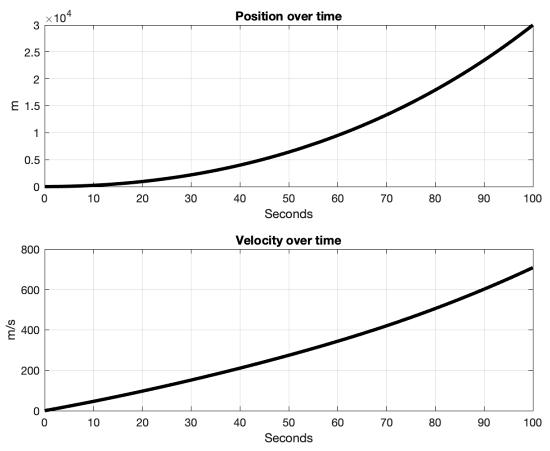

Figure 3.1: ODE45

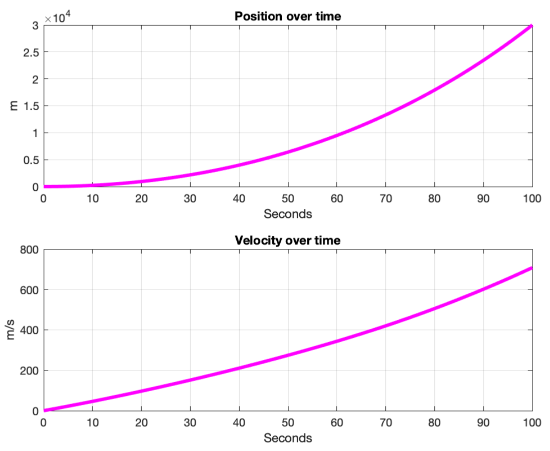

Figure 3.2: RK4

Figure 3.3 shows the quantization error between RK4 and ODE45, that is the difference between the results of the simulations above. The error is calculated on the z-axis taking position and velocity into account. The error was plotted as the absolute value on a logarithmic scale proving that our RK4 implementation was sufficient enough compared too the proof of concept completed with ODE45.

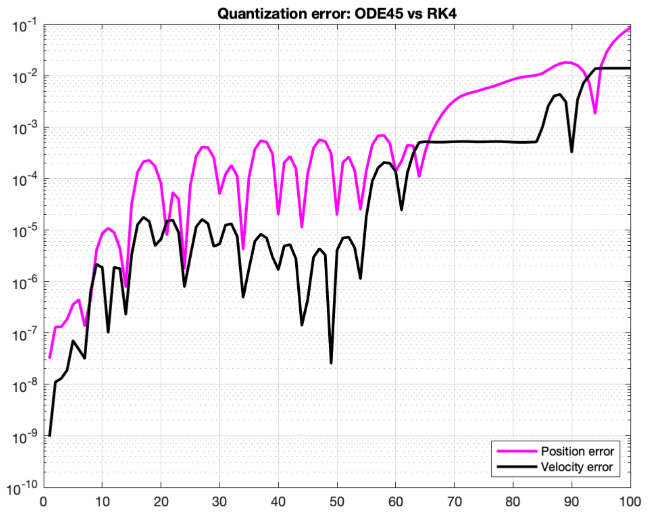

Figure 3.3: Quantization error: ODE45 vs RK4

3.2 JavaScript

Figure 3.4 shows the view of the final graphical representation of the JavaScript implementation. The window is a website running the simulation live. The view offers the user to change the angles theta θ and phi φ interactively during the simulation. The same applies to the thrust. The user interface also displays the current velocity asewll as the altitude and the fuel left. On the right hand side the user can pause, start or restart the simulation.

In the simulation, it looks like the rocket in in space. However the calculations and simulation is constructed in order for the rocket to start at the surface of the earth. The group never imported a model of the earth, thus giving the impression that the rocket takes off in space.

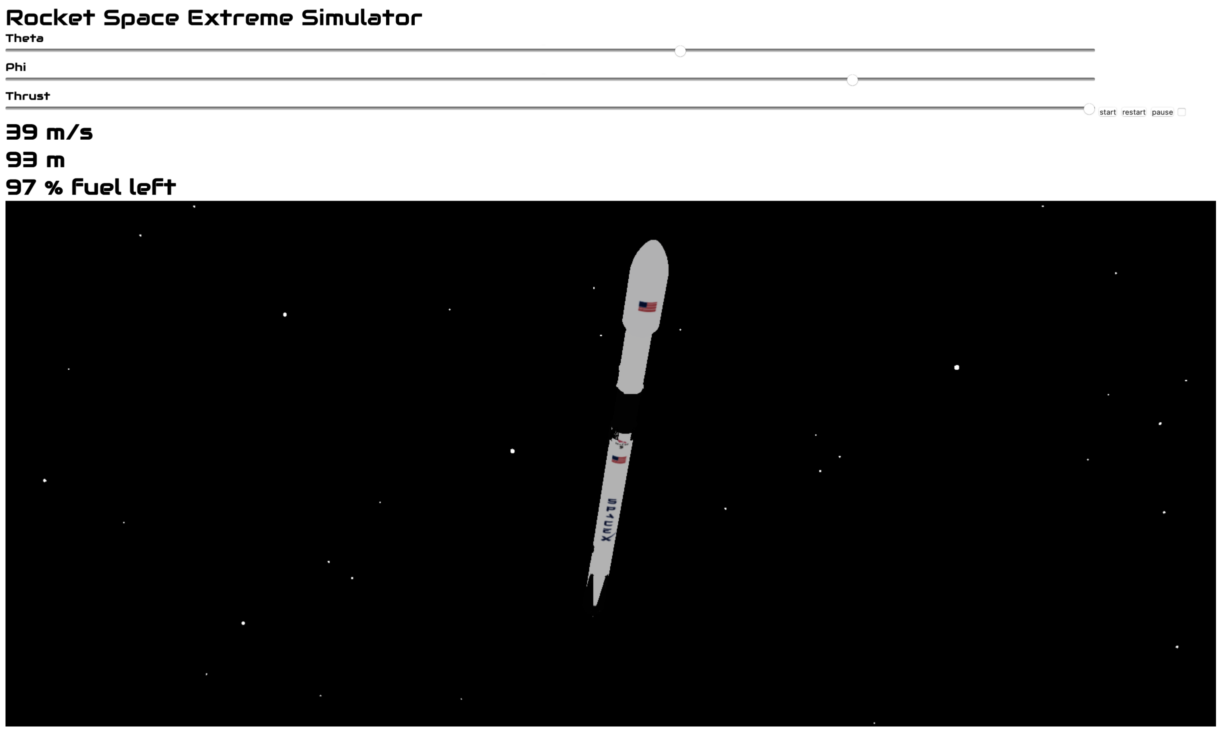

Figure 3.4: The final rocket simulation

Chapter 4

Summary

This chapter discusses the results found in both the MATLAB and JavaScript implementation.

4.1 Discussion

The simulation implemented in MATLAB gave concise and effective results, but was obviously lacking in the graphical aspect. Plots and animations were used to get a graphical representation. The time step is however not represented by real time, thus did not give the simulation sought after. By using a predetermined solver during the MATLAB phase it facilitated the debugging of the remaining parts. This allowed the group to focus on the model itself and not the numerical solver.

With the proof of concept done in MATLAB, the implementation could be moved to JavaScript. The JavaScript implementation uses a a self-developed RK4 instead of the ODE45. Since ODE45 uses a combination of RK4 and RK5, the self-developed RK4 gave similar results as ODE45 in MATLAB. This can also be seen in the quantization error between the two methods, which is near negligible. It should be noted that the lack of difference probably also depends on the rather simple differential equations and that the tests were done only with shorter time spans. It is highly likely that ODE45 outperforms RK4 when it comes to more advanced differential equations running for a longer amount of time.

In the JavaScript implementation the simulation loop takes real time into consideration, meaning one second of simulation would equal one second in the real world. Besides the time aspect and individual ODE solvers, the physical implementation in JavaScript and in MATLAB are identical. The most challenging part of the implementation was to successfully make the simulation run in real time. Since the concept was not tested in MATLAB it had to be developed directly in JavaScript from scratch. The real time rendering was implemented by setting the delta time, time it takes to run each frame, as the step size in the RK4 calculations.

4.2 Conclusion

In conclusion the rocket performed as sought taking air, mass, fuel and gravity into consideration. The goal to simulate the rocket in a real time loop with 3D graphics could be constructed by using JavaScript. The implementation was quite challenging and frankly impossible to do in MATLAB.

Bibliography

[1] Three.js, Threejs Documentation, fetched: 2023-02-23
threejs.org

[2] Ideal rocket equation, Equation Documentation, fetched: 2023-02-23
Nasa

[3] Air pressure, Equation Documentation, fetched: 2023-02-23
Nasa

[4] Gravity, Equation Documentation, fetched: 2023-03-02
The Grainger College of Engineering

[5] Rocket Aerodynamics, Equation Documentation, fetched: 2023-03-02
Nasa

Appendix A

Functions, variables and equations

In this appendix the constants used in the project can be found. The values in the table reflects the real values of Spacex's Falcon 9 rocket.

A.1 Constants

Mass flow rate, M	2100 kg/s
Velocity of Exhaust, V_e	3000 m/s
Exhaust Pressure, P_e	0.7 Pa
Exit Area, A_e	0.7 m
Drag Coefficient, C_d	0.6
Initial mass, m_0	395,700 kg
Burn rate, b_r	1451.496 kg/s
Gravity at sea level, G	9.82 m/s^2
Earth's mean radius, R	6 371 km
First stage rocket mass, m_{R1}	25 600 kg
Second stage rocket mass, m_{R2}	3900 kg
Payload mass, m_P	22 800 kg
Initial fuel mass, m_0	395 700 kg
Burn rate, b_r	1451.496 kg/s