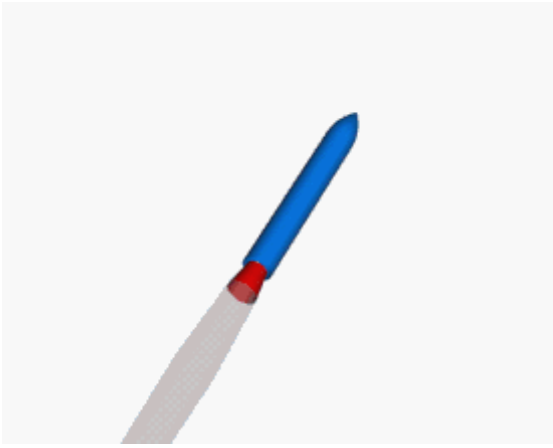
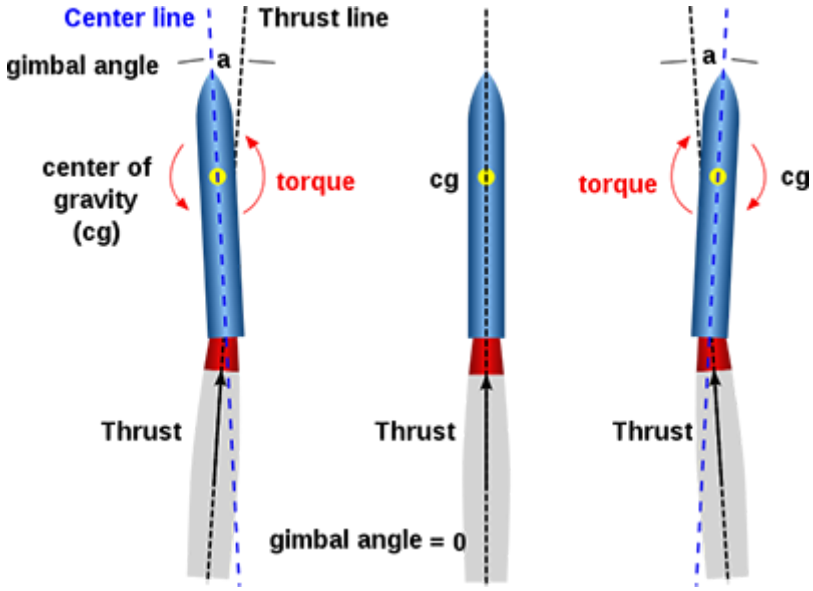
Code - 08-05-2018

Project Overview

Thrust vectoring is the ability of an aircraft, rocket, or other vehicle to manipulate the direction of the thrust from its motor in order to control the attitude (angle) of the vehicle. It is used in rocketry it is the primary means of attitude control, as aerodynamic control surfaces are ineffective outside the atmosphere.



In a gimbaled thrust system, the exhaust nozzle of the rocket can be swiveled from side to side. As the nozzle is moved, the direction of the thrust is changed relative to the center of gravity of the rocket.



I started designing and building a model rocket with a gimbaled thrust system. Its controller should allow it to be stable and follow a pre-determined trajectory without the help of fins, reducing its aerodynamic drag.

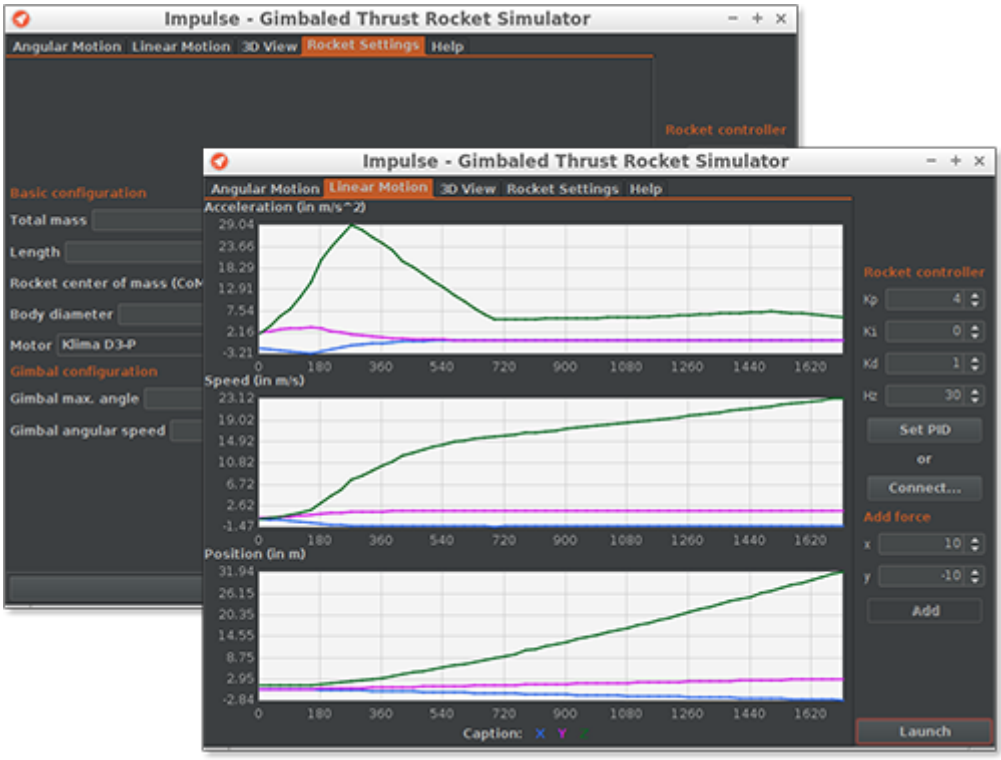
Because of the complexity of this project, it will be divided into two articles:

- Part 1: Impulse - the rocket physics simulator

- Part 2: design, build & tests of the rocket [*Work in progress*]

The software

A rocket is a complex system and the impact of every variable must be known prior to the design process. The usage of a simulation software that computes the kinematics of the system solves this issue by allowing us to visualize these impacts, and tweak the variables to optimize the flight performances.



Impulse is a open-source and cross-platform simulator for gimbaled thrust rockets:

- It accurately simulates the physics and outputs real-time data;
- The simulated rocket is **fully configurable**;
- To visualize the kinetics of the rockets, there are several graphs (for linear and angular acceleration, speed and position) and a 3D view;
- A rocket controller board can be connected through a **serial link**.

The last point is a very useful feature of Impulse: it means that it is capable of testing the control algorithm on the rocket hardware.

Download and usage

To launch Impulse, execute the jar file. As an alternative, in can be executed from the command line with `java -jar ImpulseRocketSimulator_v0.1.jar`. Java Runtime Environnement 8 must be installed.

DOWNLOAD LAST RELEASE (GITHUB) (HTTPS://GITHUB.COM/CGRASSIN/IMPULSE/RELEASES/LATES

The **help tab** in the software has all the required instructions in order to get started.

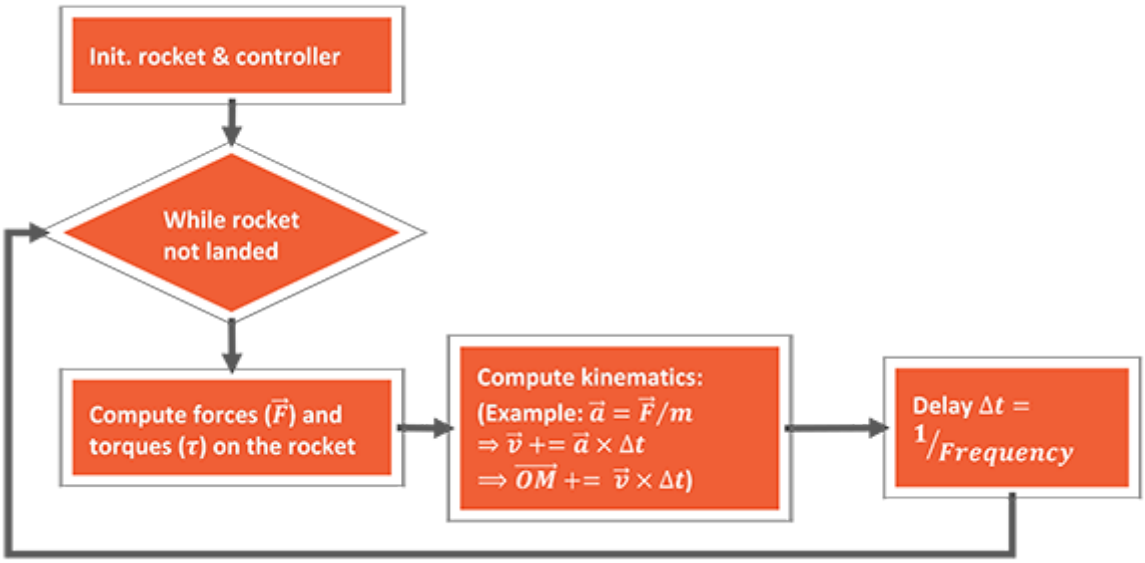
Technical Details

In this section, I will be doing a short technical overview of Impulse.

Physics engine

Impulse's engine is based on a physics model of the rocket I designed. It takes into account as many variables as possible to estimate the position and attitude of the rocket at any time: mass, repartition of mass, gimbal angle, motor thrust, etc.

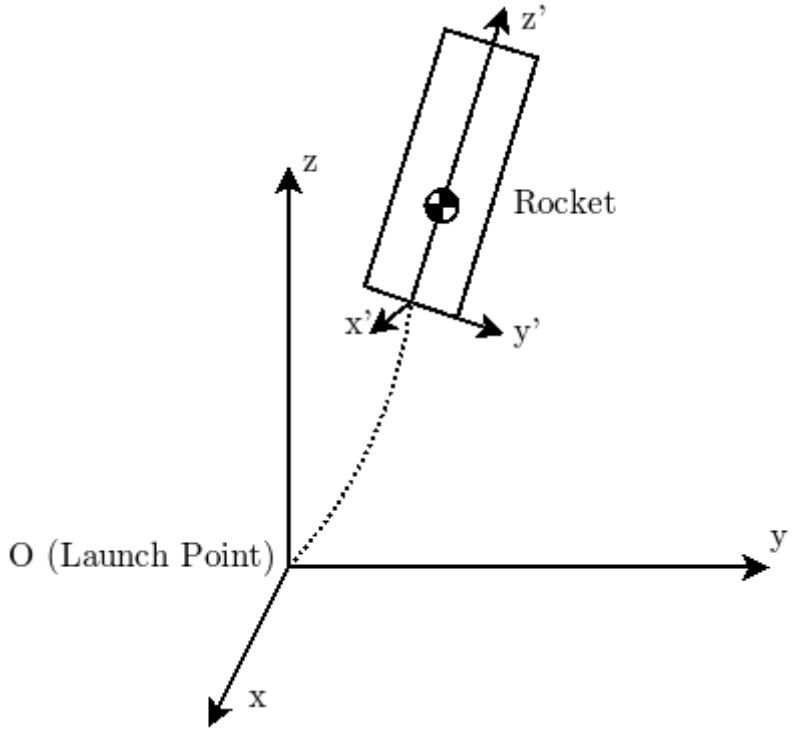
This basic flow diagram represents the way the discrete computation can be implemented in computer code.



As the model advances to get more accurate, the physics engine is intended to evolve.

Frame of reference

The frame of reference is Galilean, with an (x,y,z) orthogonal coordinate centered on the launch point. To compute rotational motion, another coordinate system (x',y',z') centered on the thruster of the rocket is defined (see figure below).



The angle of the thruster relative to the rocket is (Gx, Gy, 0). We need to compute the projection of this vector in the coordinate system of the rocket. Geometrically:

$$\vec{T}_{x'y'z'} \left(\frac{T \tan G_y}{u(G_x,G_y)} \quad \frac{T \tan G_x}{u(G_x,G_y)} \quad \frac{T}{u(G_x,G_y)} \right) \text{ with } u(a,b) = \sqrt{1 + \tan^2 a + \tan^2 b}$$

Similarly, the thrust along the long axis (z') of the rocket in the global frame of reference is:

$$\vec{T}_{xyz} \left(\frac{T \cdot tan(\theta_y)}{u(G_x,G_y) \cdot u(\theta_x,\theta_y)} \quad \frac{T \cdot tan(\theta_x)}{u(G_x,G_y) \cdot u(\theta_x,\theta_y)} \quad \frac{T}{u(G_x,G_y) \cdot u(\theta_x,\theta_y)} \right)$$

Dynamics of Translation Motion

For translation motion, we can simplify the problem by only considering the center of gravity of the rocket in our coordinate system (x,y,z). According to Newton's third law of dynamics:

$$\begin{aligned} \sum \vec{F} &= m\vec{a} \\ \vec{a} &= d\vec{v}/dt = d^2\vec{r}/dt^2 \\ \vec{v} &= d\vec{r}/dt \end{aligned}$$

We consider 3 forces acting of the rocket:

- Weight
- Thrust
- Aerodynamic drag (currently ignored)

Hence:

$$\sum \vec{F} = \vec{T} + \vec{P}$$

With:

$$\begin{matrix} \vec{T} \begin{pmatrix} \frac{T \cdot tan(\theta_y)}{u(G_x,G_y) \cdot u(\theta_x,\theta_y)} & \frac{T \cdot tan(\theta_x)}{u(G_x,G_y) \cdot u(\theta_x,\theta_y)} & \frac{T}{u(G_x,G_y) \cdot u(\theta_x,\theta_y)} \end{pmatrix} \\ \vec{P} \begin{pmatrix} 0 & 0 & -m \cdot g \end{pmatrix} \end{matrix}$$

To be used on a computer, these equations must be discretized. This is simply a question of converting integrals to sums. Hence, the system used by the software is:

$$\begin{aligned} \overrightarrow{a_n} &= \frac{1}{m} \sum \vec{F} \\ \overrightarrow{v_n} &= \overrightarrow{v_{n-1}} + \overrightarrow{a_n} \cdot \Delta t \\ \overrightarrow{r_n} &= \overrightarrow{r_{n-1}} + \overrightarrow{v_n} \cdot \Delta t \end{aligned}$$

Dynamics of Rotational Motion

The equations for rotation are very similar to those for translation:

$$\begin{aligned} \sum \vec{\tau} &= I \cdot \vec{\alpha} \\ \vec{\alpha}\ [rad \cdot s^{-2}] &= d\vec{\omega}/dt = d^2\vec{\theta}/dt^2 \\ \vec{\omega}\ [rad \cdot s^{-1}] &= d\vec{\theta}/dt \end{aligned}$$

The rotation around the z axis is ignored. There are two torques acting on the x/y axes of the rocket:

- Thrust
- Aerodynamic drag (currently ignored)

Hence:

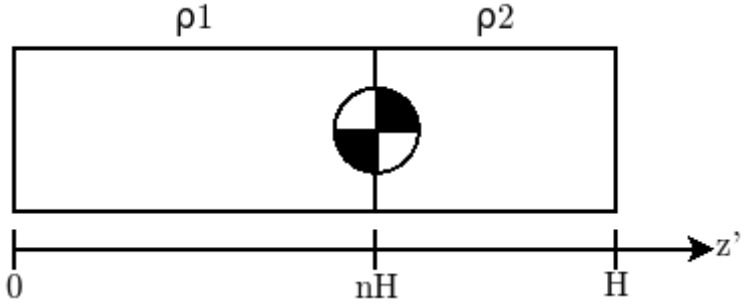
$$\sum \vec{\tau} = \begin{pmatrix} \frac{T \tan G_y}{u(G_x,G_y)} & \frac{T \tan G_x}{u(G_x,G_y)} & 0 \end{pmatrix}$$

The I factor is the 3-dimensional moment of inertia, the quantity equivalent to mass for angular acceleration. It depends on the geometry of the rocket. Its mathematical expression is:

$$I = \iiint d(x, \Delta)^2 \, \rho \, \mathrm{d}V$$

We assume that the rocket is axially symmetrical, of height H, total mass m, and that the center of gravity (CoG) is situated along the middle, at a height n×H where n is a coefficient (0<n<1). This allows us to transfer the problem to a single dimension (z').

This is equivalent to the following situation, where ρ1 and ρ2 represent the equivalent density aft and fore of the CoG.



By definition:

$$\begin{aligned} m = \int \rho dx &= \int_0^{Hn} \rho_1 dx + \int_{Hn}^{H} \rho_2 dx \\ \implies m &= nH\rho_1 + H\rho_2 - nH\rho_2 \end{aligned}$$

And, from the position of the CoG, we have:

$$\begin{aligned} \int_0^{Hn} \rho_1 dx &= \int_{Hn}^{H} \rho_2 dx \\ \implies \rho_1 &= \frac{1-n}{n}\rho_2 \end{aligned}$$

Hence:

$$
\begin{aligned}
\frac{m}{H} &= n\rho_1 + \frac{1-n}{n}\rho_1 - n\frac{1-n}{n}\rho_1 \\
\implies \frac{m}{H} &= \rho_1(n + \frac{(1-n)^2}{n}) \\
\implies \rho_1 &= \frac{m}{H}(n + \frac{(1-n)^2}{n})^{-1} \\
\text{and } \rho_2 &= \frac{m}{H}(\frac{1-n}{n})(n + \frac{(1-n)^2}{n})^{-1}
\end{aligned}
$$

From this point, we can compute I:

$$
\begin{aligned}
I_x = I_y &= \int_0^H \rho x^2 dx \\
&= \rho_1 \int_0^{nH} x^2 dx + \rho_2 \int_{nH}^H x^2 dx \\
&= \rho_1 [\frac{x^3}{3}]_0^{nH} + \rho_2 [\frac{x^3}{3}]_{nH}^H \\
&\dots \\
I_x = I_y &= \frac{m \cdot H^2}{3} \left(n^3 + \frac{(1-n)^4}{n}\right) \left(n + \frac{(1-n)^2}{n}\right)^{-1}
\end{aligned}
$$

Finally, similarly to translation, these equations must be discretized:

$$
\begin{aligned}
\overrightarrow{\alpha_n} &= \frac{1}{I} \sum \vec{\tau} \\
\overrightarrow{\omega_n} &= \overrightarrow{\omega_{n-1}} + \overrightarrow{\alpha_n} \cdot \Delta t \\
\overrightarrow{\theta_n} &= \overrightarrow{\theta_{n-1}} + \overrightarrow{\omega_n} \cdot \Delta t
\end{aligned}
$$

Code

Impulse is developped in Java, making it both efficient and cross-platform. All of its code is released under MIT license.

GO TO REPOSITORY (HTTPS://GITHUB.COM/CGRASSIN/IMPULSE)

The source code is very modular, with extensive usage of the object-oriented design principles. Therefore, it is very expendable: more motor types (hybrid, liquid fuel), multi-staging, etc. Every class is fully commented using standard Javadoc format.

Special thanks to Alexey Sokolov (JSSC), Ralf Sternberg (minimal-json) and Konstantin Bulenkov (DarculaLaf) for their great open-source libraries!

Limitations

There are still limitations to this simulator:

- It does not simulate aerodynamic drag. I advise using the great OpenRocket software (http://openrocket.info/) for accurate, open-source simulation of those forces.
- Currently, one degree of freedom is completely absent: the rotation around the vertical axis of the rocket. Although it is not required in my physics model, it might be added in the future.
- It makes several reasonable assumptions:
  - The rocket's mass repartition is radially symmetrical;
  - The rocket's mass doesn't change throughout the flight (yet).

In the future, I plan to:

- Priority: implement simulation of aerodynamic forces. This is quite a bit harder than in most other simulators because of the assumptions I can't make, due to the absence of fins!
- Import a complete motor database to provide the most often used thrust curves.
- Create an Arduino library to make the usage of Impulse with Arduino much easier.

Author: Charles Grassin (about)

What is on your mind?

Enter your comment...

Nickname

SUBMIT

#1 Boris

Bonjour
Je viens juste de découvrir ton site, je travaille sur un projet similaire depuis plusieurs années.
http://rocket.payload.free.fr/
https://www.youtube.com/feed/my_videos
J'ai les memes passions que toi et j'ai également fait une partie de mes etudes à l'etranger.
J'ai contribué au developement d'OpenRocket et suis dans l'association Rocketry France
Est ce que ça t'interesserai de collaborer?
J'habites à Suresnes en région parisienne.
Amicalement
Boris du Reau

on February 27 2020, 8:17

← BACK TO PROJECTS (PROJECTLIST)

Related articles

- Hairpin filter design (project-Hairpin+filter+design)
- Simply Appear (project-Simply+Appear)
- Reaction Wheel Attitude Control (project-Reaction+Wheel+Attitude+Control)