Coordinate System Documentation

6 Degrees-of-Freedom Trajectory Simulator

Cambridge University Spaceflight

Contents

[1 Coordinate systems 2](#_Toc63779991)

[1.1 Inertial 2](#_Toc63779992)

[1.2 Launch Site 3](#_Toc63779993)

[1.3 Body 4](#_Toc63779994)

[2 Step by step of the simulation process **(outdated)** 5](#_Toc63779995)

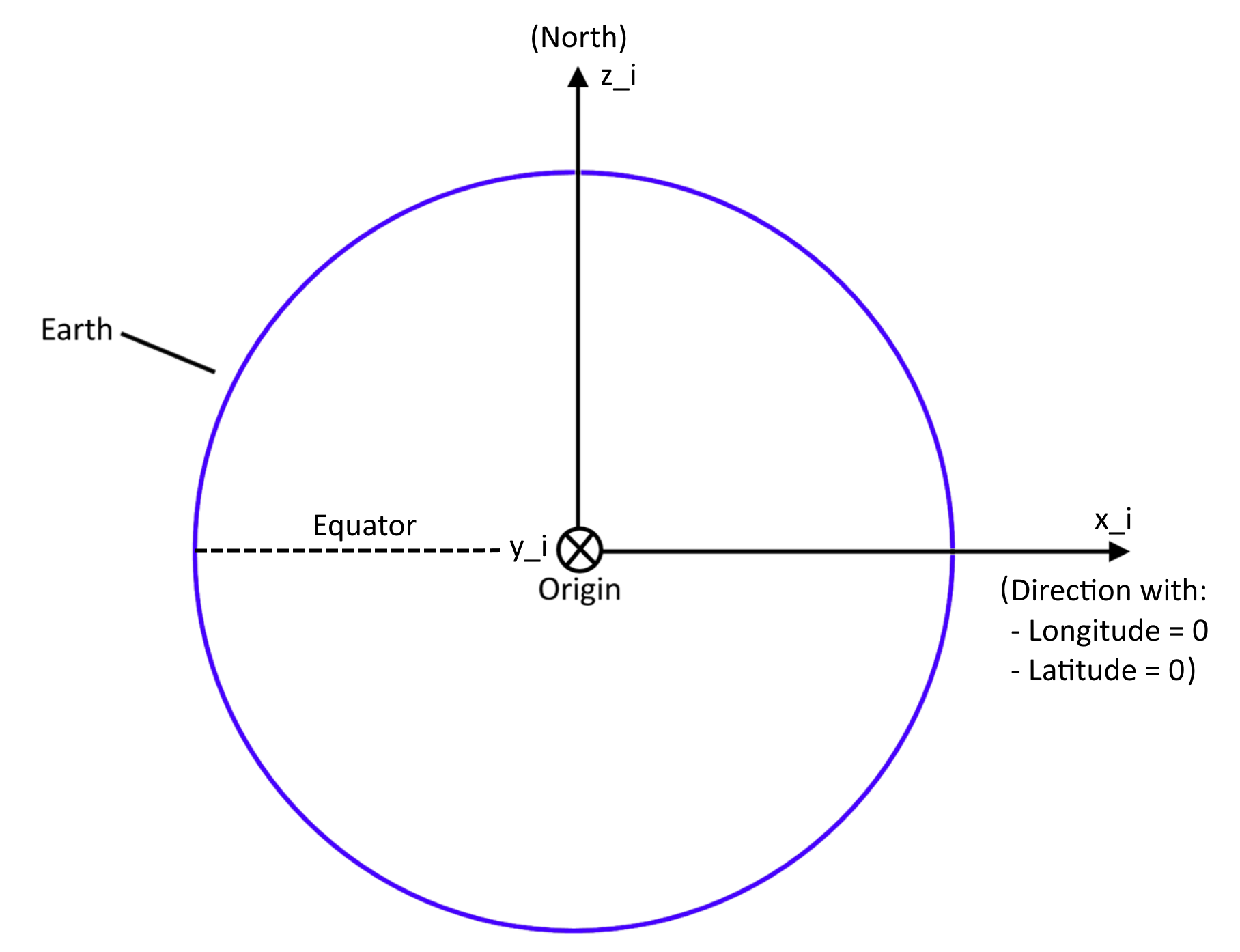
# Coordinate systems

There are three coordinate systems used in the code, all shown below.

## Inertial

The inertial coordinate system used is an Earth-centred inertial (ECI) frame.

Variables that are in inertial coordinates are denoted by the subscript ‘i’ in the code. E.g. position\_i, velocity\_i.



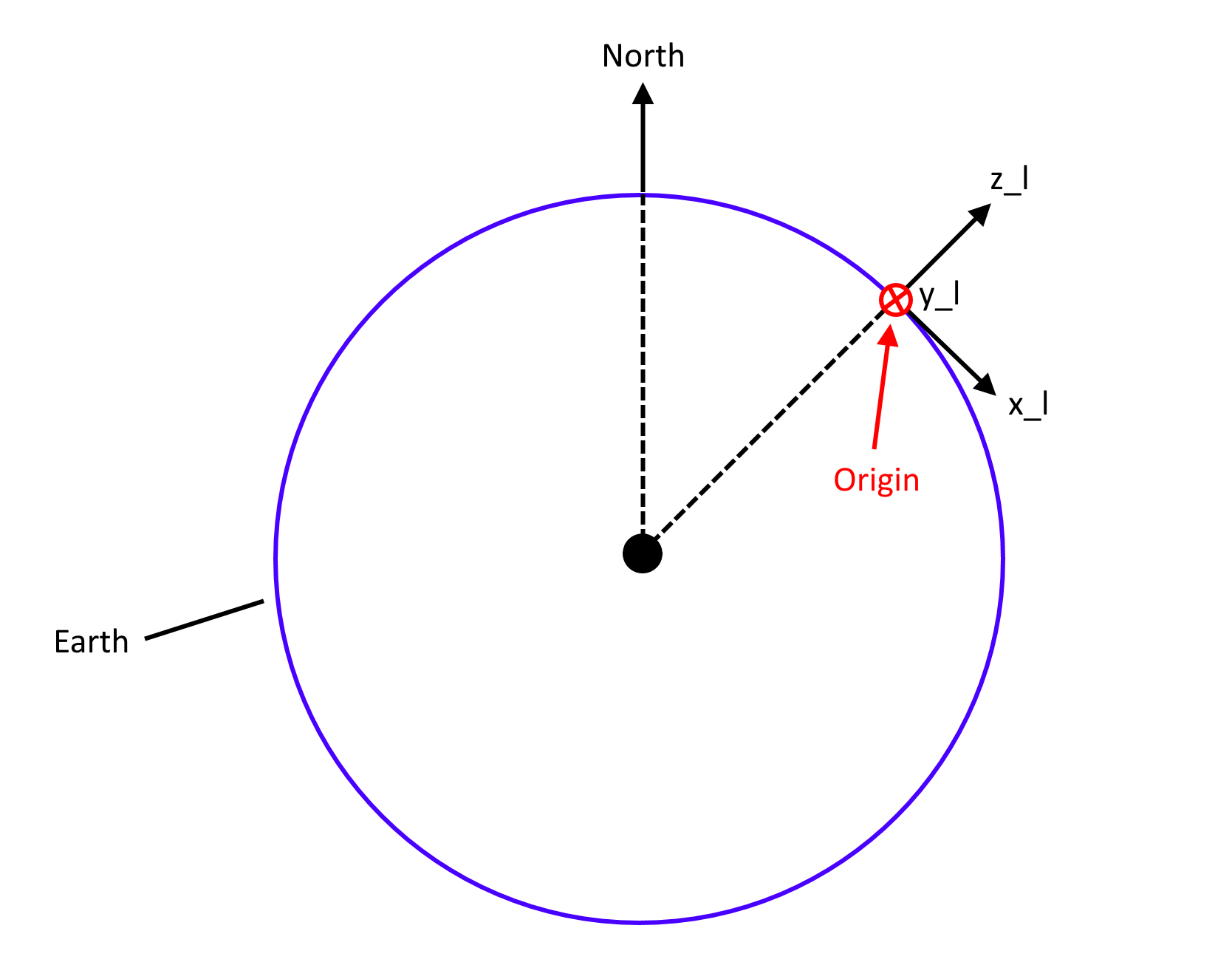
The origin is at the centre of the Earth.

* X points in the direction of 0 longitude, 0 latitude, at the start of the simulation (t=0).
* Z points North, from the centre of the Earth.
* Y is set by X and Z, given that it’s a right-hand coordinate system.

## Launch Site

This is a coordinate system that follows the launch site. Note that it is not inertial – it rotates with the Earth. Velocities in this reference frame are measured relative to the launch site – i.e. relative to the Earth’s surface.

Variables that are in inertial coordinates are denoted by the subscript ‘l’ in the code. E.g. position\_l, velocity\_l.



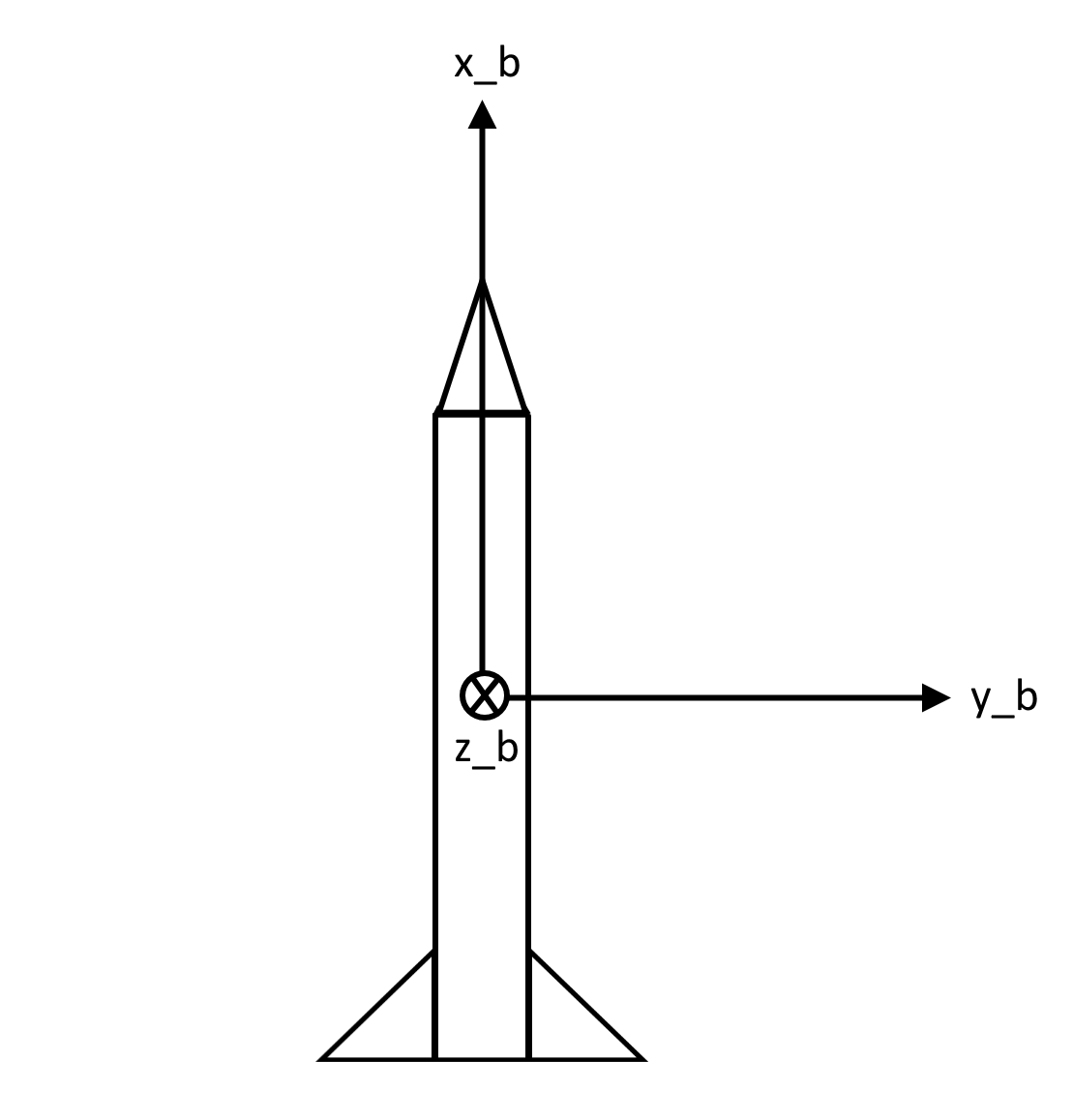
The origin is set at the same latitude and longitude as the launch site, but **at an altitude of zero**.

* Z is normal to the surface of the Earth, pointing away from the centre of the Earth.
* X is pointing southwards, tangential to the surface of the Earth.
* Y is set by X and Z since it’s a right-hand coordinate system.

## Body

The body coordinate system is fixed to the rocket and rotates with the rocket.

Variables that are in body coordinates are denoted by the subscript ‘b’ in the code. E.g. position\_b, velocity\_b.



There is no strict ‘origin’ for the body coordinate system. However, it is often necessary to state the position of various objects along the rocket, for example the position of a fuel tank, or the centre of pressure. For this, we use the convention that all positions along the rocket are given as **distances from the nosecone tip.**

Coordinate directions are specified as follows:

* X points “forwards”, i.e. towards the nose.
* Y is chosen so that it aligns with the launch site coordinate system, at the start of the simulation (i.e. when t=0).
* Z is calculated using the cross product of X and Y.

# Step by step of the simulation process **(outdated)**

User must create the Rocket object. This first requires the creation of the following ‘sub’-objects:

* MassModel – to model the moments of inertia, mass and dimensions.
* Motor – to model the propulsion system for the rocket.
* LaunchSite – to specify the launch site’s parameters (longitude, latitude, launch rail parameters).
* aerodynamic\_coefficients – currently the only option for this is to create a RasAeroData object. This specifies the information used for aerodynamic force and moment calculations.
* Parachute – Specifies drogue chute and main chute parameters.

self.accelerations():

1. Retrieve the forces acting on the rocket:
   1. self.thrust() – returns thrust as a vector\_b
   2. self.aero\_forces() – returns aero forces as a vector\_b, and the position where they act (the centre of pressure).
   3. self.gravity() – returns the gravity force, as a vector\_i
2. Calculate moments about the centre of gravity
3. Calculate linear acceleration in the inertial frame (using , and angular acceleration in the body frame (using Euler’s equations), then return them:
   1. return np.stack([lin\_acc, wdot\_b])

Get the rate of change of the state variables. These can then be used to integrate each state parameter.

Getting the rate of change of the orientation is slightly more complicated. We can define the directions of the body axes in the inertial frame as xb\_i, yb\_i and zb\_i. The rate of change of these vectors is then purely due to the rotation of them (), and is given by:

Etc. for yb\_i and zb\_i. Note that the ‘’ symbol represents the vector cross product here. Once we integrate xb\_i, yb\_i and zb\_i we can calculate the body-to-inertial rotation matrix as:

simulation\_output = Rocket.run()

Programme uses the SciPy DOP853 integrator, , and does each step with integrator.step(). The rocket’s state at each step can be defined entirely with the following data:

Inertial position (pos\_i), inertial velocity (vel\_i), angular velocity in body coordinates (w\_b), orientation in the form of a body-to-inertial rotation matrix (b2imat)

Calculate accelerations using self.accelerations().

For each execution of integrator.step()