S E S 2 0 2 6
Twenty-second International Scientific Conference
SPACE, ECOLOGY, SAFETY
20 – 24 October 2026, Stara Zagora, Bulgaria

NUMERICAL IMPLEMENTATION OF CONTROL LAW(S) FOR PROPULSIVE ROCKET LANDING

Konstantin Metodiev

Space Research and Technology Institute – Bulgarian Academy of Sciences
e-mail: komet@space.bas.bg

Keywords: *Rocket, Proportional-Integral-Derivative, Sliding Mode Control, GNU Octave*

Abstract: *This study introduces a computational approach toward implementing Proportional-Integral-Derivative (PID) and Sliding Mode Control (SMC) strategies to control rocket propulsive landings. GNU Octave environment is used to integrate linear and angular momentum conservation laws within body-fixed coordinates, complemented by quaternion-based attitude dynamics modeling. Real-time trajectory visualization is achieved through a simplex connection with FlightGear via UDP protocol. The obtained results highlight the ability of PID and SMC architectures to secure stable touchdowns under various conditions. The results also illustrate the resilience of these systems to dynamic disturbances that characterizes atmospheric re-entry situations from low Earth orbit.*

ЧИСЛЕНА РЕАЛИЗАЦИЯ НА РЕГУЛАТОР(И) ЗА ВЕРТИКАЛНО КАЦАНЕ НА РАКЕТА

Константин Методиев

Институт за космически изследвания и технологии – Българска академия на науките
e-mail: komet@space.bas.bg

Ключови думи: *Ракета, PID регулатор, SMC регулатор, GNU Octave*

Резюме: *Това изследване представя изчислителен подход към реализация на PID и SMC регулатори за управление на вертикално кацане на ракета. Използваната среда за интегриране на законите за запазване на количеството на движение и кинетичния момент в свързана координатна система е GNU Octave, като ориентацията на ракетата в пространството се определя чрез кватерниони. Визуализация на траекторията в реално време се постига чрез еднопосочна връзка с FlightGear по UDP протокол. Получените резултати подчертават способността на PID и SMC архитектурите да осигуряват стабилни кацания при различни условия. Резултатите също показват устойчивостта на тези системи към динамични смущения, типични при навлизане в атмосферата от околоземна орбита.*

Introduction

The importance of vertical landing technology lies in its potential to reduce the cost and increase the spaceflight efficiency. Rockets consume large amounts of propellant during launch, making single-use missions economically unsustainable in the long term.

Vertical landing addresses two key challenges. Firstly, it enables reusability by allowing rocket stages to return to Earth intact and be used for future missions. This greatly lowers launch costs compared to traditional disposable systems. Secondly, reusable launch vehicles might be able to support future modular deep-space transportation systems.

Current study employs two approaches towards simulating a rocket propulsive landing: Proportional-Integral-Derivative and Sliding Mode Controllers. The rocket flight is purely virtual. The flight dynamics states are computed in GNU Octave environment, [1], whilst the flight trajectory is visualized in FlightGear, [2] simulator. The project goal is to show that the maneuver under consideration is feasible enough to make it possible for coding in a high-level programming language.

Governing equations

To estimate the body linear rates, the equation of linear momentum conservation is integrated in body reference frame:

(1) $$\dot{\mathbf{v}} = \left(m\mathbf{I}_{3,3}^{-1}\right)\left[\mathbf{F} - \boldsymbol{\omega}\times\left(m\mathbf{v}\right)\right],$$

where $\mathbf{I}_{3,3}$ is 3x3 identity matrix, *m* = *const* stands for the body mass, $\mathbf{v} = [u, v, w]^T$ denotes a linear rates (velocities) vector, and **F** is a vector of forces externally applied to the system.

To estimate the body angular rates, the equation of angular momentum conservation is integrated in body reference frame:

(2) $$\dot{\boldsymbol{\omega}} = \mathbf{I}^{-1}\left[\mathbf{M} - \boldsymbol{\omega}\times\left(\mathbf{I}\boldsymbol{\omega}\right)\right],$$

where **I** stands for a 3x3 inertia tensor, $\boldsymbol{\omega} = [p, q, r]^T$ denotes an angular rates vector, and **M** is a vector of torques externally applied to the system. Mass, principal moments, and products of inertia are measured in a CAD environment in order to be used further in serial computations.

To estimate the body attitude and avoid gimbal lock (singularity) all together, it's preferable to propagate quaternion $\mathbf{q} = [q_w, q_x, q_y, q_z]^T$ by integrating the kinematic equation:

(3) $$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q}\otimes\begin{Vmatrix}0\\ \boldsymbol{\omega}\end{Vmatrix} = \frac{1}{2}\begin{Vmatrix}0 & -\boldsymbol{\omega}^T\\ \boldsymbol{\omega} & -\left[\boldsymbol{\omega}\right]\times_{3,3}\end{Vmatrix}\mathbf{q}.$$

The symbol []×3,3 denotes a skew-symmetric (cross-product) matrix, for example

(4) $$\left[\boldsymbol{\omega}\right]\times_{3,3} = \begin{Vmatrix}0 & -r & q\\ r & 0 & -p\\ -q & p & 0\end{Vmatrix}.$$

To estimate the body position, it is sufficient to transform the linear velocity from body to inertial reference frame and integrate afterwards, i.e.

(5) $$\dot{\mathbf{x}} = \mathbf{R}\mathbf{v},$$

where **R** stands for a transformation matrix from body to inertial reference frame, $\mathbf{v} = [u, v, w]^T$ is a velocity vector written in body reference frame.

The so-formed system of thirteen first order ordinary differential equations (ODE) from (1) to (3) and (5) is further integrated in GNU Octave environment by `ode45` function. The function implements Dormant-Prince method which is a member of the Runge-Kutta family of ODE solvers. It is an explicit, adaptive step, 4th and 5th order accurate method solely applicable to non-stiff systems. The state vector acquires definitive form $\mathbf{x} = [x, y, z, u, v, w, p, q, r, q_w, q_x, q_y, q_z]^T$.

To account for external forces **F** and torques **M** acting on a body, as well as its linear velocity components, coordinate transformations are necessary – specifically using a rotation matrix derived from a ZYX Euler angle sequence to toggle between body-fixed frame and inertial reference frame.

(6) $$\mathbf{R} = \begin{Vmatrix}C_\psi C_\theta & C_\psi S_\theta S_\varphi - S_\psi C_\varphi & C_\psi S_\theta C_\varphi + S_\psi S_\varphi\\ S_\psi C_\theta & S_\psi S_\theta S_\varphi + C_\psi C_\varphi & S_\psi S_\theta C_\varphi - C_\psi S_\varphi\\ -S_\theta & C_\theta S_\varphi & C_\theta C_\varphi\end{Vmatrix},$$

where Sx = sin(x) and Cx = cos(x). The transformation matrix **R** is orthogonal, hence $\mathbf{R}^{-1} = \mathbf{R}^T$.

Alternatively, the rotation matrix can be computed using quaternions. Given a unit vector representing the rotation axis and an associated angle of rotation θ, those define the quaternion components accordingly.

(7) $$q_w = \cos\left(\theta/2\right)\quad q_x = a_x\sin\left(\theta/2\right)\quad q_y = a_y\sin\left(\theta/2\right)\quad q_z = a_z\sin\left(\theta/2\right).$$

This formula is applied sequentially to the axis Cartesian components – such as [0, 0, az], [0, ay, 0], [ax, 0, 0] – along with their corresponding Euler angles [0, 0, ψ], [0, θ, 0], and [φ, 0, 0] following a rotation sequence (for example, ZYX) given in advance. This process yields three individual quaternions: q1, q2, and q3, which are then multiplied.

Quaternions are frequently split into a scalar term s and a vector term **v**

(8) $$q = \begin{bmatrix}s & \mathbf{v}\end{bmatrix}\quad s\in\mathbb{R}\quad \mathbf{v} = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}.$$

Using the notation above, the quaternion product is computed using the formula

(9) $$q_1q_2 = \left(s_1s_2 - \mathbf{v}_1\mathbf{v}_2\right) + \left(s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1\times\mathbf{v}_2\right).$$

Then, the transformation matrix (from body to inertial frame of reference) for any rotation sequence is derived from the product [qw,qx,qy,qz] = (q1q2)q3 as follows

$$\mathbf{R} = \begin{Vmatrix} 2\left(q_w^2 + q_x^2\right) - 1 & 2\left(q_x q_y - q_w q_z\right) & 2\left(q_x q_z + q_w q_y\right) \\ 2\left(q_x q_y + q_w q_z\right) & 2\left(q_w^2 + q_y^2\right) - 1 & 2\left(q_y q_z - q_w q_x\right) \\ 2\left(q_x q_z - q_w q_y\right) & 2\left(q_y q_z + q_w q_x\right) & 2\left(q_w^2 + q_z^2\right) - 1 \end{Vmatrix}. \quad (10)$$

A PID, or **Proportional-Integral-Derivative** controller is a type of feedback control loop that is extensively used in industrial automation. The PID controller maintains the output variable (process, PV) close to the target value (setpoint, SP) by minimizing the error e(t) between the two through continuous summation of proportional, integral, and derivative additives. The control signal is computed by renowned formula:

$$\mathbf{u}(t) = K_p \mathbf{e}(t) + K_i \int_0^t \mathbf{e}(\tau)\, d\tau + K_d \frac{d}{dt}\mathbf{e}(t), \quad (11)$$

where e(t) = SP – PV, Kp, Ki, Kd are proportional, integral, and differential gains respectively. Each additive could be interpreted as follows.

- The **proportional** gain responds to the current error. The output is directly proportional to the present difference between SP and PV (error = SP – PV). A high Kp gain reduces the immediate error but can cause overshoot or oscillation if too large.
- The **integral** gain eliminates steady-state errors by integrating the error over time. It acts as a "memory" of past errors, ensuring that even small persistent differences between setpoint and actual value eventually result in corrective action. However, excessive Ki gain can cause sluggish response or instability due to integral windup.
- The **derivative** gain predicts future error based on the rate at which the current error is changing. It helps dampen overshoot by opposing rapid changes in output before they become problematic. Kd gain reduces oscillations but may introduce noise sensitivity if not tuned properly.

PID control system is preferred since it can perform excellently with simple tuning by means of the Ziegler-Nichols technique or modern optimization approaches. In spite of this, tuning of PID controllers still poses an engineering challenge owing to system nonlinearities and time delays.

Two PID controllers are implemented in the current project: a position PID and an attitude PID. Implementing the position PID controller in time domain is straightforward. The attitude PID controller estimates a difference between the current orientation (represented as a quaternion) and an initial reference orientation – such as [1, 0, 0, 0], which signifies no rotation or upright alignment (a standard "rocket-stationary" state). The error quaternion is calculated by taking a dot product of the current quaternion (process value) and conjugate of the desired one (setpoint). When both quaternions are identical, their imaginary components vanish in this operation; otherwise, a control output is generated by the PID algorithm to correct the deviation.

A SMC, or **Sliding Mode Controller** is a nonlinear control approach wherein system dynamics is initially allowed to follow nonlinear trajectories determined in the phase portrait by open-loop behavior before being forced to slide onto an initially defined line, often called "state-space sliding surface." Upon reaching the surface, the system states are directed and kept there, thus guaranteeing robustness against parameter uncertainties and external disturbances.

The position and velocity setpoints in case of translational motion are **x** = **x**d and **v**d = **0**. Another commonly used sliding surface is

$$\mathbf{s} = \mathbf{e}_v + \mathbf{\Lambda}\mathbf{e}_x, \quad (12)$$

where new state variables **e**v = **v**i – **v**d and **e**x = **x**i – **x**d, **Λ** = diag(λx, λy, λz), λi > 0 is an adjustable parameter matrix. The velocity **v**i is transformed according to eq. (5) from body to inertial reference frame. If the setpoint **x**d is constant, then error vector **e**x is simply a coordinate system shift, therefore the sliding surface geometry remains unchanged.

If differentiate the surface equation (12), it gives

$$\dot{\mathbf{s}} = \dot{\mathbf{e}}_v + \mathbf{\Lambda}\dot{\mathbf{e}}_x, \quad (13)$$

where vector

$$\dot{\mathbf{e}}_v = \frac{\mathbf{F}_I}{m} + \mathbf{g} - \ddot{\mathbf{x}}_d. \quad (14)$$

This presentation is better for tracking a reference point instead of zero state. It is derived from linear momentum conservation equation written in inertial reference frame. Taking into account that

$$\dot{\mathbf{e}}_x = \dot{\mathbf{x}}_I - \dot{\mathbf{x}}_d = \mathbf{v}_I - \mathbf{v}_d = \mathbf{e}_v \quad (15)$$

and for a fixed setpoint

(16) $\ddot{\mathbf{x}}_d = \mathbf{0}$

then

(17) $\dot{\mathbf{s}} = \frac{\mathbf{F}_I}{m} + \mathbf{g} + \mathbf{\Lambda}\mathbf{e}_v$.

The reaching condition usually implies

(18) $\dot{\mathbf{s}} = -\mathbf{K}.sat(\mathbf{s})$,

where *sat* stands for saturation function, **K** = diag(Kx, Ky, Kz), Ki > 0 is an adjustable parameter matrix. After substituting in eq. (17)

(19) $\frac{\mathbf{F}_I}{m} + \mathbf{g} + \mathbf{\Lambda}\mathbf{e}_v = -\mathbf{K}.sat(\mathbf{s})$

and solve for **F**I, following formula is obtained to compute linear control law (translational motion)

(20) $\mathbf{F}_I = -m\mathbf{\Lambda}\mathbf{e}_v - m\mathbf{K}.sat(\mathbf{s}) - m\mathbf{g}$

which is an exact force law for the rocket to reach desired position. It must be transformed back to body reference frame

(21) $\mathbf{F}_b = \mathbf{R}^T\mathbf{F}_I$.

Angular control law (rotational motion) is computed likewise. Consider equation of angular momentum conservation written in body reference frame and rearranged with respect to d**ω**/dt:

(22) $\dot{\boldsymbol{\omega}} = \mathbf{I}^{-1}\left[\mathbf{M} - \boldsymbol{\omega}\times(\mathbf{I}\boldsymbol{\omega})\right]$.

Define a sliding surface

(23) $\mathbf{s} = \boldsymbol{\omega} + \mathbf{\Lambda}_{att}\mathbf{q}_v$,

where

(24) $\mathbf{q}_{err} = \mathbf{q}_0^* \otimes \mathbf{q}$

and

(25) $\mathbf{q}_v = \mathbf{q}_{err}(2:4)$

is the vector part of the error quaternion. The goal is to make system reach **s** = **0** and slide along it.

The SMC reaching law implies

(26) $\dot{\mathbf{s}} = -\mathbf{K}.sat(\mathbf{s})$,

where saturation is used to reduce chattering. After differentiating (23)

(27) $\dot{\mathbf{s}} = \dot{\boldsymbol{\omega}} + \mathbf{\Lambda}_{att}\dot{\mathbf{q}}_v$

and substituting for d**ω**/dt (22) and reaching law (26)

(28) $\dot{\mathbf{s}} = \mathbf{I}^{-1}\left[\mathbf{M} - \boldsymbol{\omega}\times(\mathbf{I}\boldsymbol{\omega})\right] + \mathbf{\Lambda}_{att}\dot{\mathbf{q}}_v = -\mathbf{K}_{att}.sat(\mathbf{s})$.

In last formula, **Λ**att = diag(λx, λy, λz), λi > 0, **K**att = diag(Kx, Ky, Kz), Ki > 0 are adjustable parameter matrices. Rearranging with regard to **M**, it yields

(29) $\mathbf{M} = -\mathbf{I}\mathbf{\Lambda}_{att}\dot{\mathbf{q}}_v - \mathbf{I}\mathbf{K}_{att}.sat(\mathbf{s}) + \boldsymbol{\omega}\times(\mathbf{I}\boldsymbol{\omega})$.

The formula above is an exact SMC torque law.

The error quaternion is defined as

(30) $\mathbf{q}_{err} = \begin{Vmatrix} q_0 \\ \mathbf{q}_v \end{Vmatrix}$.

In context of quaternion kinematics, the derivative of quaternion is given by

(31) $\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \otimes \begin{Vmatrix} 0 \\ \boldsymbol{\omega} \end{Vmatrix}$.

When expanded into its scalar and vector components, the vector part is expressed as:

(32) $\dot{\mathbf{q}}_v = \frac{1}{2}\left(q_0\boldsymbol{\omega} + \mathbf{q}_v \times \boldsymbol{\omega}\right)$.

Using the skew-symmetric matrix notation to represent the cross product, the formula becomes

(33) $\dot{\mathbf{q}}_v = \frac{1}{2}\left(q_0\mathbf{I}_{3,3} + [\mathbf{q}_v]\times\right)\boldsymbol{\omega}$,

where $\mathbf{I}_{3,3}$ stands for an identity matrix. Near desired attitude $\mathbf{q}$ = [1, 0, 0, 0], q0 ≈ 1 and $\mathbf{q}v \approx \mathbf{0}$, hence the approximation

$$\dot{\mathbf{q}}_v \approx \frac{1}{2}\boldsymbol{\omega}. \tag{34}$$

Substituting into the exact law (29):

$$\mathbf{M} \approx -\frac{1}{2}\mathbf{I}\boldsymbol{\Lambda}_{att}\boldsymbol{\omega} - \mathbf{I}\mathbf{K}_{att}.sat(\mathbf{s}) + \boldsymbol{\omega}\times(\mathbf{I}\boldsymbol{\omega}) \tag{35}$$

which is an approximate torque law for the rocket to stay upright.

The *sat* function taking part in (20) and (35) is employed so as to reduce control chattering as specified by the following formulae

$$sat(\mathbf{s}) = \begin{cases} \mathbf{s}/\varphi & if \quad \mathbf{s}/\varphi \le 1 \\ sign(\mathbf{s}) & otherwise \end{cases}, \tag{36}$$

where value of **s** surface is computed by either (12) or (23) depending on type of motion (linear, angular) the rocket undergoes.

A simplex UDP connection between GNU Octave and FlightGear is established via UDP protocol for visualization purposes. In order to do so, it is necessary to upload an XML file in directory `%FlightGearInstallDir%\fgdata_2024_1\Protocol` for the flight simulator to interpret incoming UDP data. The XML file is composed by chunks for timestamp, position, and attitude variables being transmitted to the simulator. The UDP sender implemented in GNU Octave must send these variables in same order precisely. Given XML file name `myRocket.xml`, localhost resolved to IP address 127.0.0.1, port 5500, update rate of 50 Hz, FlightGear must be started with following script:

```
--generic=socket,in,50,127.0.0.1,5500,udp,myRocket
--fdm=null
--altitude=1250
--httpd=8080
```

I/O functions used in current study to establish a serial UDP communication are implemented in `instrument-control` package, [3]. The rocket model folder might be copied from almost any aircraft already installed to the simulator. The CAD file with .ac extension must be replaced, though.

Results

Following figure illustrates the rocket CAD model in FlightGear and GNU Octave environment prior to commencing numerical experiment. In the figure, the rocket stance does not correspond to initial conditions for visualization purpose. In Octave console, it is necessary to type and execute the line `pkg load instrument-control` prior to commencing the simulation.

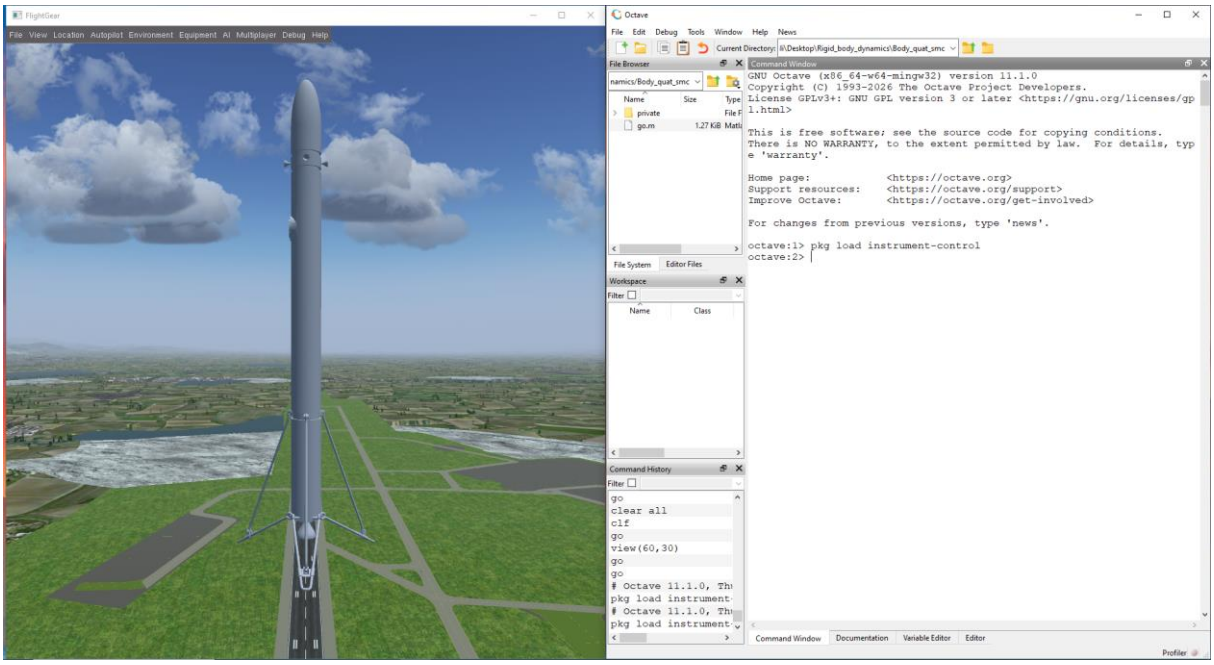


Fig. 1. A CAD model of the rocket in FlightGear

Following graphs illustrate the rocket trajectory during numerical simulation. The initial conditions used in numerical experiments with both controllers are pitch angle `-pi/2` rad (head down), height of 1250 m, and zero velocity. The setpoint is height 10 m, zero velocity, upright stance. At the simulation onset, two forces set the rocket in motion: the gravity and a constant aerodynamic drag. The trajectory is divided into two stages:

- stage 1–2: landing approach, during which the rocket is descending and taking an upright stance simultaneously
- stage 2–3: touchdown, which is self-explanatory

The initial conditions are clearly discernable, for example the blue axis is pointing downwards, initial height on Z axis and so on, so is a slight trajectory overshoot in PID control case.

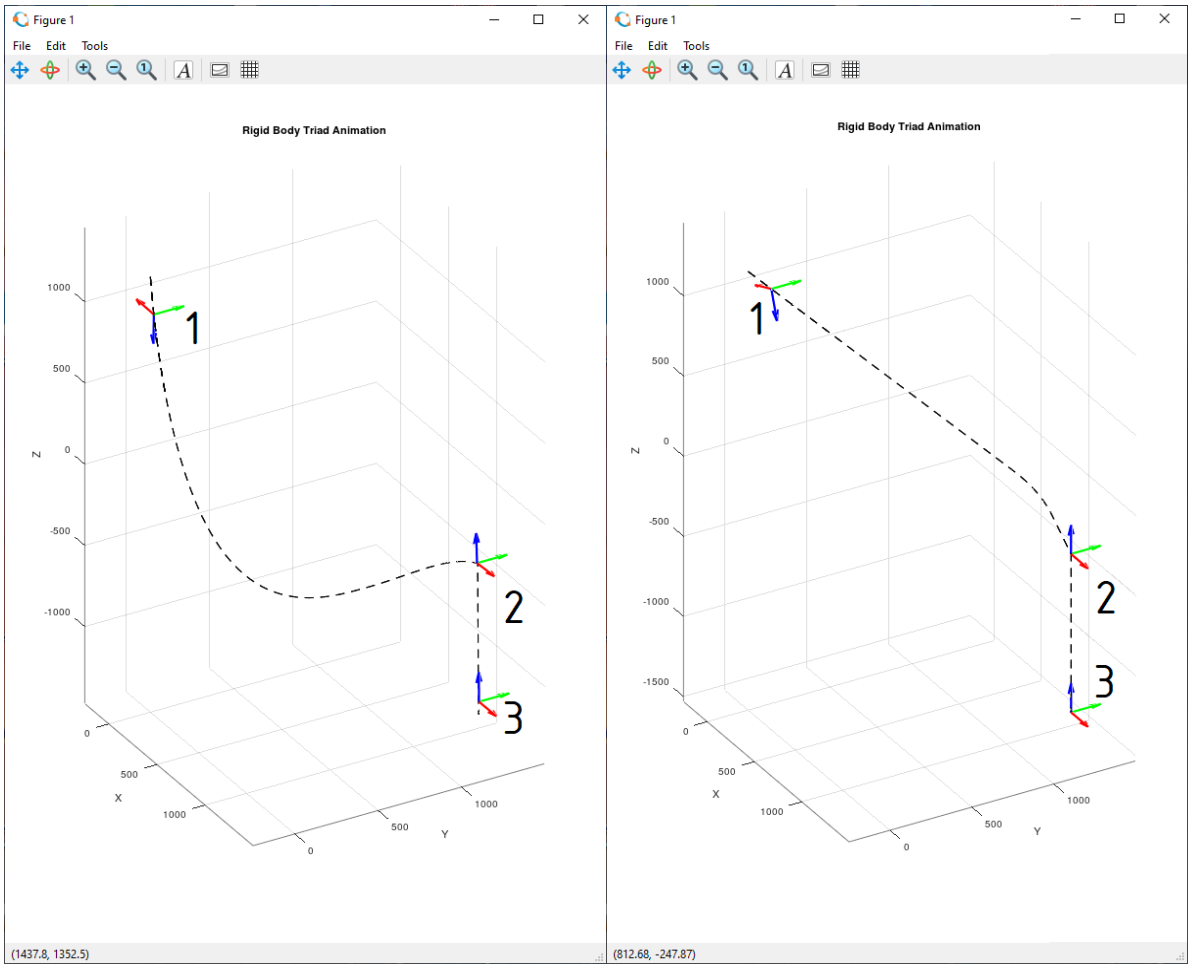


Fig. 2. Landing trajectory: PID controller (left) and SM Controller

Discussion

The current research presumes the rocket is fully actuated which is almost never the case. A hypothetical rocket with thrusters producing forces in x, y, z directions and a torque generator about roll, pitch, yaw angles could control all six degrees of freedom (DOF) independently. A simplified and achievable landing model however solely possesses thrust magnitude and two gimbal angles control, i.e. three control inputs acting upon a 6-DOF vehicle. Still, suggested research and developed source code demonstrate how to implement PID and SMC controllers in time domain, which is a good prerequisite to developing a feasible model.

The developed source code may be downloaded from link [4], as may a short demonstration video. Giving a quotation credit to the author is obligatory prior to using the code.

References:

1. https://octave.org/
2. https://www.flightgear.org/
3. https://octave.sourceforge.io/instrument-control/overview.html
4. https://github.com/samolet4e/Propulsive-rocket-landing