Dynamic Feedback Linearizability of Ingenuity

Antonio D'Orecchia
Sapienza, University of Rome
dorecchia.1403841@studenti.uniroma1.it

Abstract

This project aims for exploring a possible way to control a coaxial helicopter built by NASA and sent to Mars from 2021 to 2024 (Wikipedia, 2024), called Ingenuity, through a Dynamic Feedback Linearization (DFL) approach: the basic idea is to converge the dynamics of this Unmanned Aerial Vehicles (UAV) to the equations of a classic quadrotor (by means of some basic assumptions) and then implement on them the DFL method valid for a quadcopter, already described and validated by many sources in Robotics literature.

1 Introduction

A drone designed to be sent to Mars must face several challenges compared to Earth, such as different atmosphere and gravity, extreme temperature variations, and high level of radiation (Grip et al., 2018). Most UAVs research on rotary-wing aircrafts has focused on quadrotor helicopters, systems with four rotors rotating around four distinct parallel axes; in contrast to quadrotors, coaxial helicopters are aircrafts with two rotors rotating in opposite directions around the same axis. The main advantages in choosing a coaxial helicopter instead of a quadcopter for a space mission are (Seisan, 2012):
- Coaxial helicopters consume less power for the same payload, so being more efficient;
- Coaxial helicopters can be made much smaller, and this compactness is very suitable for transportation and packaging purposes, as well as being less sensitive to wind gusts (Hua, 2009);
But, on the other side:
- Coaxial helicopters are more difficult to control due to increased mechanical complexity (Chen and McKerrow, 2007);
- Coaxial helicopters have limited commercialization.
Also, the robot dynamics is very different in principle, but with a series of simplifications and considerations, it is possible to bring the coaxial helicopter equations back to the classic quadcopter equations, thus allowing to use the same methods available in literature for controlling the system, like for example the feedback linearization.
The report is organized as follows: section 2 breaks the dynamic of Ingenuity as found in literature with a different perspective, constrained to an inertial frame-only representation; In section 3, a physical comparison between helicopter and quadcopter is done, with the crucial assumption of doing maneuvers close to hover (Dzul et al., 2002) to connect the dynamics of the coaxial helicopter to the one of the quadcopter; section 4 focuses on the results of trajectory tracking simulations executed in MATLAB (Simulink) integrated with CoppeliaSim, by using a DFL control apporach; the final section recaps the purpose of the work and shows possible improvements and future investigations about the topic.

2 Ingenuity dynamics: different perspective

The dynamics of a coaxial helicopter is often presented in a hybrid form, i.e. mixing equations from the viewpoint of both the inertial and body frames. The translational motion is expressed as (Vendittelli, 2024):

$$\dot{p}^i = Rv^b \tag{1}$$

$$m\dot{v}^b = T^b + R^T(F_g^i + F_e^i) - \omega \times mv^b \tag{2}$$

Where:

$$p^i = \begin{bmatrix} x^i \\ y^i \\ z^i \end{bmatrix} \tag{3}$$

$$v^b = \begin{bmatrix} v_x^b \\ v_y^b \\ v_z^b \end{bmatrix} \tag{4}$$

$$R = \begin{bmatrix} cos(\psi)cos(\theta) & cos(\psi)sin(\theta)sin(\phi) - sin(\psi)cos(\phi) & cos(\psi)sin(\theta)cos(\phi) + sin(\psi)sin(\phi) \\ sin(\psi)cos(\theta) & sin(\psi)sin(\theta)sin(\phi) + cos(\psi)cos(\phi) & sin(\psi)sin(\theta)cos(\phi) - sin(\phi)cos(\psi) \\ -sin(\theta) & cos(\theta)sin(\phi) & cos(\theta)cos(\phi) \end{bmatrix} \tag{5}$$

$$T^b = G(\alpha, \beta, \gamma)|f|, \qquad G(\alpha, \beta, \gamma) = \begin{bmatrix} -tan(\alpha)cos(\gamma) \\ tan(\beta)cos(\gamma) \\ -cos(\gamma) \end{bmatrix} \tag{6}$$

$$F_g^i = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{7}$$

$$\omega = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{8}$$

F_e^i identifies the external forces (unknown) acting on the robot, $|f|$ represent the net thrust determined by the angular velocity of the rotors, while α, β and γ are the angles used to project the thrust direction towards the three Cartesian axis. The rotational motion is defined by:

$$\dot{R} = S(\omega)R \tag{9}$$

$$I\dot{\omega} = -\omega \times I\omega + \tau_{tot}^b + R^T\tau_e^i \tag{10}$$

Where:

$$S(\omega) = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{11}$$

$$I = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{12}$$

$$\tau_{tot}^b = \begin{bmatrix} \tau_x^b \\ \tau_y^b \\ \tau_z^b \end{bmatrix} \tag{13}$$

τ_e^i identifies the external torques (unknown) acting on the robot. In both cases, the actuation coupling is discharged. With this representation of the dynamics, and in order to put it on the same level of the dynamics of a quadrotor, there are some negative points:

1) Mixed variables between world and body frames;

2) Matrix R to represent orientation instead of angles;
3) Coriolis-Centrifugal terms due to the non-inertial frame representation.

In response to this, the intuition is then:

1) Considering the translational-rotational equations with respect to the inertial reference frame only;
2) Expliciting the Roll-Pitch-Yaw (or Euler) angles both as part of the rotational dynamics and of the state of the system;
3) Removing the Coriolis-Centrifugal terms (as a consequence of point 1).

The new representation becomes (Seisan, 2012):

$$\dot{p}^i = v^i \tag{14}$$

$$m\dot{v}^i = RT^b + F_g^i \tag{15}$$

$$\dot{\Theta} = Y^{-1}(\phi, \theta)\omega \tag{16}$$

$$I\dot{\omega} = \tau \tag{17}$$

Where:

$$v^i = \begin{bmatrix} v_x^i \\ v_y^i \\ v_z^i \end{bmatrix} \tag{18}$$

$$\dot{\Theta} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{19}$$

$$Y^{-1}(\phi, \theta) = \begin{bmatrix} 1 & sin(\phi)tan(\theta) & cos(\phi)tan(\theta) \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi)sec(\theta) & cos(\phi)sec(\theta) \end{bmatrix} \tag{20}$$

$$\tau = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \tag{21}$$

Assuming also the external forces to be negliglible ($F_e = 0, \tau_e = 0$). In fact, in quasi-stationary flight conditions and in the absence of wind, aerodynamic forces arising from relative vehicle/wind velocity can be neglected in front of gravity since their intensities are proportional to the square of this velocity (Hua et al., 2013). This physical approximation is the key to another result of the present work that is explained in section 3.

3 Approximation: from coaxial helicopter to quadcopter

The main differences between quadcopter and coaxial helicopter can be summarized in:
- Num. rotors: 4 vs 2;
- Quadcopter is more easily controllable, helicopter is more comfortable in terms of packaging;
- Quadcopter has a vertical thrust vector T, while in helicopter in general it is not entirely vertical, but in both the upper and lower rotors it has components on every axis, as shown in equation (6):

$$T = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = \begin{bmatrix} G_1|f| \\ G_2|f| \\ G_3|f| \end{bmatrix} \tag{22}$$

- Helicopter has a torque vector τ with two contributions on the yaw axis (due to the reaction torque):

$$\tau = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_{\psi_1} + \tau_{\psi_2} \end{bmatrix} \tag{23}$$

A simplified dynamics consists in assuming close-hovering conditions (Dzul et al., 2002), i.e. sufficiently small angles α, β, γ , so that matrix in (6) becomes:

$$G(\alpha, \beta, \gamma) \cong \begin{bmatrix} -\alpha \\ \beta \\ -1 \end{bmatrix} \tag{24}$$

with the limit $\alpha, \beta \to 0$:

$$G(\alpha, \beta, \gamma) \cong \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}, \qquad T \cong \begin{bmatrix} 0 \\ 0 \\ -|f| \end{bmatrix} \tag{25}$$

In other words, with this simplifying assumption the lift force of the helicopter (directed along z) is consistently higher in magnitude than the small body forces (directed along x and y). Finally, concerning the translational dynamics, the thrust vector becomes aligned to the z axis exactly like in quadrotor dynamics, while the rotational dynamics is essentially comparable "by default" to the quadrotor without doing any other observation (see equations (16) and (17)). With these achievements, the same considerations and DFL computations seen for the quadrotor can be exploited for the coaxial helicopter, as illustrated in the next section.

4 Simulations and results

DFL algorithm is implemented in a Simulink environment in addition to MATLAB (vs R2022b) initialization and plotting scripts. The overall structure of the program is the following:

- dfl_params.m - Script with init parameters;
- dfl_model.slx - Simulink model used for DFL;
- dfl_plotting.m - Script for printing the model outputs;
- sym_inverse.m - Auxiliary script for computing the inverse of J.

The block schema built in Simulink is described here:

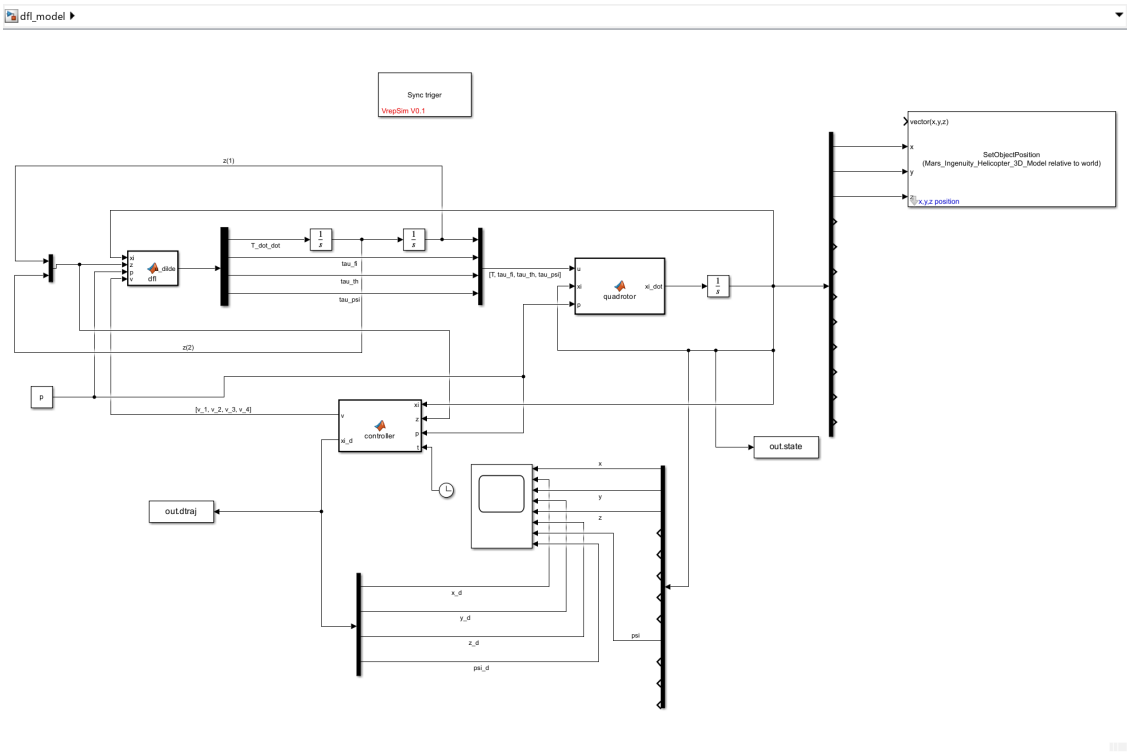

Figure 1: Simulink schema for controlling a quadrotor with DFL

The main blocks are constituted by the following MATLAB functions:

- xi_dot = quadrotor(u, xi, p)
  Quadrotor (or approximated coax helicopter) dynamics, with force/torque u, state xi and parameters p (inherited by dfl_param.m pg) as inputs and evolution xi_dot as output;
- (v,xi_d) = controller(xi, z, p, t)
  Stabilizing controller, with state xi, compensator z, parameters p and time t (provided by a clock block) as inputs and errors v, desired state xi_d (for plotting purposes) as outputs;
- u_dilde = dfl(xi, z, p, v) - Dynamic Feedback Linearizator, with state xi, compensator z, parameters p and errors v as input and modified force/torque u_dilde (followed by a double integrator on the thrust channel) as output;

A trajectory tracking task is tested, with a diagonal parabolic trajectory simulating a take-off and landing operation; the following plots show the convergence of the output trajectory towards the desired trajectory in any direction:

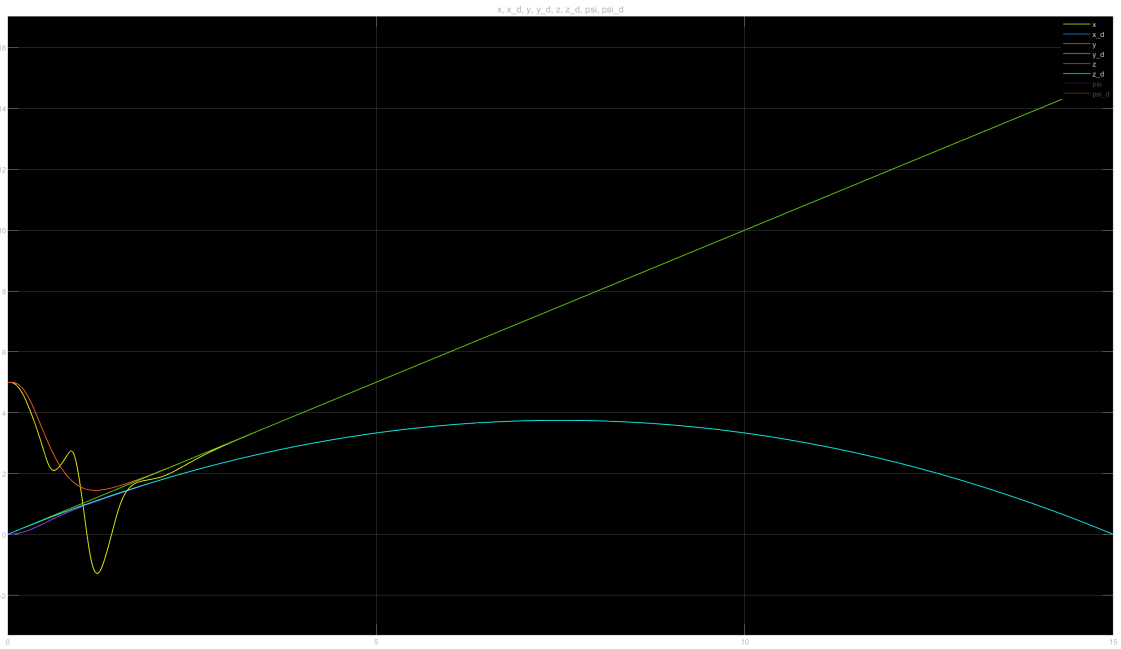

Figure 2: Trajectory tracking - desired and simulated 3D position plotted in Simulink

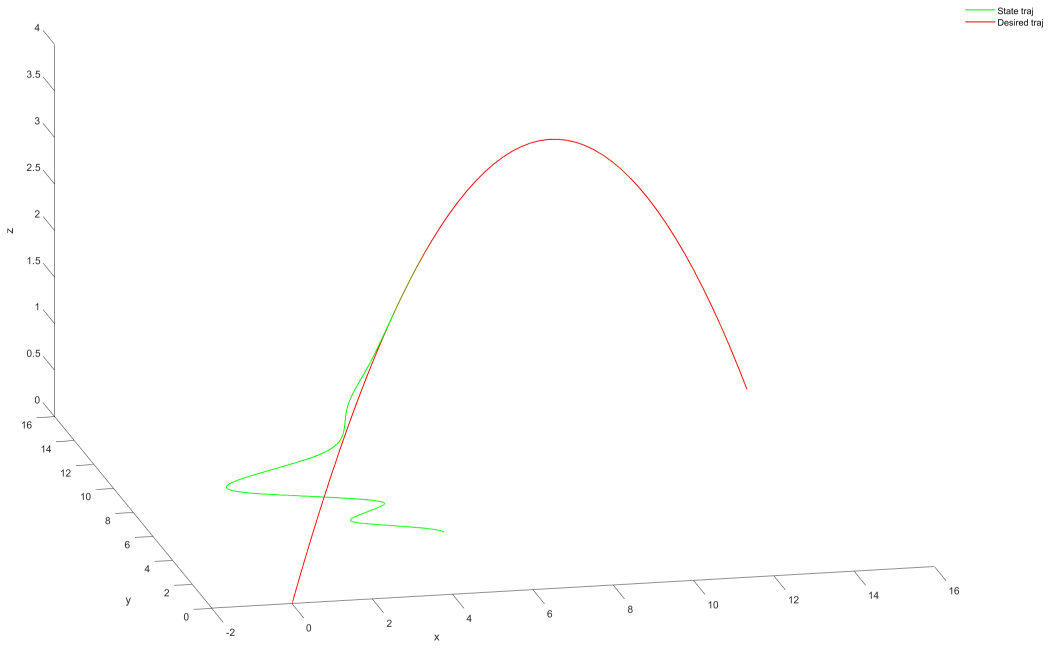

Figure 3: Trajectory tracking - desired and simulated 3D position plotted in MATLAB

Also, the yaw angle is well controlled through the algorithm:

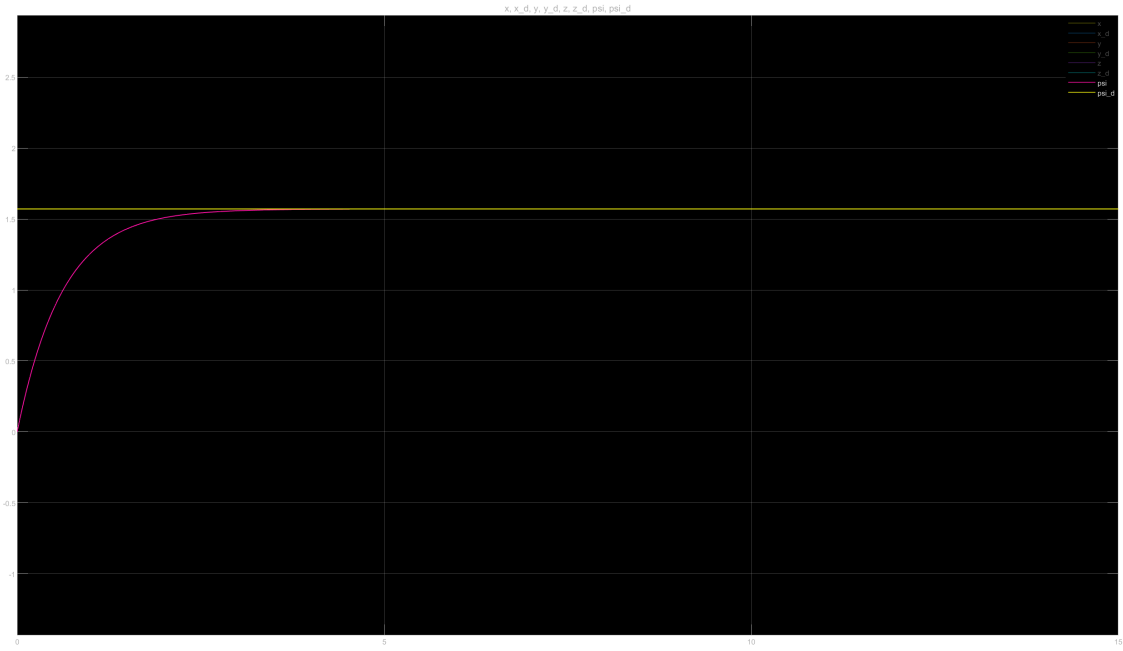

Figure 4: Trajectory tracking - desired and simulated orientation (ψ only)

Furthermore, an integration with CoppeliaSim suite (vs. 4.7.0) through RemoteAPIs is tried to reproduce the trajectory of a 3D model of Ingenuity drone (.stl file) imported directly from the NASA website and composed of over 9000 triangles (NASA, 2024):

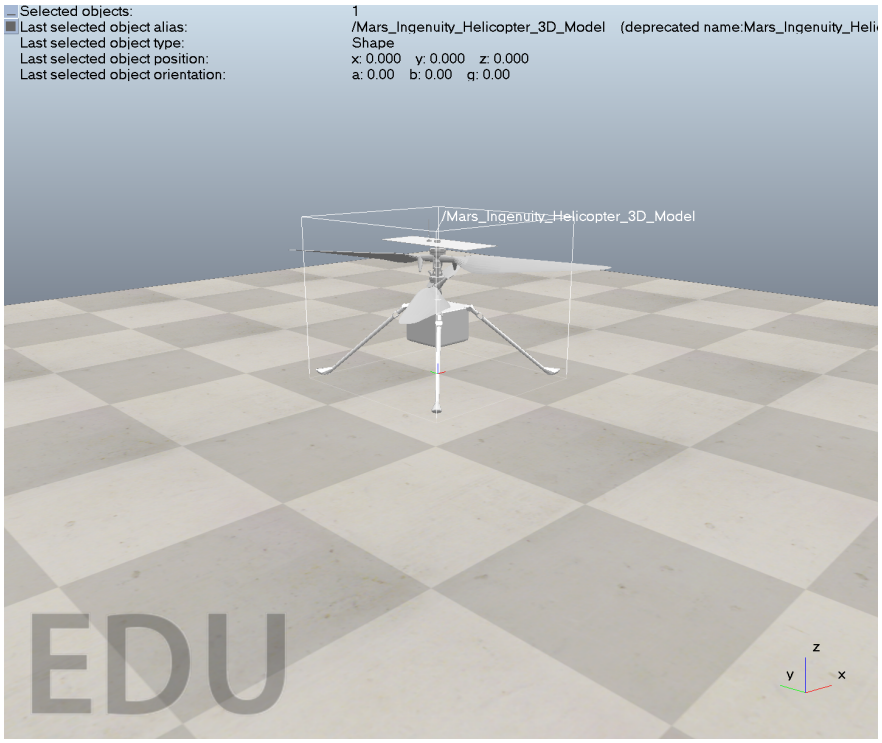

Figure 5: Ingenuity helicopter - 3D model in CoppeliaSim

The integration is performed using a library found on GitHub (sunshineharry, 2023) that consists in exploiting some functionalities taken from the Robotics Toolbox in MATLAB together with some custom wrappers for CoppeliaSim APIs. Among the many, two important ones have been added to MATLAB init path and inserted in the Simulink model:

- Sync trigger - Used to synchronize MATLAB solver step with CoppeliaSim simulation dt (50 ms);
- SetObjectPosition - Block that takes into input the state subvector related to the trajectory of the drone (x,y,z) and provides it as input to the CoppeliaSim object referred inside the block, so that its evolution in position can be simulated in real-time.

The last function actually is cited but not provided inside the library, it is developed ex novo by taking inspiration from the other functions available (GetObjectPose.m, SetJointVelocity.m):

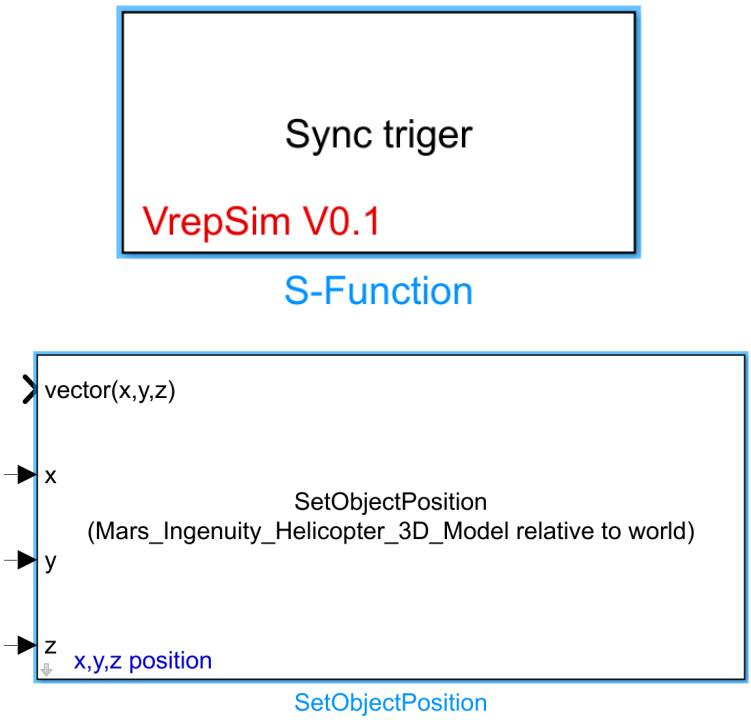

Figure 6: Sync trigger (a) and SetObjectPosition (b) blocks

The scene configured in CoppeliaSim is made up of the following elements:

- DefaultCamera - Standard object;
- XYZCameraProxy - Standard object;
- DefaultLights - Standard object;
- Floor - Standard object;
- Mars_Ingenuity_Helicopter_3D_Model - .stl model of Ingenuity UAV, 9060 triangles;
  1. Revolute_joint_up - Joint element to simulate rotation of the upper rotor;
  2. Upper_rotor - Subsection of the Ingenuity model involving only the two upper blades;

3. Script_up - LUA script setting the velocity of upper rotor;
4. Revolute_joint_down - Joint element to simulate rotation of the lower rotor;
5. Lower_rotor - Subsection of the Ingenuity model involving only the two lower blades.
6. Script_down - LUA script setting the velocity of lower rotor.

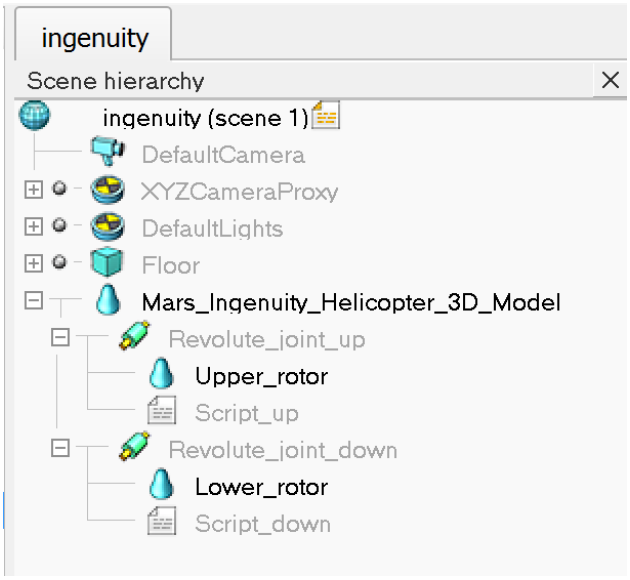

Figure 7: CoppeliaSim scene hierarchy

The main object hierarchy is clear: joints are decoupled and put on the first level in order to control the motion of the rotors, rotors are put on the second level together with the controlling scripts; the decoupling between the upper and lower rotors is done through the shape edit mode available in CoppeliaSim, with about 2400 triangles each:

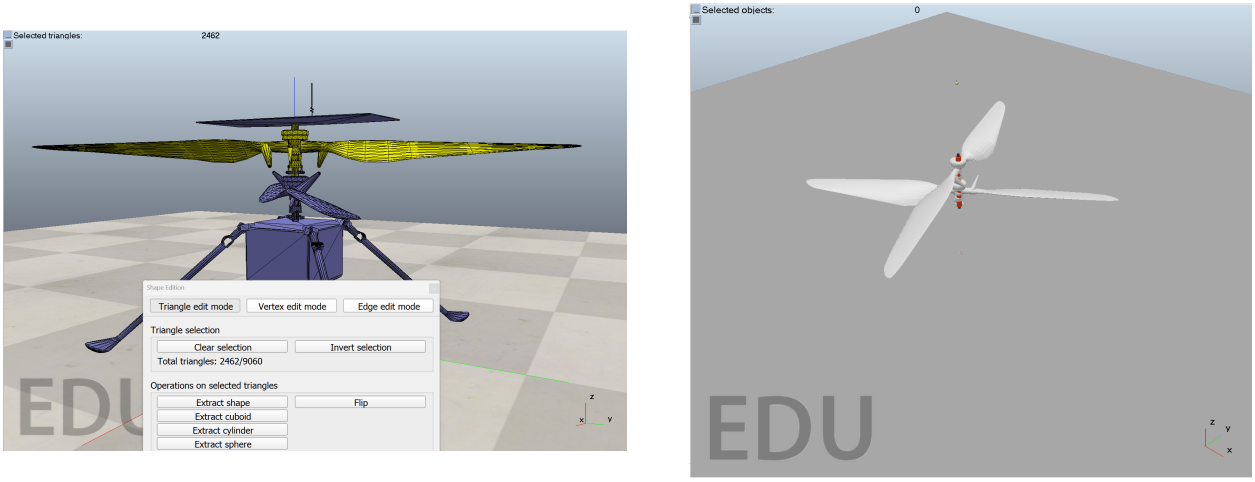

Figure 8: Rotors selection (a) and joint-rotors layer (b)

To command the joints, the mode is set to kinematic and the control to velocity for both, keeping dynamic properties of all the objects untouched. By running the CoppeliaSim simulation first, followed by Simulink simulation, the Ingenuity model will move accordingly to the Cartesian position evolution produced in time by the DFL algorithm, converging after a while to the desired trajectory (e.g. spiral):

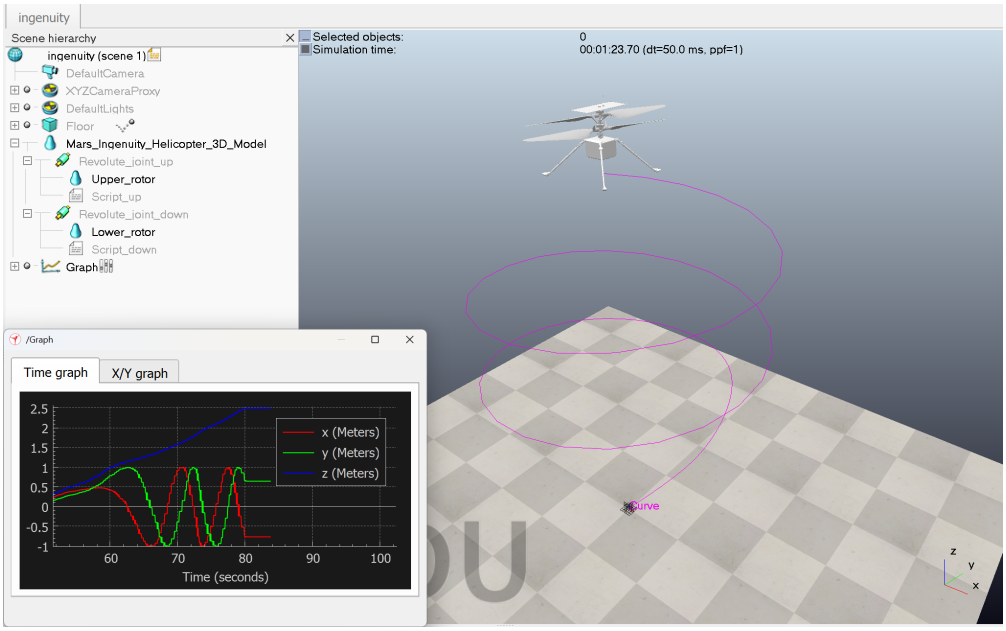

Figure 9: Trajectory tracking (spiral) with Ingenuity UAV in CoppeliaSim

The details of the code written in MATLAB (in particular, parameters, desired trajectories and the script to set the object position in CoppeliaSim) and LUA (in particular the scripts to simulate rotors velocity control) can be found in the appendices.

5 Conclusion

In this work, a different perspective to modelize and control the Ingenuity helicopter is presented. By exploiting the quasi-hovering condition on the robot dynamics, a simplification of the dynamics itself occurs so that it becomes comparable to a quadcopter. Possible improvements could be:

- Recovering into the dynamics also the discharged terms (noise, actuation coupling, external forces/torque etc.) to produce a complete DFL control;
- Relaxing the close-hovering condition and work on the linearizability of Ingenuity with a 3D thrust (see equation (22)).

For the first point, the issue of robustness with respect to aerodynamic perturbations should be taken into account (it is very difficult to model them, (Hua, 2009)). For the latter point, the intuition is that, since for the general quadrotor the modified input $\tilde{u}$ is 4-dimensional and contains the thrust vector with its only direction T_z, the modified input for a coaxial helicopter should be 6-dimensional with all the components of T inside. As a consequence, the square Jacobian J mapping $\tilde{u}$ with the modified output η will be [6 x 6] and the modified output itself will be a 6D vector containing the full pose of the robot. In the end, the Jacobian J appears to be nonsingular by deriving 4 times the Cartesian position (first 3 output components) and 2 times the orientation (last 3 output components). In summary:

$$\tilde{u} = \begin{bmatrix} \ddot{T}_x \\ \ddot{T}_y \\ \ddot{T}_z \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix}, \qquad J = \begin{bmatrix} j_{11} & j_{12} & \cdots & j_{16} \\ j_{21} & \ddots & \cdots & \vdots \\ \vdots & \cdots & \ddots & \vdots \\ j_{61} & \cdots & \cdots & j_{66} \end{bmatrix}, \qquad \eta = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \end{bmatrix} \tag{26}$$

$$\begin{bmatrix} x^{(4)} \\ y^{(4)} \\ z^{(4)} \\ \phi^{(2)} \\ \theta^{(2)} \\ \psi^{(2)} \end{bmatrix} = J\tilde{u} + l \tag{27}$$

References

Chen, L. and McKerrow, P. J. (2007). Modelling the lama coaxial helicopter. In *Proceedings of the Australasian Conference on Robotics and Automation*, Brisbane, Australia.

Dzul, A., Hamel, T., and Lozano, R. (2002). Modeling and nonlinear control for a coaxial helicopter. In *IEEE International Conference on Systems, Man and Cybernetics*, Yasmine Hammamet, Tunisia.

Grip, H. F., Scharf, D. P., Malpica, C., Johnson, W., Mandic, M., Singh, G., and Young, L. (2018). Guidance and control for a mars helicopter. In *AIAA Guidance, Navigation, and Control Conference*, Kissimmee, Florida.

Hua, M.-D. (2009). *Contributions to the automatic control of aerial vehicles.* PhD thesis, University of Nice Sophia Antipolis.

Hua, M.-D., Hamel, T., Morin, P., and Samson, C. (2013). *Introduction to Feedback Control of Underactuated VTOL Vehicles.* IEEE Control Systems Magazine.

Martin, M. S., Mettler, B., Allan, B., Young, L., Mandic, M., Scharf, D. P., Malpica, C., Johnson, W., and Grip, H. F. (2017). *Flight Dynamics of Mars Helicopter.* Jet Propulsion Laboratory, National Aeronautics and Space Administration.

NASA (2024). Mars ingenuity helicopter, 3d model.

Seisan, F. Z. (2012). *Modeling and Control of a Co-Axial Helicopter*. PhD thesis, University of Toronto.

sunshineharry (2023). Vrepsimulink.

Vendittelli, M. (2024). Ingenuity dynamic model ingenuity dynamic model. Slides of Elective in Robotics course.

Wikipedia (2024). Ingenuity (helicopter).

A Model parameters

The list of parameters initialized in MATLAB before running the simulations:

Description	Name	Value
Mass (kg)	m	1.80
MARS gravity (m/s^2)	g	3.71
Inertia matrix ($kg * m^2$)	I_x	0.024
	I_y	0.024
	I_z	0.024
Polynomial coefficients	c0	1764
	c1	1092
	c2	253
	c3	26
Initial state	x0	5
	y0	5
	z0	0
	v_x0	0
	v_y0	0
	v_z0	0
	phi0	0
	th0	0
	psi0	0
	w_x0	0
	w_y0	0
	w_z0	0

Table 1: Physical and control parameters

The components of the inertia matrix have been estimated in the paper "Flight Dynamics of a Mars Helicopter" by NASA (Martin et al., 2017).
The desired trajectories tested are the following:

Trajectory	Name	Value
Parabolic	x_d	t
	y_d	t
	z_d	$-(1/15)t^2 + t$
	psi_d	$\pi/2$
Spiral	x_d	$cos(t)$
	y_d	$sin(t)$
	z_d	$1 + t/10$
	psi_d	$\pi/2$

Table 2: Desired trajectories

With time variable t (s) in the range [0,15].

B MATLAB library (Remote API functions)

SetObjectPosition.m script, main library used to send data from Simulink to CoppeliaSim:

```
function [sys,x0,str,ts] = SetObjectPosition(t,x,u,flag,vrep,clientID,
    object_name,relative_object_name)
    switch flag
        case 0
            [sys,x0,str,ts]=mdlInitializeSizes;
        case 1
            sys=mdlDerivatives(t,x,u);
        case 2
            sys=mdlUpdate(t,x,u,vrep,clientID,object_name,
                relative_object_name);
        case 3
            sys=mdlOutputs(t,x,u)
        case {4,9}
            sys=[];
        otherwise
            error(['Unhandled flag = ',num2str(flag)]);
    end
end

function [sys,x0,str,ts] = mdlInitializeSizes
    sizes = simsizes;
    sizes.NumContStates  = 0;
    sizes.NumDiscStates  = 0;
    sizes.NumOutputs     = 0;
    sizes.NumInputs      = 3;
    sizes.DirFeedthrough = 1;
    sizes.NumSampleTimes = 1;
    sys = simsizes(sizes);
    x0  = [];
    str = [];
    ts  = [0 0];
end
function sys = mdlUpdate(t,x,u,vrep,clientID,object_name,
    relative_object_name)
    [~, object_handle] = vrep.simxGetObjectHandle(clientID, object_name,
        vrep.simx_opmode_blocking);

    if strcmp(relative_object_name, 'world')
        relative_handle = -1;
    elseif strcmp(relative_object_name, 'parent')
        relative_handle = vrep.sim_handle_parent;
    else
        [~, relative_handle] = vrep.simxGetObjectHandle(clientID,
            relative_object_name, vrep.simx_opmode_blocking);
    end

    vrep.simxSetObjectPosition(clientID, object_handle, relative_handle, [
        u(1),u(2),u(3)], vrep.simx_opmode_oneshot);
    sys = [];
end
```

C LUA scripts

Script_up.lua, used to set a target velocity for the upper rotor of the Ingenuity model in CoppeliaSim:

```
function sysCall_init()
    sim = require('sim')
    object = sim.getObject(":/Revolute_joint_up")
end

function sysCall_actuation()
    sim.setJointTargetVelocity(object,5)
end
```

Script_down.lua, used to set a target velocity for the lower rotor of the Ingenuity model in CoppeliaSim:

```
function sysCall_init()
    sim = require('sim')
    object = sim.getObject(":/Revolute_joint_down")
end

function sysCall_actuation()
    sim.setJointTargetVelocity(object,-5)
end
```