Exercise 3

Modeling and control of a multi-copter

Mina Kamel, Marija Popović, Alex Millane

Question 1:

In this exercise, the full dynamic model of a quadcopter has to be derived assuming that the vehicle is a rigid body. The dynamic model has to represented as a set of ordinary differential equations. The quadcopter structure is shown in Figure 1 including forces and torques acting on the vehicle and inertial and body frames.

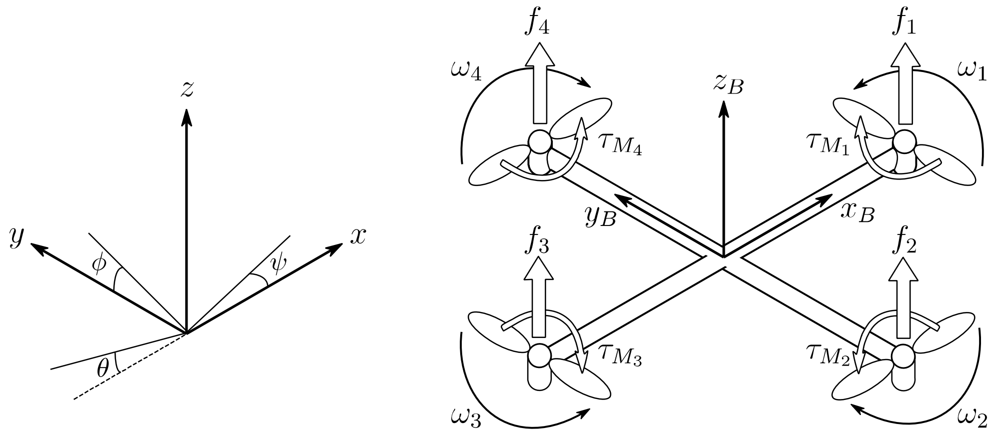

FIGURE 1 INERTIAL AND BODY FRAME OF QUADCOPTER

a. Derive the dynamic model of the quadcopter $\dot{\boldsymbol{X}} = \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{U})$ in terms of forces and drag torques generated by each propeller. For convenience, express the vehicle's position and velocity in the inertial frame while angular velocity in the body frame.
   Hints: The state vector $\boldsymbol{X}$ is given by

$$\boldsymbol{X} = (\boldsymbol{p} \quad \boldsymbol{v} \quad \boldsymbol{R}_{\boldsymbol{W},\boldsymbol{B}} \quad \boldsymbol{\omega})^{\boldsymbol{T}}$$

   where $\boldsymbol{p}$ is the vehicle position, $\boldsymbol{v}$ is the vehicle velocity in inertial frame, $\boldsymbol{R}_{\boldsymbol{W},\boldsymbol{B}}$ is the rotation matrix between body frame and inertial frame and $\boldsymbol{\omega}$ is the body angular velocity.
   While the control input vector $\boldsymbol{U}$ is the virtual control input (as shown in Slide 11 of the lecture slides).
   The time derivation of the rotation matrix $\boldsymbol{R}_{\boldsymbol{W},\boldsymbol{B}}$ is given by

$$\frac{d}{dt}\boldsymbol{R}_{\boldsymbol{W},\boldsymbol{B}} = \boldsymbol{R}_{\boldsymbol{W},\boldsymbol{B}}\widehat{\boldsymbol{\omega}}$$

   where the hat operator is the skew-symmetric matrix operator.

b. Show that the system model is composed of two subsystems, translational dynamics and attitude dynamics.

c. (**Extra**) A nonlinear system is called *differentially flat* if there exists a set of output variables $\boldsymbol{Y} = \boldsymbol{h}(\boldsymbol{x})$ (called flat outputs) such that the system state $\boldsymbol{X}$ and control input **U** can be

written as a function of $\boldsymbol{Y}$ and finite number of its time derivatives. This property is interesting for system control and trajectory generation.
Show that given the flat output $\boldsymbol{Y} = (\boldsymbol{p} \quad \psi)^T$, the full system state $\boldsymbol{X}$ can be written as a function of $\boldsymbol{Y}, \dot{\boldsymbol{Y}}, \ddot{\boldsymbol{Y}}, \dddot{\boldsymbol{Y}}, \dots, \boldsymbol{Y}^{(n)}$.

Question 2:

In this exercise, a PD attitude controller is to be designed and analyzed using MATLAB and Simulink.

a. First, linearize the vehicle attitude dynamics around hovering condition.

b. Write the control input as a function of measured attitude ϕ, θ, ψ and desired attitude ϕ_d, θ_d, ψ_d.

c. Write the linearized system closed-loop dynamics by plugging the control action in the linearized model obtained from Part a. What is the order of the closed loop system?

Now, use the `quadcopter.slx` Simulink model to simulate the quadcopter model and to implement attitude controller. You can see that the quadcopter model is divided into 2 subsystems (shown in cyan). The first subsystem is the attitude dynamics, and the second subsystem is the translational dynamics. Your first task is to complete the model equations obtained in Question 1, then implement an attitude PD controller and position PID controller.

The vehicle and controller tuning parameters are stored into `param` struct that can be modified in `parameters.m`.

d. The translational dynamics block is getting as input the vehicle attitude represented by a rotation matrix $\boldsymbol{R}$ and total force generated by the propellers U_1. In the block `translational_dynamics_eqn` complete the translational dynamics as obtained from Question 1.

e. The attitude dynamics subsystem is getting as input the torque generated by propellers around the vehicle body axes U_2 , U_3 , U_4. In the block `calculate_angular_acc` write the expression of the angular acceleration obtained from Question 1.

f. Now, we implement the attitude PD controller. In the block `PD_attitude_controller_eqn` fill in the control action equations as obtained from Part b. Apply step references to desired roll, pitch, yaw and tune the controller until you are satisfied with the step response. The initial controller parameters in the `param` struct are a reasonable initial guess.