Robot Dynamics Exercise: Modeling and Control of Multicopter

Mike Allenspach

December 1st, 2021

Abstract

The aim of this exercise is to derive the linearized equations of motion of a hexacopter and to implement control algorithms to stabilize its attitude, as well as track velocity references. You will first derive all necessary equations by hand and then the implementation will be done in MATLAB and SIMULINK.

Introduction

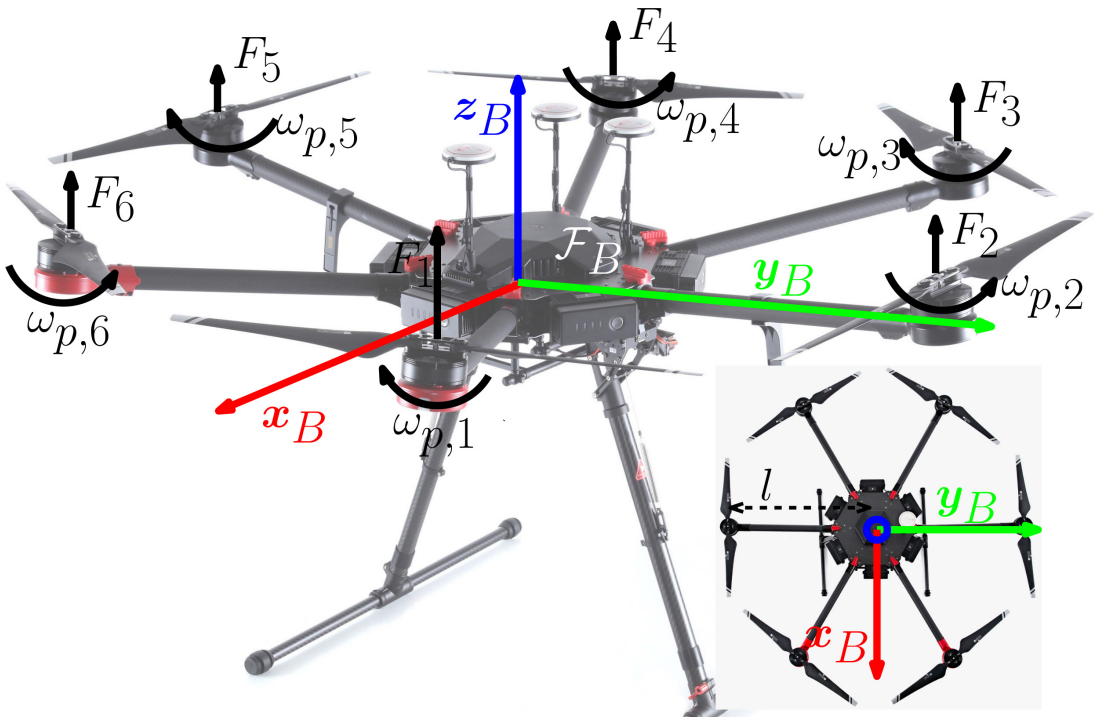

Figure 1: Hexacopter model. Note that the rotation arrows on the propellers represent the spinning direction of the propeller.

This exercise is divided in two parts. In the first section, the linearized dynamic model of a hexacopter will be derived. You further have to derive the mapping between the force generated by each propeller and the total forces and moments acting on the hexacopter rigid body.

In the second section, you design and tune a PD controller to stabilize the attitude of the platform. On top of the attitude controller, you will implement a velocity tracking controller to make the platform follow a reference velocity. It is recommended to review the previous lecture slides on modeling and control of multirotors.

Dynamic Model

In this exercise, the linearized equations of motion of a hexacopter have to be derived assuming that the vehicle is a rigid body. The dynamic model has to be represented as a set of ordinary differential equations. The hexacopter structure is shown in Figure 1, including propeller forces acting on the vehicle and the body frame.

1. Denote the state and virtual control inputs (thrust and body torques) of the hexacopter as

$$\boldsymbol{x} = \begin{pmatrix} {}_B\boldsymbol{v}^\top & \boldsymbol{\Theta}^\top & {}_B\boldsymbol{\omega}^\top \end{pmatrix}^\top \qquad \boldsymbol{u} = \begin{pmatrix} U_1 & U_2 & U_3 & U_4 \end{pmatrix}^\top, \tag{1}$$

where ${}_B\boldsymbol{v},\ {}_B\boldsymbol{\omega} \in \mathbb{R}^3$ are the linear and angular velocity expressed in body frame and $\boldsymbol{\Theta} = \begin{pmatrix} \phi & \theta & \psi \end{pmatrix}^\top$ are the roll ϕ, pitch θ and yaw ψ Tait-Bryan Angles.

Based on the nonlinear dynamics $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$:

$${}_B\dot{\boldsymbol{v}} = -{}_B\boldsymbol{\omega} \times {}_B\boldsymbol{v} + \begin{pmatrix} 0 \\ 0 \\ \frac{U_1}{m} \end{pmatrix} - \boldsymbol{R}_{IB}^\top \, {}_I\boldsymbol{g} \tag{2}$$

$$\frac{d}{dt}\begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} = \begin{pmatrix} 1 & 0 & -s_\theta \\ 0 & c_\phi & c_\theta s_\phi \\ 0 & -s_\phi & c_\theta c_\phi \end{pmatrix}^{-1} {}_B\boldsymbol{\omega} \tag{3}$$

$${}_B\dot{\boldsymbol{\omega}} = \boldsymbol{J}^{-1}\left(-{}_B\boldsymbol{\omega} \times \boldsymbol{J}\, {}_B\boldsymbol{\omega} + \begin{pmatrix} U_2 \\ U_3 \\ U_4 \end{pmatrix} \right), \tag{4}$$

show that the hover configuration

$$\boldsymbol{x}_{eq} = \boldsymbol{0},\ \boldsymbol{u}_{eq} = \begin{pmatrix} mg & \boldsymbol{0}^\top \end{pmatrix}^\top \tag{5}$$

is an equilibrium and linearize around it

$$\dot{\boldsymbol{x}} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}|_{\boldsymbol{x}_{eq}, \boldsymbol{u}_{eq}} \left(\boldsymbol{x} - \boldsymbol{x}_{eq} \right) + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}|_{\boldsymbol{x}_{eq}, \boldsymbol{u}_{eq}} \left(\boldsymbol{u} - \boldsymbol{u}_{eq} \right) \tag{6}$$

2. From Figure 1 derive the *allocation matrix* $\boldsymbol{A}$ that maps between the square of rotor angular velocities $\omega_{p,i}^2$ and the total forces and torques acting on the platform:

$$\begin{pmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{pmatrix} = \mathbf{A} \begin{pmatrix} \omega_{p,1}^2 \\ \vdots \\ \omega_{p,6}^2 \end{pmatrix} \tag{11}$$

Note that $\mathbf{A}$ depends on the vehicle arm length l, the thrust constant b and drag constant d.

Control

In this exercise, attitude and velocity controllers will be designed and evaluated using MATLAB and SIMULINK. The control structure is depicted in Figure 2. The outer loop controller (velocity controller) generates commands $\boldsymbol{\Theta}_{ref}$ to the inner loop controller (attitude controller). The commanded force and torques (total thrust U_1 and moments U_2, U_3, U_4) are converted into rotor speed $\omega_{p,i}$ by inverting the expression in Equation (11). Note that matrix $\mathbf{A}$ has dimensionality 4×6. Therefore, the pseudo-inverse is used to solve Equation (11) for $\boldsymbol{\omega}_{p,1\ldots6}$. The pseudo-inverse is given by $\mathbf{A}^\dagger = \mathbf{A}^T \left(\mathbf{A}\mathbf{A}^T \right)^{-1}$.

3. Write the control inputs U_2, U_3, U_4 as a function of measured attitude $\boldsymbol{\Theta} = \begin{pmatrix} \phi & \theta & \psi \end{pmatrix}^\top$, angular velocity ${}_B\boldsymbol{\omega}$ and desired attitude $\boldsymbol{\Theta}_{ref} = \begin{pmatrix} \phi_{ref} & \theta_{ref} & \psi_{ref} \end{pmatrix}^\top$ using the standard PD control approach.

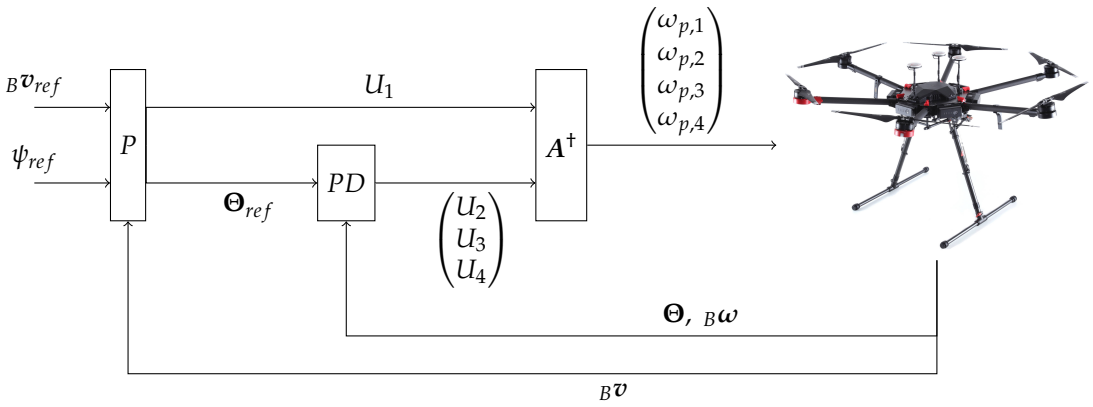

Figure 2: Control structure.

4. On top of the attitude controller, we will design a velocity controller to achieve a desired reference velocity ${}_B\boldsymbol{v}_{ref}$. To this end, you have to design a controller that outputs desired attitude ϕ_{ref}, θ_{ref} and total thrust U_1 as a function of measured velocity ${}_B\boldsymbol{v}$ and desired velocity ${}_B\boldsymbol{v}_{ref}$. A simple proportional controller can be used to achieve this goal. Write down the equations of such a controller.

   Hint: Don't forget to add a feed-forward term for gravity compensation to U_1.

MATLAB &SIMULINK Implementation

Based on the derived equations of motion and control laws, we now use the *hexacopter.slx* SIMULINK model to simulate the hexacopter. We first complete the model equations obtained in the first section of this exercise, then implement an attitude PD controller and velocity P controller.

Note that the vehicle and controller tuning parameters are stored in the *param* struct that can be modified in *parameters.m*.

Task 1

Open the *Vehicle dynamics* block and complete the *Bv_dyn*, *rpy_dyn* and *Bomega_dyn* functions based on the results you obtained in Exercise 1.

Task 2

Open the *parameters.m* file and complete the *allocation_matrix* matrix you obtained in Exercise 2.

Task 3

Now, we implement the attitude PD controller. Complete the *PD_att_ctrl* function (inside Attitude PD controller subsystem) by filling in the control action equations from Exercise 3. Apply step references for roll, pitch and yaw to tune the controller until you are satisfied with the step response (e.g. following Ziegler-Nichols method).
Note: You can switch between velocity and attitude controllers by double clicking on the *Att/Vel switch*.

Task 4

Finally, we implement the velocity P controller. Complete the block *P_vel_ctrl* (inside Velocity P controller subsystem) by filling in the control action equations from Exercise 4. Apply step references to the desired velocity and tune the controller until you are satisfied with the step response.