Exercise 4: Modeling and Control of Multicopter

Weixuan Zhang, Maximilian Brunner

November 27, 2019

Abstract

The aim of this exercise is to derive the full dynamic model of a hexacopter and to implement control algorithms to stabilize the platform attitude and to achieve velocity reference tracking. The implementation will be done on MATLAB SIMULINK. All necessary files can be downloaded on the Piazza.

Introduction

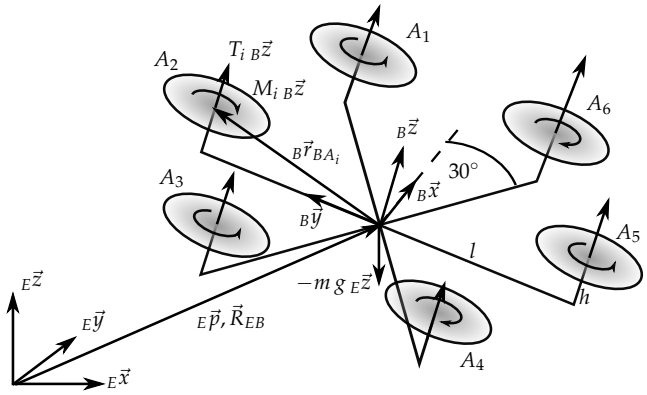

Figure 1: Hexacopter model.

This exercise is divided in two parts. In the first section, the full dynamic model of a hexacopter will be derived. You 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 will linearize the nonlinear system dynamics around hovering condition and design 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 multirotor.

Dynamic Model

In this exercise, the full dynamic model of a hexacopter 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 hexacopter structure is shown in Figure 1 including forces and torques acting on the vehicle and inertial and body frames.

1. Derive the dynamic model of the hexacopter $\dot{\boldsymbol{X}} = \boldsymbol{f}(\boldsymbol{X}, \boldsymbol{U})$ 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.
   Hint: The state vector is given by:

$$\boldsymbol{X} = \begin{pmatrix} {}_E\boldsymbol{p} & {}_B\boldsymbol{v} & \boldsymbol{R}_{EB} & \boldsymbol{\omega} \end{pmatrix}^T \tag{1}$$

where ${}_E\boldsymbol{p}$ is the vehicle position expressed in inertial frame, ${}_B\boldsymbol{v}$ is the vehicle velocity in body frame, $\boldsymbol{R}_{EB}$ is the rotation matrix between body frame and inertial frame, $\boldsymbol{\omega}$ is the body angular velocity. and the control input vector $\boldsymbol{U}$ is the virtual control input (see Slide 29 of the lecture slides for the quadcopter case). The time derivation of the rotation matrix $\boldsymbol{R}_{EB}$ is given by

$$\frac{d}{dt}\boldsymbol{R}_{EB} = \boldsymbol{R}_{EB}\hat{\boldsymbol{\omega}} \tag{2}$$

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

2. From Figure 1 write the linear mapping between the square of rotors angular velocity $\omega_{p,i}^2$ and the total forces and torques acting on the platform. i.e. find the so called *allocation matrix* $\mathbf{A}$ in the following relation:

$$\begin{pmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{pmatrix} = \mathbf{A} \begin{pmatrix} \omega_{p,1}^2 \\ \vdots \\ \omega_{p,6}^2 \end{pmatrix} \tag{3}$$

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

Control

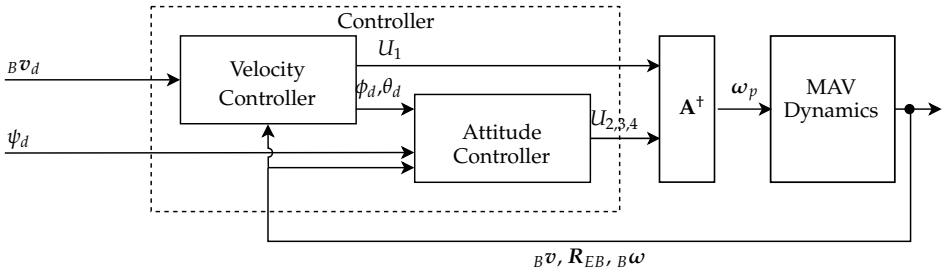

Figure 2: Control structure.

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 to the inner loop controller (attitude controller). The commands (total thrust U_1 and moments $U_2, \dots U_4$) are converted into rotor speed $\omega_{p,i}$ by inverting the expression in Equation (3). Note that matrix $\mathbf{A}$ has dimensionality 4×6. Therefore, the pseudo-inverse is used to solve Equation (3) for $\boldsymbol{\omega}_{p,1\dots6}$. The pseudo-inverse is given by $\mathbf{A}^\dagger = \mathbf{A}^T \left(\mathbf{A}\mathbf{A}^T\right)^{-1}$.

3. Linearize the vehicle linear and angular velocity dynamics around hovering condition assuming zero heading angle ψ.

4. Write the control inputs U_2, U_3, U_4 as a function of measured attitude ϕ, θ, ψ, angular velocity $\boldsymbol{\omega}$ and desired attitude ϕ_d, θ_d, ψ_d using the standard PD control approach.

5. On top of the attitude controller, we will design a velocity controller to achieve a desired reference velocity ${}_B\boldsymbol{v}_d$. To this end, you have to design a controller that outputs desired attitude ϕ_d, θ_d and total thrust U_1 as a function of measured velocity ${}_B\boldsymbol{v}$ and desired velocity ${}_B\boldsymbol{v}_d$. 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.

6. Now, use the *hexacopter.slx* SIMULINK model to simulate the hexacopter model and to implement attitude and velocity controllers. You can see that the vehicle 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 the first section of this exercise, then implement an attitude PD controller and velocity P controller.

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

Task 1

Double click on the *Translational dynamics* block to open it and double click on *translational_dynamics_eqn* to edit the translational dynamics equations. The input to this block is the attitude of the vehicle expressed by the rotation matrix $\boldsymbol{R}_{EB}$ and the angular velocity of the propellers $\boldsymbol{\omega}_p$. Complete the acceleration equation.

Task 2

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

Task 3

Now, we implement the attitude PD controller. In the block *PD_attitude_controller_eqn* (inside Attitude PD controller subsystem) fill in the control action equations. 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.

Note: You can switch between velocity and attitude controllers by double click on the *Att/Vel switch*.

Task 4

Now, we implement the velocity P controller. In the block *P_velocity_controller_eqn* (inside Velocity P controller subsystem) fill in the control action equations. Apply step references to desired velocity 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.