Project: Part 5

24-677 Modern Control Theory - Modern Control for Robotics

Prof. D. Zhao

Due: Dec 6, 2024, 11:59 pm. Submit within the deadline.

- Your online version and its timestamp will be used for assessment.
- We will use Gradescope to grade. The link is on the panel of CANVAS. If you are confused about the tool, post your questions on Campuswire.
- Submit **lqr_controller.py** and **adaptive_controller.py** to Gradescope under **P5-code** and your solutions in **.pdf** format to **P5-writeup**. Insert the output tracking performance plot image in the **.pdf**. We will test your controller and manually check all answers.
- We will make extensive use of Webots, an open-source robotics simulation software, for this project. Webots is available here for Windows, Mac, and Linux.
- For Python usage with Webots, please see the Webots page on Python. Note that you may have to reinstall libraries like `numpy`, `matplotlib`, `scipy`, etc. for the environment you use Webots in.
- Please familiarize yourself with Webots documentation, specifically their User Guide and their Webots for Automobiles section, if you encounter difficulties in setup or use. It will help to have a good understanding of the underlying tools that will be used in this assignment. To that end, completing at least Tutorial 1 in the user guide is highly recommended.
- If you have issues with Webots that are beyond the scope of the documentation (e.g. the software runs too slow, crashes, or has other odd behavior), please let the TAs know via Campuswire. We will do our best to help.
- We advise you to start with the assignment early. All the submissions are to be done before the respective deadlines of each assignment. For information about the late days and scale of your Final Grade, refer to the Syllabus in Canvas.

1 Introduction

In the previous projects, we have learned how to apply modern control techniques to an Unmanned Ground Vehicle (UGV). In this project, we will step up the difficulty and investigate how to control an Unmanned Aerial Vehicle (UAV).

Quadrotor drones has risen in popularity in recent year for a variety of uses, such as aerial photography for topology and agriculture, search and rescue missions, and product delivery. Ever-increasing demands on safety and performance have necessitated the development of more and more sophisticate flight control systems. Unlike ground vehicle, these flying vehicles could be inherently unstable due to many sources of uncertainty such as actuator degradation, external disturbances, and complex unmodeled dynamics (blade flapping and asymmetric angular speed of propellers), which often results in a crash.

In this project, we will make use of adaptive control theory to augment a LQR-based MPC controller and test its effectiveness in the event of a single quadrotor motor of a DJI$^{\text{TM}}$ Mavic 2 Pro experience a 50% pr higher percent lost of thrust during hovering. As before, the experiment will be performed in the Webots simulator.

This project is the last part of the course project series:

P5 (a) Design a baseline LQR controller to fly the quadrotor without any motor failure.

(b) Design a MRAC to fly the quadrotor with 50% lost of thrust in one motor.

2 Model

We will use a popular model of the quadrotor as the control model. As shown in Figure 1, the inertial frame is fixed to the ground. The roll axis of the quadrotor is pointing forward w.r.t. to the drone body. The pitch axis is pointing to the left. The yaw axis is pointing upwards. We will present the simplified nonlinear dynamics model and linearized dynamics model in Section 2.1 and 2.2. The model parameters are defined in Table 1.

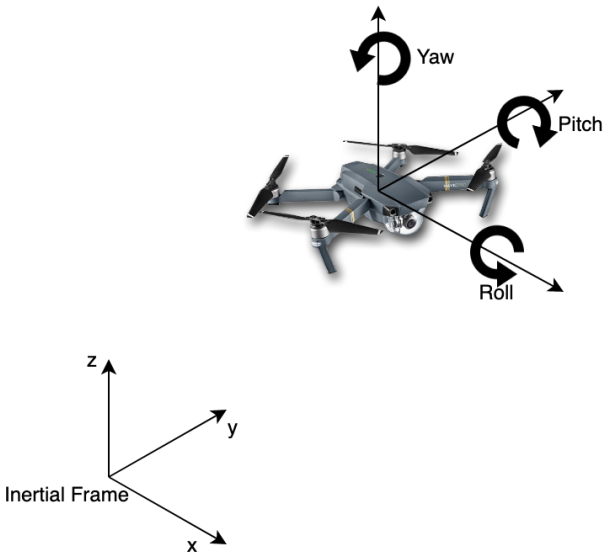

Figure 1: Coordinate systems

2.1 Nonlinear dynamics

The dynamics of quadrotor helicopters have been studied in detail by many researchers. Assuming the vehicle as a rigid-body and operating at low speed, the dynamics are given by:

$$\ddot{x} = (\cos(\phi)\sin(\theta)\cos(\psi) + \sin(\phi)\sin(\psi))\frac{U_1}{m}$$
$$\ddot{y} = (\cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi))\frac{U_1}{m}$$
$$\ddot{z} = -g + \cos(\phi)\cos(\psi)\frac{U_1}{m}$$
$$\ddot{\phi} = \dot{\theta}\dot{\psi}(\frac{I_y - I_z}{I_x}) - \frac{J_R}{I_x}\dot{\theta}\Omega_R + \frac{1}{I_x}U_2$$
$$\ddot{\theta} = \dot{\phi}\dot{\psi}(\frac{I_z - I_x}{I_y}) - \frac{J_R}{I_y}\dot{\phi}\Omega_R + \frac{1}{I_y}U_3$$
$$\ddot{\psi} = \dot{\phi}\dot{\theta}(\frac{I_x - I_y}{I_z}) + \frac{1}{I_z}U_4$$

where x, y, and z are the positions of the center of mass in the inertial frame; ϕ, θ, and ψ are the Euler angles which describe the orientation of the body-fixed frame with respect to the inertial frame; m, I_x, I_y, and I_z are the mass and moments of inertial of the quadrotor respectively; and J_R and Ω_R are the moment of inertia and angular velocity(i.e. rotor speed) of the propeller blades. U_1 is the collective thrust, U_2, U_3, and U_4 are the roll, pitch and yaw torques generated by the four propellers.

2.2 Linear dynamics

Since the quadrotor typically operates very near the hover position, we can linearize the system by making small angle approximations, neglecting higher order terms and letting $U_1 = mg + \Delta U_1$, which will result in the linear dynamics:

$$\ddot{x} = g\theta$$
$$\ddot{y} = -g\phi$$
$$\ddot{z} = \frac{\Delta U_1}{m}$$
$$\ddot{\phi} = \frac{1}{I_x}U_2$$
$$\ddot{\theta} = \frac{1}{I_y}U_3$$
$$\ddot{\psi} = \frac{1}{I_z}U_4$$

Although this linear system seems a bit over simplified than the nonlinear dynamics, it still captures the dominant features of the quadrotor, and is sufficient near the hover position. As expected, the roll, pitch and yaw input command moments about their respective axes

and the collective input commands acceleration in the positive z-direction. Accelerations in the x- and y-directions are achieved primarily through vectoring the collective thrust.

For the purposes of the controller design, we will use the linearized dynamics.

2.3 Physical Parameters

All relevant physical parameters are listed here. The definitions of d_{1x}, d_{2x}, d_{1y} and d_{1y} are showd in Figure 2.

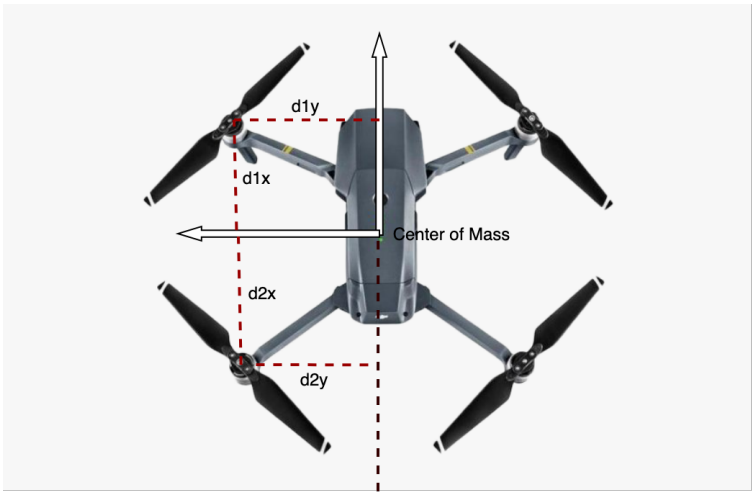

Figure 2: Definition of d_{1x}, d_{2x}, d_{1y} and d_{1y}

Table 1: Model parameters.

Name	Description	Unit	Value
(X, Y, Z)	Vehicle's coordinates in the world frame	m	State
$(\dot{x}, \dot{y}, \dot{z})$	Vehicle's velocity along the direction of vehicle frame	m/s	State
(ϕ, θ, ψ)	Body row, pitch, yaw angle	rad	State
$\left(\dot{\phi}, \dot{\theta}, \dot{\psi}\right)$	Body row, pitch, yaw angle rate	rad/s	State
m	Vehicle mass	kg	0.4
d_{1x}	Length in x-axis from front rotors to the center of mass	m	0.1122
d_{2x}	Length in x-axis from rear rotors to the center of mass	m	0.1171
d_{1y}	Length in y-axis from front rotors to the center of mass	m	0.1515
d_{2y}	Length in x-axis from rear rotors to the center of mass	m	0.1280
I_x	Moment of inertia around x-axis	kg m^2	0.000913855
I_y	Moment of inertia around y-axis	kg m^2	0.00236242
I_z	Moment of inertia around z-aixs	kg m^2	0.00279965
c_t	thrust coefficient	N	0.00026
c_τ	torque coefficient	kg m^2	5.2e-06
ΔT	Simulation timestep	sec	0.01

3 Controllers

In order to access the effectiveness of adaptive controller in the presence of model uncertainty, We will use a LQR controller as the baseline controller for comparison.

3.1 Problem statement

The primary function of the adaptive controller is to accommodate any uncertainties which may arise in the dynamics. We can write the equations of motion in section (2.2) along with these uncertainties as

$$\dot{x}_p = A_p x_p + B_p \Lambda u \tag{1}$$

where $B_p \in R^{n_p \times m}$ is constant and known, $A_p \in R^{n_p \times n_p}$ is constant and unknown, $x_p \in R^{n_p}$, $u \in R^m$, $\Lambda \in R^{m \times m}$ is an unknown diagonal positive definite matrix. The state $x_p = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}]$. The control input $u = [\Delta U_1, U_2, U_3, U_4]$. The goal is to track a reference command $r(t) \in R^m$ in the presence of the unknown A_p, and Λ. We define the system output as

$$y_p = C_p x_p$$

In the case of the quadrotor, the output states are x, y, z, ψ and $m = 4$. The output tracking error is then given by

$$e_y = y_p - r$$

Augmenting (1) with the integrated output tracking error,

$$e_{yI} = \int e_y dt, \quad \dot{e}_{yI} = e_y$$

leads to the extended open loop dynamics

$$\dot{x} = Ax + B\Lambda u + B_c r \tag{2}$$

where $x = [x_p^T \quad e_{yI}^T]^T$is the extended system state vector. The extended open-loop system matrices are given by

$$A = \begin{bmatrix} A_p & 0_{n_p \times m} \\ C_p & 0_{m \times m} \end{bmatrix} \quad B = \begin{bmatrix} B_p \\ 0_{m \times m} \end{bmatrix} \quad B_c = \begin{bmatrix} 0_{n_p \times m} \\ -I_{m \times m} \end{bmatrix}$$

and the extended system output (notice that we abuse the notations here: the output y is different from the the y positions of the center of mass in the inertial frame)

$$y = [C_p \quad 0_{m \times m}]x = Cx$$

3.2 Baseline LQR Controller

A baseline LQR controller

$$u_{bl} = K_{bl}x$$

can be designed for the system in (2) in the case where there is no model discrepancy, i.e., $\Lambda = I_{m\times m}$ and A_{real} is taken at some nominal value $\bar{A}$ where actually $A_{\text{real}} = \bar{A} + A^*$ and A^* contains all the unmodeled terms ($\bar{A}$ is the A in section 3.1). The feedback gains K_{bl} can be selected using standard LQR design techniques.

3.3 Model Reference Adaptive Controller

The reference model used by MRAC is the closed loop system given by (2), again in the case of no uncertainty, along with the control input in Section 3.2

$$\dot{x}_m = \underbrace{\left(\bar{A} + BK_{bl}\right)}_{A_m} x_m + B_c r \tag{3}$$

We can design an adaptive control input

$$u_{ad} = \hat{K}_x^T x_t \tag{4}$$

such that the resulted system (5) can track the reference model in (3)

$$\dot{x} = \left(A + B\Lambda\hat{K}_x^T\right) x + B_c r \tag{5}$$

The adaptive rate of the control gain is given by

$$\dot{\hat{K}}_x^T = -\Gamma x e^T P B$$

where Γ is a diagonal positive definite matrix, $e = x - x_m$ is the model tracking error, and P is the unique symmetric positive definite solution of the Lyapunov equation

$$A_m^T P + P A_m = -Q$$

where Q is also symmetric positive definite.

The augmented structure of the adaptive controller implies that in the nominal case, i.e., the case with no parameter uncertainty, the overall system is equivalent to the baseline control. However, when failures or other uncertainties arise, the adaptive controller works to assist the baseline controller in maintaining stability and performance.

With the Lyapunov function candidate given by

$$V = e^T P e + Tr(\tilde{\theta}^T \Gamma^{-1} \tilde{\theta})$$

where $\tilde{\theta} = \hat{\theta} - \theta$ is the control gain estimation error, it can be shown [7] that the derivative of the Lyapunov function candidate is given by

$$\dot{V} = -e^T P e \leq 0$$

The system is globally asymptotically stable and the tracking error asymptotically converges to 0, that is

$$\lim_{t\to\infty} e(t) = 0$$

4 P5: Problems

Exercise 1. (50 points) (all the files for this exercise are in **ex1** folder) Design a discrete-time infinite horizon LQR controller for the quadrotor to track the command when there is no motor failure. You should complete the code in **lqr_controller.py**. Attach the plot generated by completing 60s simulation and report the error generated at the end of simulation in your write-up. If your error is lower than *lower_bar*, you will receive full score. Otherwise your score will be calculated as $(upper\_bar - your\_error)/(upper\_bar - lower\_bar) * 50$ where *lower_bar* is 0.3 and *upper_bar* is 0.6. Note that the full score for this exercise is 50.

Submission instruction:

1. Submit your script **lqr_controller.py** to P5_programming.
2. In writeup submission, attach (a) the plot generated by completing 60s simulation and (b) the snapshot of final error and score (shown in console)

You are strongly recommended to go through the code base provided. Some hints that may be useful:

- The physical parameters are defined in the base_controller.py.
- Make sure to discretize your continuous state-space system with the provided discrete timestep (self.delT) prior to solving the ARE.
- Using LQR requires designing the Q and R matrices. Q penalizes state variables, while R penalizes control input. Try to think about what form your Q and R matrices should take for good performance. To help you started the tuning, you can take a look at the *Example code* part of the comment in the code base.
  - For Q, large values will restrict changes to the respective states, while smaller values will allow the states to easily change.
  - Similarly, in R, larger values will restrict control input, while smaller values will allow the control input to vary more.
  - Empirically, it is reasonable to set the entries in the Q and R to be

$$\frac{1}{(\textbf{max value of the corresponding state/input})^2}$$

    in order to normalize the penalty weights.

Exercise 2.

1. **(30 points)** (all the files for this exercise are in **ex2** folder) Design a MRAC to fly the quadrotor with 50% lost of thrust in one motor. The motor failure happens after 14s in simulation. You should complete the code in **adaptive_controller.py**. Attach the plot generated by completing 60s simulation and report your percentage lost of thrust. Please refer to Appendix 1 for details on how we simulate motor failure. If your LQR and MRAC controller can stablize the system with 50% lost of thrust in one motor, you will get full score. Otherwise, you will get $(your\_percentage\_lost\_of\_thrust)/50 * 30$ points.

   To get the correct plot, you first need to copy your **lqr_controller.py** from Exercise 1 to the folder of Exercise 2 (ex2/controllers/ex2_controller/). Run the code with LQR controller and obtain the saved data for LQR controller. Follow the instruction below to run either the LQR or adaptive controller again to obtain the data and the plot. Every time you change the percentage lost of thrust, you need to repeat this process.

   In order to change the percentage lost of thrust, go to **ex2_controller.py** and change the value of the variable **lossOfThust** (line 24 of **ex2_controller.py**) to your preferred percentage lost of thrust. If you want to change which controller to use, edit line 27 of **ex2_controller.py** to either
   customController = AdaptiveController(driver, lossOfThust)
   or
   customController = LQRController(driver, lossOfThust)

2. **(20 points)** Find a percentage lost of thrust such that LQR fails (the drone will fall onto the floor) but MRAC is still able to maintain tracking. Report the percentage lost of thrust and attach the plot.

Submission instruction:

1. Submit your script **adaptive_controller.py** to P5_programming.
2. In writeup submission, for each question, (a) report the loss of thrust and (b) attach the plot with this loss of thrust.

Tips and Warnings:

- This simulation may be substantially slower than previous project parts if you do not have a computer with a dedicated graphics card.
- Initialize the gain matrix $\hat{K}_x^T$ to your LQR gain. It will help you stabilize the system.

5 Appendix

5.1 Conversion from control input to rotor speed

For our quadrotor model in webots, the propeller turns the motor angular velocity into a thrust and a (resistant) torque. The resultant thrust T of each motor is given by the product of thrust constants c_t and the square of the angular velocity, where as the resultant torque τ is given by the product of torque coefficient c_τ and the the square of the angular velocity.

$$T = c_t\omega^2$$
$$\tau = c_\tau\omega^2$$

The control input U_1 is simply the sum of the thrust of all propellers, and the row, pitch torque U_2, U_3 are the sum of torque generated by the opposite propellers around certain axes. The yaw torque U_4 is the sum of the torque generated by all the propellers around the z-aixs. Hence, we can write the conversion matrix H from rotor speed $\Omega = [\omega_1^2 \quad \omega_2^2 \quad \omega_3^2 \quad \omega_4^2]^T$ to $U = [U_1 \quad U_2 \quad U_3 \quad U_4]^T$ as:

$$U = H\Omega = c_t \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & -L & 0 & L \\ -L & 0 & L & 0 \\ -\frac{c_\tau}{c_t} & \frac{c_\tau}{c_t} & -\frac{c_\tau}{c_t} & \frac{c_\tau}{c_t} \end{bmatrix} \Omega$$

Inversely, to get the desired motor speeds from control input, we use the inverse of H matrix.

$$U = H\Omega \rightarrow \Omega = H^{-1}U.$$

5.2 Simulating Actuator failure

To simulate around 50% lost of thrust in motor number 1, the control input to be used for the simulation U_{fail} is given by:

$$U_{fail} = H \begin{bmatrix} 0.7 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} H^{-1}U$$

6 Reference

[1] Dydek, Zachary T., Anuradha M. Annaswamy, and Eugene Lavretsky. "Adaptive control of quadrotor UAVs: A design trade study with flight evaluations." IEEE Transactions on control systems technology 21.4 (2012): 1400-1406.