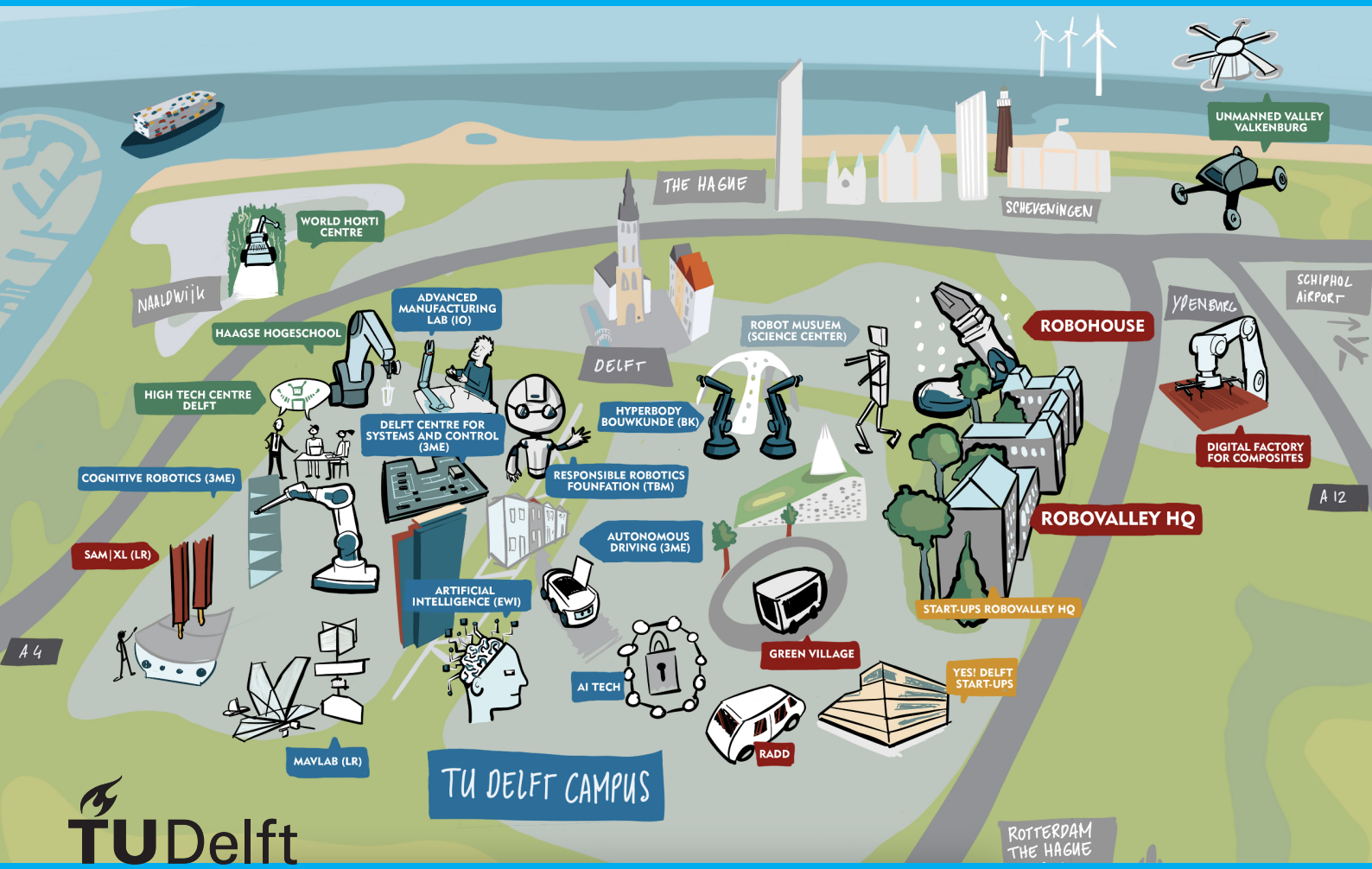
Robot Dynamics & Control: Quadrotor RO47001 - 21/22

Assignment 4

Wei Pan



Preface

1. The course material and assessment of the "Drone" part in this course is build based on a similar course "MEAM 620: Robotics" in the University of Pennsylvania[1]. Also it is also available at Coursera[2]. You are strongly recommended to learn from there.
2. You can consult our teaching assistant, Jingyue Liu, regarding Python programming questions. She is a second year master student of robotics and her email address is `J.Liu-28@student.tudelft.nl`
3. I assume you have been very familiar with Rigid body transformations and rotations. If not, read Chapter 2 and 3 in [2]. You can download here[3]

Wei Pan
Delft, October 21, 2021

[1] `https://alliance.seas.upenn.edu/~meam620/wiki/index.php`
[2] `https://www.coursera.org/learn/robotics-flight/home/welcome`
[3] `https://github.com/yangmingustb/planning_books_1/blob/master/Siciliano%20-%202009%20-%20Robotics%20modelling%2C%20planning%20and%20control.pdf`

Contents

1

Introduction

In this project, you will learn about quadrotor dynamics and develop algorithms to control them. A thorough understanding of the theory and clean implementation of the controller is crucial because you will use the base Matlab code. You will firstly have a quick test in Matlab and visualize the control performance of your controller. During the interactive session, implement in Python, and OpenAI Gym [1].

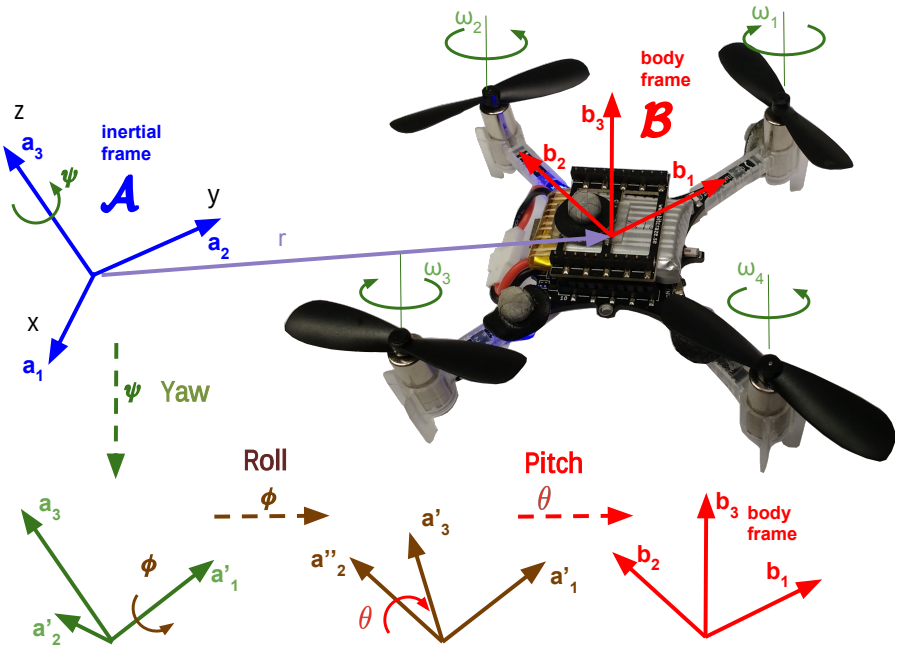


Figure 1.1: The CrazyFlie 2.0 robot that will be used in our course. An Euler Z-X-Y transformation takes the inertial frame $\mathscr{A}$ to the body-fixed frame $\mathscr{B}$. First, a rotation by yaw angle ψ around the $\mathbf{a_3}$ axis is performed, then a rotation by roll angle ϕ around the $\mathbf{a_1}$ axis, and finally a rotation by pitch angle θ around $\mathbf{a_2}$ axis. A translation $\mathbf{r}$ then produces the coordinate system $\mathscr{B}$, coinciding with the center of mass C of the robot and aligned along the arms.

2

Modeling

2.1. Coordinate Systems and Reference frames

The coordinate systems and free body diagram for the quadrotor are shown in Figure. 1.1. The inertial frame, $\mathscr{A}$, is defined by the triad $\mathbf{a}_1$, $\mathbf{a}_2$, and $\mathbf{a}_3$ pointing upward. The body frame, $\mathscr{B}$, is attached to the center of mass of the quadrotor with $\mathbf{b}_1$ coinciding with the preferred forward direction and $\mathbf{b}_3$ being perpendicular to the plane of the rotors pointing vertically up during perfect hover (see Figure. 1.1). These vectors are parallel to the principal axes.

More formally, the coordinate of a vector $\mathbf{x}$ that is expressed in $\mathscr{A}$ as $\mathbf{x} = \sum_i^{\mathscr{A}} x_i \mathbf{a}_i$ and in $\mathscr{B}$ as $\mathbf{x} = \sum_i^{\mathscr{B}} x_i \mathbf{b}_i$ are transformed into each other by the rotation matrix ${}^{\mathscr{A}}R_{\mathscr{B}}$ and translation vector ${}^{\mathscr{A}}\mathbf{T}_{\mathscr{B}}$:

$$ {}^{\mathscr{A}}\mathbf{x} = {}^{\mathscr{A}}R_{\mathscr{B}} + {}^{\mathscr{A}}\mathbf{T}_{\mathscr{B}} \tag{2.1} $$

To express the rotational motion of the moving frame $\mathscr{B}$, it is useful to introduce the angular velocity vector ω that describes how the basis vectors $\mathbf{b}_i$ move:

$$ \frac{d}{dt}\mathbf{b}_i = \omega \times \mathbf{b}_i \tag{2.2} $$

Note that this equation is coordinate free, meaning ω is not yet expressed explicitly in any particular coordinate system. We will denote the components of angular velocity of the robot in the body frame by p, q, and r:

$$ \omega = p\mathbf{b}_1 + q\mathbf{b}_2 + r\mathbf{b}_3. \tag{2.3} $$

The heading (yaw) angle of the robot plays a special role since we can choose it freely without directly affecting the robots dynamics. For this reason we use $Z-X-Y$ Euler angles to describe the transform from $\mathscr{A}$ to $\mathscr{B}$" first a yaw rotation by ψ around the $\mathbf{a}_3$ axis is performed, then a roll by ϕ around the $\mathbf{a}_1$ axis, and finally a pitch by θ around the $\mathbf{a}_2$ axis. From the Euler angles one can compute the rotation matrix as follows:

$$ {}^{\mathscr{A}}R_{\mathscr{B}} = \begin{bmatrix} \cos(\psi)\cos(\theta) - \sin(\phi)\sin(\psi)\sin(\theta) & -\cos\phi\sin(\psi) & \cos(\psi)\sin(\theta) + \cos(\theta)\sin(\phi)\sin(\psi) \\ \cos(\theta)\sin(\psi) + \cos(\psi)\sin(\phi)\sin(\theta) & \cos\phi\cos(\psi) & \sin(\psi)\sin(\theta) - \cos(\psi)\cos(\theta)\sin(\phi) \\ -\cos(\phi)\sin(\theta) & \sin(\phi) & \cos(\phi)\cos(\theta) \end{bmatrix} \tag{2.4} $$

The angular velocity and Euler angle velocities are related by:

$$ \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & -\cos(\phi)\sin(\theta) \\ 0 & 1 & \sin(\phi) \\ \sin(\theta) & 0 & \cos(\phi)\cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{2.5} $$

2.2. Motor Model

This section describes how we model the motors. Each rotor has an angular speed ω_i and produces a vertical force F_i according to

$$ F_i = k_F \omega_i^2. \tag{2.6} $$

Experimentation with a fixed rotor at steady-state shows that $k_F \approx 6.11 \times 10^{-8} N/rpm^2$. The rotors also produce a moment according to

$$M_i = k_M \omega_i^2. \tag{2.7}$$

The constant k_M, is determined to be about $1.5 \times 10^{-9} Nm/rpm^2$ by matching the performance of the simulation to the real system.

Data obtained from system identification experiments suggest that the rotor speed is related to the commanded speed by a first-order differential equation

2.3. Rigid Body Dynamics: Newton's Equations of Motion for the Center of Mass

We will describe the motion of the center of mass (CoM) in the inertial ("world") coordinate frame $\mathscr{A}$. This makes sense because we will want to specify our waypoints (where the robot should fly), trajectories and controller targets (where the robot should be at this moment) in the inertial frame.

Newton's equation of motion for the robot's CoM $\mathbf{r}$ is determined by the robot's mass m, the gravitational force $\mathbf{F}_g = mg$, and the sum of the motor's individual forces $\mathbf{F}_i$:

$$m\ddot{\mathbf{r}} = \mathbf{F}_g + \sum_i \mathbf{F}_i \tag{2.8}$$

In the coordinate of the inertial frame $\mathscr{A}$ this reads:

$$m\,{}^{\mathscr{A}}\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + {}^{\mathscr{A}}R_{\mathscr{B}} \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix} \tag{2.9}$$

where ${}^{\mathscr{A}}R_{\mathscr{B}}$ is the rotation matrix from Eq. (2.1). If the robot is tilted, ${}^{\mathscr{A}}R_{\mathscr{B}}$ (cf Eq. (2.4)), will mix the propeller forces into the x and y plane and so generate horizontal acceleration of the robot. In other words, we can accomplish movement in all three directions by manipulating the attitude of the vehicle and its thrust.

Eq. (2.9) further suggests that rather than using the forces F_i as control inputs, we should define our first input u_1 as the sum

$$u_1 = \sum_{i=1}^{4} F_i. \tag{2.10}$$

2.4. Euler's Equations of Motion for the Attitude

Since from Eq. (2.9) we know that attitude control is necessary to generate horizontal motion, in this section will examine how the motors affect the orientation of the vehicle.

In the inertial frame, the rate at which the robot's angular momentum $\mathbf{L} = I\omega$ changes is determined by the total moment $\mathbf{M}$ generated by the propellers:

$$\dot{\mathbf{L}} = \mathbf{M}. \tag{2.11}$$

While this equation looks clean and simple in the inertial frame, it is actually hard to work with because the initial tensor I changes with the attitude of the robot, and is thus time dependent and non diagonal. Fir control purposes, this equation is best expressed in the body frame that is aligned with the principal axis, where the inertia tensor I is constant, and diagonal. However, since now the basis vectors $\mathbf{b}_i$ are time-dependent, the equation for the time derivative of the angular momentum in the body frame becomes:

$$\begin{aligned} \dot{\mathbf{L}} &= \sum_i {}^{\mathscr{B}}\dot{L}_i \mathbf{b}_i + \sum_i {}^{\mathscr{B}}L_i \dot{\mathbf{b}}_i \\ &= \sum_i {}^{\mathscr{B}}\dot{L}_i \mathbf{b}_i + \sum_i \omega \times {}^{\mathscr{B}}L_i \mathbf{b}_i \\ &= \sum_i {}^{\mathscr{B}}\dot{L}_i \mathbf{b}_i + \omega \times \mathbf{L}. \end{aligned} \tag{2.12}$$

Combining Eq. (2.11) and 2.12, and using the fact that I is constant in $\mathscr{B}$ yields Euler's Equation

$${}^{\mathscr{B}}I\dot{\omega} = {}^{\mathscr{B}}\mathbf{M} - {}^{\mathscr{B}}\omega \times {}^{\mathscr{B}}I\omega, \tag{2.13}$$

where depending on context I means alternating a tensor or matrix expressed in the body frame.

How do the motors come into play? First, by generating a force F_i that is a distance l away from the CoM, each motor can exert a torque that is in the $\mathbf{b}_1$, $\mathbf{b}_2$ plane. In addition, each rotor produces a moment M_i perpendicular to the plane of rotation of the blade. Rotors 1 and 3 rotate clockwise in the $-\mathbf{b}_3$ direction while 2 and 4 rotate counter clockwise in the $\mathbf{b}_3$ direction. Since the moment produced on the quadrotor is opposite to the direction of rotation of the blades, M_1 and M_3 act in the $\mathbf{b}_3$ direction while M_2 and M_4 act in the $-\mathbf{b}_3$ direction. In contrast to this, this forces F_i are all in the positive $\mathbf{b}_3$ direction due to the fact that the pitch is reversed on two of the propellers, see Figure. 1.1.

Using this geometric intuition, and expressing the angular velocity in the body frame by p, q, and r as in Eq. (2.3), the Euler equation Eq. (2.13) becomes:

$$I\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix} l(F_2-F_4)\\ l(F_3-F_1)\\ M_1-M_2+M_3-M_4\end{bmatrix} - \begin{bmatrix}p\\ q\\ r\end{bmatrix} \times I \begin{bmatrix}p\\ q\\ r\end{bmatrix} \tag{2.14}$$

Note that the total net torque along the yaw ($\mathbf{b}_3$) axis of the robot is simply the signed sum of the motor's torques M_i.

We can rewrite this as:

$$I\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix} 0 & l & 0 & -l\\ -l & 0 & l & 0\\ \frac{k_M}{k_F} & -\frac{k_M}{k_F} & \frac{k_M}{k_F} & -\frac{k_M}{k_F}\end{bmatrix}\begin{bmatrix}F_1\\ F_2\\ F_3\\ F_4\end{bmatrix} - \begin{bmatrix}p\\ q\\ r\end{bmatrix} \times I \begin{bmatrix}p\\ q\\ r\end{bmatrix}. \tag{2.15}$$

where $\frac{k_M}{k_F}$ is the relationship between lift and drag given by Eqs. (2.6) and (2.7). Accordingly, we will define our second set of inputs to be the vector of moments $\mathbf{u}_2$ given by:

$$\mathbf{u}_2 = \begin{bmatrix} 0 & l & 0 & -l\\ -l & 0 & l & 0\\ \frac{k_M}{k_F} & -\frac{k_M}{k_F} & \frac{k_M}{k_F} & -\frac{k_M}{k_F}\end{bmatrix}\begin{bmatrix}F_1\\ F_2\\ F_3\\ F_4\end{bmatrix} \tag{2.16}$$

Dropping the reference frame superscripts for brevity, we can now write the equations of motion for center of mass and orientation in compact form:

$$m\ddot{\mathbf{r}} = \begin{bmatrix}0\\ 0\\ -mg\end{bmatrix} + R\begin{bmatrix}0\\ 0\\ u_1\end{bmatrix} \tag{2.17}$$

$$I\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \mathbf{u}_2 - \begin{bmatrix}p\\ q\\ r\end{bmatrix} \times I \begin{bmatrix}p\\ q\\ r\end{bmatrix}, \tag{2.18}$$

where Eqs. (2.17) and (2.18) are in inertial coordinates and body coordinates respectively. The inputs u_1 and $\mathbf{u}_2$ are related to the motor forces F_i via the linear system of equations formed by Eqs.(2.10) and (2.16). We are thus facing a system governed by two coupled second order differential equations (recall eq. (2.5)) which we will exploit later to design controllers.

3

Robot Controllers

3.1. Trajectory Generation

Our ultimate goal is to make the robot follow a trajectory. For the time let's assume we are given a trajectory generator that for any given time t produces a trajectory target $\mathbf{z}^{des}(t)$ consisting of target position $\mathbf{r}_T(t)$ and target yaw $\psi_T(t)$:

$$\mathbf{z}^{des}(t) = \begin{bmatrix} \mathbf{r}_T(t) \\ \psi_T^(t) \end{bmatrix} \tag{3.1}$$

and its first and second derivatives $\dot{\mathbf{z}}^{des}$ and $\ddot{\mathbf{z}}^{des}$. To hover for example, the trajectory generator would produce a constant $\mathbf{r}(t) = \mathbf{r}_0$ and e.g., a fixed $\psi(t) = \psi_0$, with all derivatives being zero.

The controller's job will then be to produce the correct torque $\mathbf{u}_2$ and thrust u_1 to bring the robot to the desired state specified by the trajectory generator.

3.2. Linear Backstepping Controller

For this controller we will make the following assumptions:

1. The robot is near the hover points, meaning the roll angle ϕ and pitch angle θ are small enough to allow linearization of trigonometric functions, i.e., $\cos(\theta) = 1, \sin(\phi) = \phi, \cos(\theta) = 1, \sin(\theta) = \theta$. This will allow use the linearize the rotation matrix in Eq. (2.17).

2. The robot's angular velocity is small enough for the cross product term between angular momentum and velocity in Eq. (2.18) to be negligible. This is usually a good assumption for almost any quadrotor (how about aggressive racing quadrotor?)

3. The attitude of the quadrotor can be controlled at a much smaller time scale than the position. This "back-stepping" approach to controller design allows a decoupling of position and attitude control loops. In practice this is generally warranted since the attitude controller usually runs almost an order of magnitude faster than the position controller.

Making the backstepping approximation is equivalent to assuming that R in Eq. 2.17 can be commanded instantaneously. This further implies that it is possible to directly command the acceleration $\ddot{\mathbf{r}}^{des}$. Fig. 3.1 shows how trajectory generator, position, and attitude controller play together.

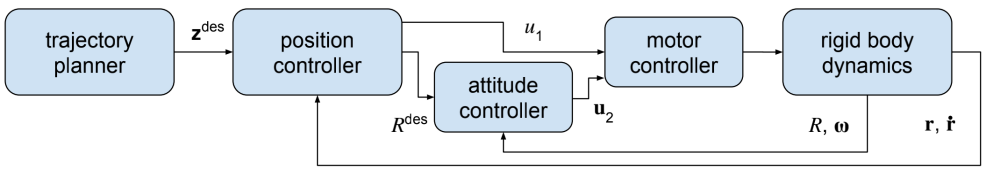


Figure 3.1: The position and attitude control loops.

Define the position error in terms of components by:

$$e_i = (r_i - r_{i,T}). \tag{3.2}$$

In order to guarantee that this error goes exponentially to zero, we require

$$(\ddot{r}_i^{des} - \ddot{r}_{i,T}) + k_{d,i}(\dot{r}_i - \dot{r}_{i,T}) + k_{p,i}(r_i - r_{i,T}) \tag{3.3}$$

In Eq. (3.3), $r_{i,T}$ and its derivative are given by the trajectory generator, and r_i and $\dot{r}_i$ are provided by the state estimation system[1], allowing for the commanded acceleration $\ddot{i}^{des}$ to be calculated:

$$\ddot{r}_i^{des} = \ddot{r}_{i,T} - k_{d,i}(\dot{r}_i - \dot{r}_{i,T}) - k_{p,i}(r_i - r_{i,T}). \tag{3.4}$$

Now the attitude of the quadrotor must be controlled such that it will generate $\ddot{r}_i^{des}$. For this, the linearized version of Eq. (2.17) is used: Maxwell's equations:

$$\ddot{r}_1^{des} = g(\theta^{des}\cos(\psi_T) + \phi^{des}\sin(\psi_T)) \tag{3.5a}$$

$$\ddot{r}_2^{des} = g(\theta^{des}\sin(\psi_T) - \phi^{des}\cos(\psi_T)) \tag{3.5b}$$

$$\ddot{r}_2^{des} = \frac{1}{m}u_1 - g \tag{3.5c}$$

From the third equation we can directly read off the thrust control u_1. The first two equations can be solved for θ^{des} and ϕ^{des}, since $\psi^{des} = \psi_T$ is given directly be the trajectory generator.

Linearizing Eq. (2.18), we get:

$$I\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \mathbf{u}_2. \tag{3.6}$$

and by further exploiting that via Eq. 2.5 Euler angle velocities are approximately equal to angular velocities due to linear approximation ($\theta \approx 0, \psi \approx 0, \phi \approx 0$):

$$I\begin{bmatrix}\ddot{\phi}\\ \ddot{\theta}\\ \ddot{\psi}\end{bmatrix} = \mathbf{u}_2. \tag{3.7}$$

Thus the "inner loop" attitude control can also be done with a simple PD controller:

$$\mathbf{u}_2 = I\begin{bmatrix}-k_{p,\phi}(\phi - \phi^{des}) - k_{d,\phi}(p - p^{des})\\ -k_{p,\theta}(\theta - \theta^{des}) - k_{d,\theta}(q - q^{des})\\ -k_{p,\psi}(\psi - \psi^{des}) - k_{d,\psi}(r - \dot{\psi}_T)\end{bmatrix}, \tag{3.8}$$

where the desired roll and pitch velocities p^{des} and q^{des} can be computed from the equations of motion and the specified trajectory, but in practice can be set to zero.

In summary, the controller then works as follows:

1. Use Eq. (3.4) to compute the commanded acceleration $\ddot{\mathbf{r}}^{des}$.
2. Use Eq. (3.5c) to compute u_1 and the desired angles θ^{des} and ϕ^{des} from Eqs. (3.5a) and (3.5b) respectively.
3. Use Eq. (3.8) to compute $\mathbf{u}_2$.

[1] At the moment, you can assume these states can be estimated/measured perfectly.

4

Robot Description

4.1. Quadrotor Platform

For this project, we will be using the CrazyFlie 2.0 platform made by Bitcraze, shown in Fig. 1. The CrazyFlie has a motor to motor (diagonal) distance of 92 mm, and a mass of 30 g, including a battery. A microcontroller allows low-level control and estimation to be done onboard the robot. An onboard IMU provides feedback of angular velocities and accelerations.

4.2. Software and Integration

Position control and other high-level commands are computed in Matlab at 100 Hz and sent to the robot via the CrazyRadio (2.4GHz). Attitude control is performed onboard using the microcontroller.

4.3. Inertial Properties

Since $\mathbf{b_i}$ are principal axes, the inertia matrix referenced to the center of mass along the $\mathbf{b_i}$ reference triad, I, is a diagonal matrix. In practice, the three moments of inertia can be estimated by weighing individual components of the quadrotor and building a physically accurate model in SolidWorks. The key parameters for the rigid body dynamics for the CrazyFlie platform are as follows:

1. mass $m = 0.030kg$;
2. the distance from the center of mass to the axis of a motor: $l = 0.046m$
3. the components of the inertia dyadic using $\mathbf{b_i}$ in kgm^2

$$[I_C] = \begin{bmatrix} 1.43 \times 10^{-5} & 0 & 0 \\ 0 & 1.43 \times 10^{-5} & 0 \\ 0 & 0 & 2.89 \times 10^{-5} \end{bmatrix}$$

You can find these parameters in `crazyflie.m`

4.3.1. Constraints

1. the maximum commanded angle is $40\pi/180$
2. the maximum force is $2.5mg$

5

Project Work

5.1. Simulator

The quadrotor simulator can be found under "Assignments" on the course Brightspace page. In "Assignment 4: Drones", it comes with skeleton code that has interface documentation. Before implementing your functions, you should first try running `runsim.m` in your Matlab. If you see a quadrotor falling from position $(0,0,0)$, the simulator works on your computer, and you may continue with other tasks. This is because the outputs of `controller.m` are all zeros. Thus no thrust is generated.

When you have the basic version of your trajectory generator and controller done (see below), you will be able to see the quadrotor flying in the space with proper roll, pitch, and yaw, leaving trails of desired and actual position behind. The desired position is color-coded blue and the actual position is red. After the simulation finishes, two plots of position and velocity will be generated to give you an overview of how well your trajectory generator and controller are doing. Note that you will not be graded on these plots but by the automatic grader.

5.2. Grading your assignment

5.2.1. Performance criteria

You grade will be evaluated by three criteria. Suppose you use a total time T to finish the tracking task.

1. Tracking performance: the desired position is color-coded blue and the actual position is red should be as close as possible with each other. Mathematically, this criteria can be written as the following cost function and you want to minimize the cost function

$$\min \sum_{t=0}^{T} \|\mathbf{r}(t) - \mathbf{r}(t)^{\text{des}}\|_2^2 = \sum_{t=0}^{T} \left((x(t) - x(t)^{\text{des}})^2 + (y(t) - y(t)^{\text{des}})^2 + (z(t) - z(t)^{\text{des}})^2 \right) \tag{5.1}$$

2. The total time spent to finish the task, i.e., you want to minimize T:

$$\min T \tag{5.2}$$

3. The total battery storage spent to finish the task. Most of the battery life was consumed by the motors while we assume that on sensors and microcomputers are negligible[1] and roughly equivalent to the sum of square of angular velocities of each rotor i over time t:

$$\min \sum_{i=1}^{4} \sum_{t=0}^{T} (\omega^i(t))^2 \tag{5.3}$$

5.2.2. Grading based on ranking in leader board

If you submitted code, output the three values in equations 5.1, 5.2 and 5.3, respectively. Your code will be automatically tested. Your ranking in each criterion will be published on a leader board. Your overall ranking will be averaged over your ranking in each criterion. The actual grade will be adjusted based on the overall grade distributions of previous assignments. Don't miss any point!

[1] If you run deep neural networks on a quadrotor, this may not be true. Why? How to solve this?

5.2.3. Bonus points

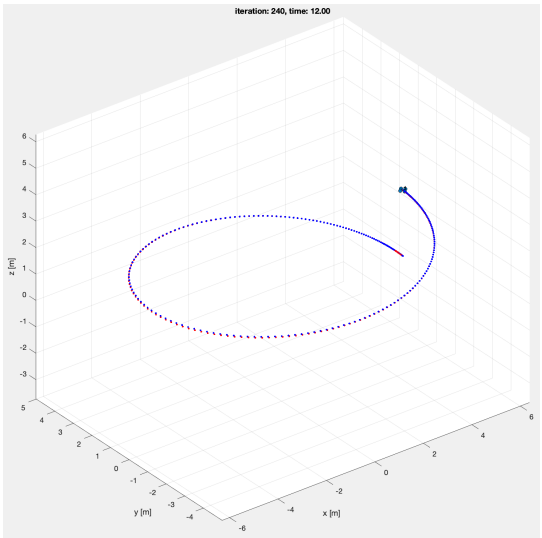
You overall rank might be affected by poor performance in some of the criteria. You have a chance to get some bonus to potentially rank higher.

5.3. Task 1 (40%): Tracking a trajectory in Matlab

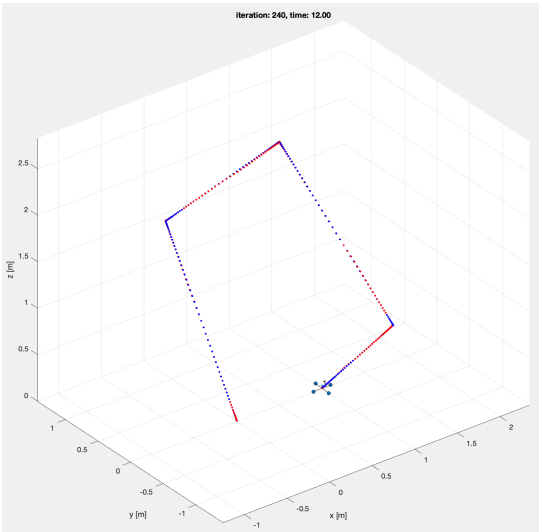
To warm-up, you will simulate the quadrotor dynamics and control using the Matlab simulator. The simulator relies on the numerical solver textttode45, details about this solver can be easily found online. You need to know the basics of how ode solvers and implement the solver in Python using Runge–Kutta fourth-order method[2], but it would be beneficial if you know the basics of how ode solvers work.

5.3.1. Trajectory Generator

You will first implement three trajectory generators to produce a hover, a circle, and a diamond trajectory. In order to specify the trajectory in the simulator, you will need to change the variable `trajhandleinrunsim.m`. The handle `trajhandle` is the name of a function that takes in current time t and current quadrotor number `qn` and returns a struct of desired state as a function of time. The struct `desired_state` has 5 fields, which are `pos`, `vel`, `acc`,`yaw` and `yawdot`. For a proper trajectory, you need to specify `pos`, `vel` and `acc`, whereas `yaw` and `yawdot` can be left as 0.



(a) Helix trajectory (`circle.m`)



(b) Diamond trajectory (`diamond.m`)

Figure 5.1: Trajectory Generator. Also check pre-recorded animations with a controller `circle.avi` and `diamond.avi` in the template code.

1. `hover.m`
   Hover at whatever altitude you wish. This is just a test trajectory for you to debug your controller.

2. `circle.m`
   A helix in the xy plane of radius $5m$ centered about the point $(0,0,0)$ starting at the point $(5,0,0)$. The z coordinate should start at 0 and end at 2.5. The helix should go counter-clockwise. See Fig.5.1a.

3. `diamond.m`
   A "diamond helix" with corners at $(0,0,0)$, $(0,\sqrt{2},\sqrt{2})$, $(0,0,2\sqrt{2})$, and $(0,-\sqrt{2},\sqrt{2})$ when projected into the yz plane, and an x coordinate starting at 0 and ending at 1. The quadrotor should start at $(0,0,0)$ and end at $(1,0,0)$. See Fig.5.1b.

Remark 1 *There are some clever ways to design a trajectory. However, you may not know them now. DR Javier Alonso-Mora will tell more in the course "RO47005 Planning & Decision Making" in Q2. If you have already known some of the planning algorithms, do use them here.*

[2] `https://en.wikipedia.org/wiki/Runge-Kutta_methods#Derivation_of_the_Runge-Kutta_fourth-order_method`

Remark 2 *In the folder* `trajectories`*, we implement a naive motion strategy* `tj_from_line.m`*. You can check* `circle.m` *to see how we design the desired* `pos`*,* `vel`*,* `acc`*,*`yaw` *and* `yawdot`*. In line 21 of* `circle.m`*, we fix the total time* T*. This is the total time we expect to finish the task. You can't set it arbitrarily small. For example, even the quadrotor flies at a speed of light, and it still needs some time to finish. Do you have any idea to make* T *as small as possible?*

5.3.2. Controller

You will then implement a controller in file `controller.m` that makes sure that your quadrotor follows the desired trajectory. The controller takes in a cell struct `qd`, current time `t`, current time t and quadrotor parameters `params`, and outputs thrust `F`, moment `M`, desired thrust, roll, pitch, yaw `trpy` and derivatives of desired roll, pitch, yaw `drpy`.

5.3.3. Simulation

Lastly, you need to fill in the appropriate variables for `trajhandle` and `controlhandle` in the script `runsim.m` for simulating your result.

5.4. Task 2 (50%)

5.4.1. Implement in Python and OpenAI Gym

Re-write the code in Python and visualize your tracking performance.

Hint: You can start from CartPole-v1[3] then download and modify the simulator code [4].

5.4.2. Track "TUD"

Track the letters "TUD" in `yz` plane in Figure. 5.2. The coordinate of these letters can be found in `tud.m`.

1. The drone can start from any position in the plane. Choose your initial position carefully.
2. specifies your trajectory generator.
3. You will inevitably fly in the white space. Please carefully design your transition strategy between letters. Be aware of the time and energy.

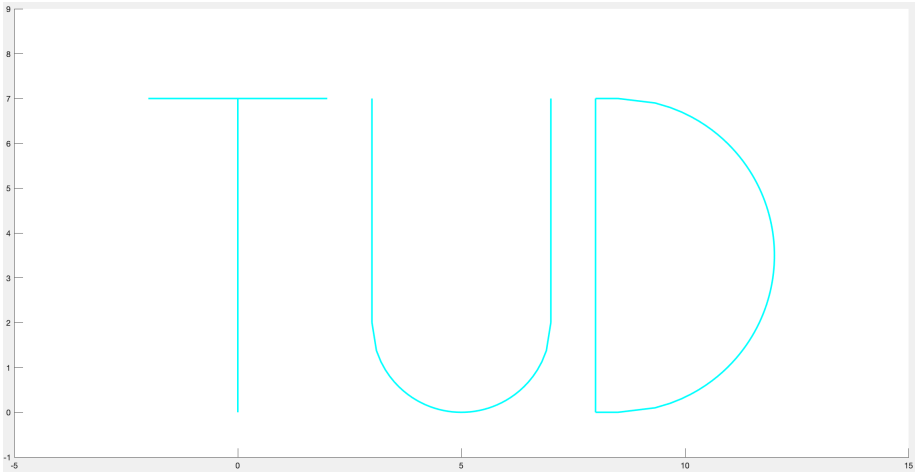


Figure 5.2: Track center line of TUD logo outdoors.

[3] `https://gym.openai.com/envs/#classic_control`
[4] `https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py`

5.5. Task 3 (bonus 10%)

Suppose now you have a chance to fly outdoors. You will have an unlimited budget to choose any accessories[5] for your Crazyflie. What will you do to achieve the following goals?

1. We assume any state estimator is not needed.
2. What sensors will you choose and why?
3. Similar to the setups in Task 2, the quadrotor is facing the logo vertically. The goal is to track the centerline of the letters "TUD".
4. The distance of quadrotor to yz plane should keep $2m$, i.e., $x = 1, \forall t$.

Figure 5.3: Track center line of TUD logo outdoors.

[5] https://store.bitcraze.io/collections/accessories

Bibliography

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[2] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. Robotics: Modelling, planning and control. *Advanced Textbooks in Control and Signal Processing. Springer,*, 2009.