Dartmouth COSC 89.18/189.02 Course Notes 6:
Quadrotors: simulation and control

November 2019

1 Rigid-body dynamics

The world and local frames We first establish the world and local frames: our world frame is a right-handed coordinate system with the x axis pointing to the north direction, the y axis pointing to the east direction, and the z axis pointing down. The local frame is a right-handed coordinate system rigidly attached to the quadrotor: the origin of the local frame is the center of mass of the quadrotor, the x axis points along the nose of the quadrotor, the y axis points to the right side of the quadrotor, and the z axis points down. We further assume that the two frames overlap at the beginning of the simulation.

The state of the quadrotor At time t, the state of our quadrotor is fully determined by the position and orientation of our local frame (and its first-order time derivatives). Let $\mathbf{p} \in \mathbb{R}^3$ be the position of the center of mass in the world frame. Let $\mathbf{R} \in \mathbb{R}^{3\times3}$ be the rotational matrix from the local frame to the world frame. In other words, if $\mathbf{v} \in \mathbb{R}^3$ represents a vector in the local frame, then $\mathbf{Rv}$ represents the same vector in the world frame.

While $\mathbf{R}$ has 9 elements, it only has three degrees of freedom because it is a rotational matrix. Typically, $\mathbf{R}$ is defined by three *Euler angles* ϕ (roll), θ (pitch), and ψ (yaw), which represents the rotation along x, y, and z axes of the local frame. Figure 1 explains how the rotational matrix is defined from three Euler angles. In this figure, the green frame on the left is the world frame, and the red frame on the right shows the final local frame.

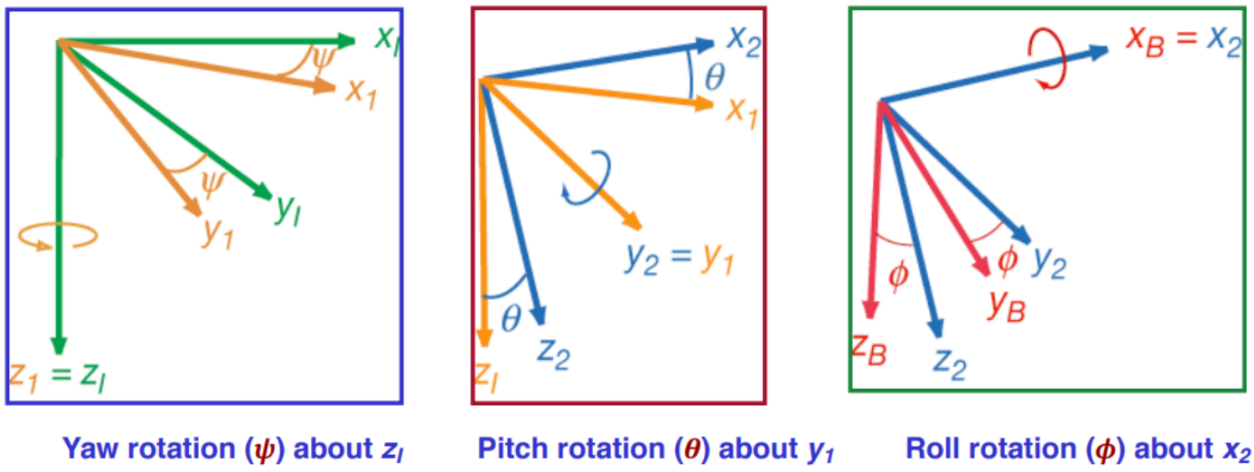

Figure 1: Euler angles. Picture credit: `http://www.stengel.mycpanel.princeton.edu/MAE331Lecture10.pdf`

The actuators A quadrotor has four rotors. Each rotor spins its propeller to generate thrust and the induced torque. The order and spinning direction of our quadrotor is depicted in Figure 2. Let $l \in \mathbb{R}^+$ be the arm length (the distance from the rotor to the center of mass). We can define the position of four rotors in the local frame as

follows:

$$\begin{aligned}\mathbf{d}_1 =&[\frac{l}{\sqrt{2}}, -\frac{l}{\sqrt{2}}, 0]\\ \mathbf{d}_2 =&[\frac{l}{\sqrt{2}}, \frac{l}{\sqrt{2}}, 0]\\ \mathbf{d}_3 =&[-\frac{l}{\sqrt{2}}, \frac{l}{\sqrt{2}}, 0]\\ \mathbf{d}_4 =&[-\frac{l}{\sqrt{2}}, -\frac{l}{\sqrt{2}}, 0]\end{aligned} \tag{1}$$

The thrust generated by rotor i is defined as $\mathbf{T}_i = [0, 0, -T_i] \in \mathbb{R}^3$. Note that there is a negative sign before T_i because I use $T_i > 0$ to represent the magnitude of thrust but our z axis points down.

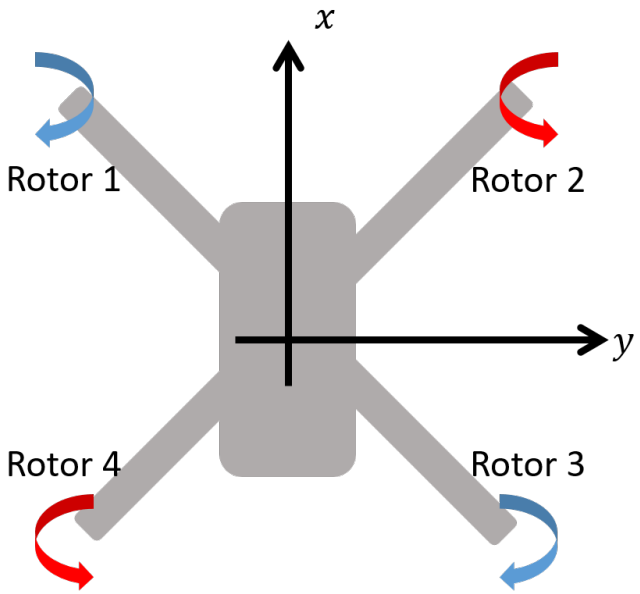

Figure 2: The four rotors and their spinning directions from a top-down view of our quadrotor.

The governing equations The linear and angular motions of the quadrotor is characterized by the time derivatives of its position and rotational matrix, i.e., $\dot{\mathbf{p}}$ and $\dot{\mathbf{R}}$. The linear motion is governed by the Newton's second law:

$$m\ddot{\mathbf{p}} = m\mathbf{g} + \mathbf{R}\sum_i \mathbf{T}_i \tag{2}$$

where m represents the mass and $\mathbf{g} = [0, 0, 9.81]$ is the gravitational acceleration.

The angular motion is determined by the Euler's equation:

$$\mathbf{I}\dot{\omega}_B + \omega_B \times (\mathbf{I}\omega_B) = \sum_i \mathbf{d}_i \times \mathbf{T}_i + \lambda_i \mathbf{T}_i \tag{3}$$

We use ω and ω_B to represent the angular rate in the world and local frames, which are connected to the rotation matrix by $\mathbf{R}\omega_B = \omega$ and $\omega \times \mathbf{R} = \dot{\mathbf{R}}$. $\mathbf{I} \in \mathbb{R}^{3\times3}$ is the inertial matrix in the local frame, which remains constant throughout the simulation. $\lambda_i \mathbf{T}_i$ captures the *spinning torque* induced from each rotor: if a rotor spins clockwise, it will cause the body to spin counterclockwise. $|\lambda_i|$ is the ratio of the magnitudes of the torque and the thrust, which is fully determined by the rotor itself. Since our four rotors are the same, we can use a single $\lambda = |\lambda_i|, \forall i$. In our current rotor setup, $\lambda_1 = \lambda_3 = \lambda$, and $\lambda_2 = \lambda_4 = -\lambda$.

2 Sensor

Our quadrotor is equipped with a gyroscope, an accelerometer, a barometer, a sonar, and a camera. All of these sensors work together to fully determine the current statues of the quadrotor. For example, to infer the altitude,

we retrieve data from the sonar (the distance to the ground) and the barometer (the surrounding air pressure), and fuse them to gain a more accurate estimation of the altitude. We will skip the sensor fusion process and assume a clean estimate of linear and angular motion has been given, which includes the following: the position of the center of mass $\mathbf{p}$, the velocity of the center of mass $\mathbf{v}$ represented as a vector in the *local* frame (in other words, $\mathbf{R}\mathbf{v} = \dot{\mathbf{p}}$), the Euler angles ϕ, θ, ψ, and the Euler angular rate $\dot{\phi}, \dot{\theta}, \dot{\psi}$.

3 Controller

Now we will learn how to design a position controller that allows the quadrotor to track an arbitrary location in 3D: given any target position $(x, y, z) \in \mathbb{R}^3$, this controller will instruct the quadrotor to fly to that location and stay level as quickly as possible. To help you better understand how it works, we will use a concrete target position $\mathbf{p} = (0, 0, -1.5)$ in the following sections. Note that we implicitly require the quadrotor to hover at this position in a steady state, i.e., $\dot{\mathbf{p}} = (0, 0, 0), \phi = \theta = \psi = 0.0$, and $\dot{\phi} = \dot{\theta} = \dot{\psi} = 0$. We will walk you through a hierarchical controller design in three steps: 1) design an altitude controller to maintain its height, 2) design an attitude controller to stay level in the air, and 3) design an horizontal controller to cancel out any horizontal offsets.

The altitude controller Let's assume that the attitude of our quadrotor is always level ($\phi = \theta = \psi = 0$) and focus on how to ascend/descend to the desired altitude for now. Let $z_{ref} = -1.5$ be our desired height (unit: meter). Let z be the detected altitude from the sensors, which will be provided in both simulation and on the hardware platform. Keep in mind that both z_{ref} and z are negative due to the fact that the z axis points down. We use a PD controller to determine the desired net thrusts from all the four rotors. To derive the correct form, let's first introduce $h = -z$ and $h_{ref} = -z_{ref}$ to represent all the altitude as a positive number, which will make the derivation more intuitive:

$$\begin{aligned} T_z =& mg + P_z(h_{ref} - h) + D_z(\dot{h}_{ref} - \dot{h}) \\ =& mg + P_z(h_{ref} - h) - D_z\dot{h} \end{aligned} \tag{4}$$

Here $P_z > 0$ and $D_z > 0$. As a sanity check, imagine we want the quadrotor to climb, i.e., $h_{ref} - h > 0$, which predicts a T greater than the gravity mg and lift the quadrotor. Now we can replace h with z:

$$T_z = mg - P_z(z_{ref} - z) + D_z\dot{z} \tag{5}$$

This describes how much *net thrust* is needed, but how do we distribute it over four rotors? The answer is to ask each rotor to provide $T_z/4$ thrust. If you revisit the governing equation, you will understand this distribution has the benefit of decoupling altitude control from attitude control: since all four rotors always provide equal thrust, whenever we adjust the altitude, it won't cause extra rotations in any of the roll, pitch, or yaw directions. Now if you abstract our quadrotor as a mass point and ignore its orientation, this mass point can be controlled to climb/descend to desired height.

The attitude controller Let's start with the roll controller: given a desired ϕ_{ref}, we want to predict what changes to the four thrusts are needed to set $\phi = \phi_{ref}$ as fast as possible. The trick is to generate torques along the local x axis without disturbing the rotation along the other two axes and the linear motion, and here is how: consider adding δT_ϕ to T_1, T_3 and subtracting δT_ϕ from T_2, T_4. The net thrust remains unchanged, so it won't affect the linear motion. We argue this change has the desired effects on the angular motion because 1) it induces a torque $2\sqrt{2}l\delta T_\phi$ along x to change ϕ and 2) the induced torque has zero value along the y and z axes, so it won't affect the motion of pitch and yaw. The actual value of δT_ϕ can be determined by a PID controller on the error of roll ($\phi_{ref} - \phi$). We will skip the details of its implementation for simplicity.

Similarly, the pitch controller can be realized by a PID controller on the pitch error and adding the predicted δT_θ to T_1, T_2 and subtracting δT_θ from T_3, T_4. The yaw controller is implemented by adding δT_ψ to T_2, T_4 and subtracting it from T_1, T_3.

We now summarize the expressions of each T_i below:

$$\begin{aligned} T_1 =& \frac{1}{4}T_z + \delta T_\phi + \delta T_\theta - \delta T_\psi \\ T_2 =& \frac{1}{4}T_z - \delta T_\phi + \delta T_\theta + \delta T_\psi \\ T_3 =& \frac{1}{4}T_z - \delta T_\phi - \delta T_\theta - \delta T_\psi \\ T_4 =& \frac{1}{4}T_z + \delta T_\phi - \delta T_\theta + \delta T_\psi \end{aligned} \tag{6}$$

We further connect the induced torque to the change of thrust:

$$\begin{aligned} \tau_\phi =& 2\sqrt{2}l\delta T_\phi \\ \tau_\theta =& 2\sqrt{2}l\delta T_\theta \\ \tau_\psi =& 4\lambda\delta T_\psi \end{aligned} \tag{7}$$

In this assignment, we first use a PID controller to predict the desired torque, then translate them into the corresponding δT and send it to the simulator. As a concrete example, τ_ψ is defined by:

$$\tau_\psi = P_{yaw}(\psi_{ref} - \psi) - D_{yaw}\dot{\psi} \tag{8}$$

The roll and pitch controllers are defined similarly but with an additional integral controller.

We reiterate that this controller design decouples the control of height, roll, pitch, and yaw in a clean way in the sense that updating one controller signal does not affect the other three. This simple but powerful controller design is one of the core reasons why quadrotors are more popular than other types of aerial robots these days.

The horizontal controller Let's revisit our original task: by setting $z_{ref} = -1.5$ and $\phi = \theta = \psi = 0$, our controller should be able to level itself and maintain its height. Everything works fine except that the horizontal location is still uncontrolled: imagine we perturb the quadrotor by pushing it sideways a little bit, it will then gain a horizontal velocity and drift away but still stay level and maintain the altitude. As a result, the existing altitude and attitude controllers won't be able to correct itself from drifting.

To resolve this issue, we need to reuse our attitude controller in a more creative way: assuming we want to reset the location of quadrotor to somewhere in front of it, we can assign $\theta_{ref} < 0$ so that the quadrotor leans forward. This will cause the quadrotor to advance. We can gradually decrease θ_{ref} to 0 as the quadrotor gets closer to the target. In practice, θ_{ref} is usually implemented by a PD controller on the x error in the local frame. Similarly, the y error is typically corrected by setting ϕ_{ref} from a PD controller. The two PD controllers usually have the same gains (with different signs) because of the symmetric design of the quadrotor.

There is one last caveat before we finalize this idea: recall that the local and world frames may not align during the flight (e.g., after we change its heading), we need to first convert the position error from the world frame to the local frame. Since the quadrotor almost always stays level, we will simply spin the position error vector along the z axis by yaw only and gently ignore the rotations from nonzero pitch and roll. The rotational matrix is given below:

$$\mathbf{R}_\psi = \begin{bmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{bmatrix}. \tag{9}$$

You can verify that $\mathbf{R}_\psi$ indeed converts the xy components of a vector in the world frame to the xy coordinates in the local frame, assuming we ignore the roll and pitch rotations.

Now let $\mathbf{p}_{xy} \in \mathbb{R}^2$ be the offset vector pointing from quadrotor to the target horizontal position in the world frame. In other words, $\mathbf{p}_{xy}$ is the drift we want to correct. We now explain the work flow of the x controller in detail:

- Convert $\mathbf{p}_{xy}$ into the local frame and get the error along the local x axis: $x_{err} = (\mathbf{R}_\psi \mathbf{p}_{xy})$.
- Use $-v_x = -(\mathbf{v})_x$ to approximate $\dot{x}_{err}$. Recall that $\mathbf{v}$ is the linear velocity in the local frame.
- Design a PD controller to estimate the desired θ_{ref}:

$$\theta_{ref} = -P_{xy}x_{err} - D_{xy}\dot{x}_{err} \approx -P_{xy}x_{err} + D_{xy}v_x \tag{10}$$

Here P_{xy} and D_{xy} are positive gains shared by the x and y PD controllers. Note that we use negative signs before P_{xy} and D_{xy} because we need negative θ to decrease x_{err}. The derivation of the y controller is similar, and we leave it as an exercise.

3.1 The complete loop

Given all the information above, we now summarize what happens at each time step in simulation below:

- At the beginning of each time step, retrieve the state $\mathbf{p}$, $\dot{\mathbf{p}}$, $\mathbf{R}$, and ω. These variables are only used to simulate the rigid body of the quadrotor and provide data to the sensors. In particular, our controller should never access them.
- Simulate the sensors by computing $\mathbf{v}, \phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}$. These variables along with $\mathbf{p}$ will be fed into the controller and get the predicted thrust $\mathbf{T}_i$ for each rotor.
- Use $\mathbf{T}_i$ to compute $\ddot{\mathbf{p}}$ and $\dot{\omega}_B$ by following Equation (2) and (3).
- Use $\ddot{\mathbf{p}}$ and forward Euler time integration to update $\mathbf{p}$ and $\dot{\mathbf{p}}$:

$$\begin{aligned} \mathbf{p} =& \mathbf{p} + \Delta t \dot{\mathbf{p}} \\ \dot{\mathbf{p}} =& \dot{\mathbf{p}} + \Delta t \ddot{\mathbf{p}} \end{aligned} \tag{11}$$

- Use $\dot{\omega}_B$ and forward Euler time integration to update $\mathbf{R}$ and ω:

$$\begin{aligned} \mathbf{R} =& \texttt{AxisAngle}(\omega \Delta t)\mathbf{R} \\ \omega_B =& \omega_B + \Delta t \dot{\omega}_B \\ \omega =& \mathbf{R}\omega_B \end{aligned} \tag{12}$$

  where `AxisAngle(n)` represents a rotation matrix which spins along the direction of $\mathbf{n}$ by an angle of $|\mathbf{n}|$.