MEAM 620 Project 1 Phase 1

Modeling and Control of a Quadrotor

January 19, 2022

1 Introduction

In this project you will put your knowledge of quadrotor dynamics to work by developing algorithms for control. A good controller is the backbone of any quadrotor autonomy stack and thus it is crucial that you understand the theory and implementation in this project. You will be reusing the code throughout this course. All code written in this project will be in Python. Refer to the instructions on Piazza for more information about which version to use and getting set up.

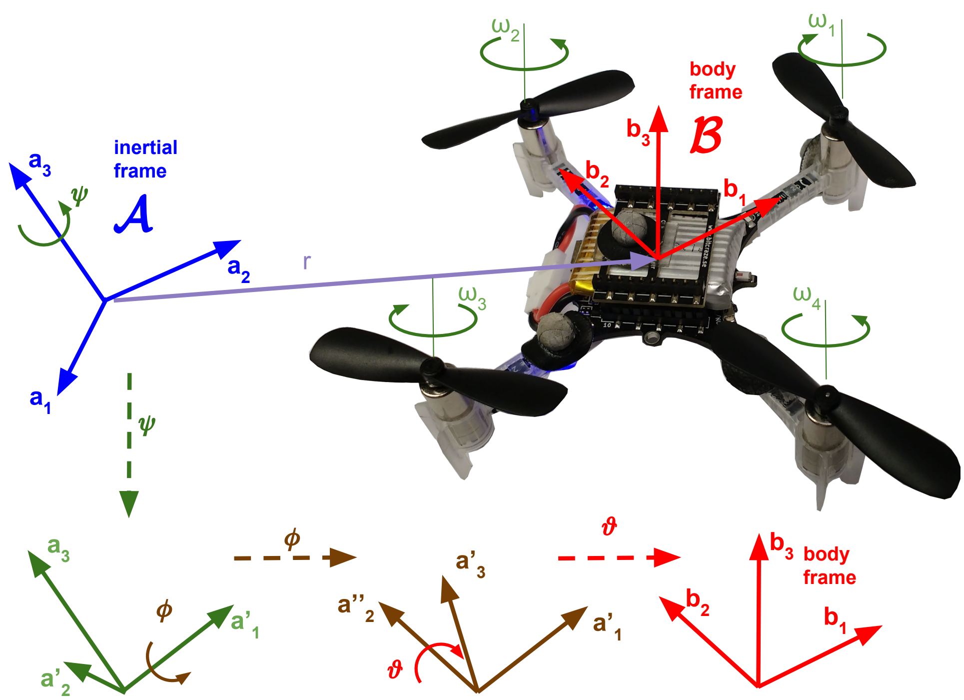

Figure 1: The CrazyFlie 2.0 robot that will be used for our exercises. An Euler Z-X-Y transformation takes the inertial frame $\mathcal{A}$ to the body-fixed frame $\mathcal{B}$. First a rotation by ψ around the $\mathbf{a_3}$ axis is performed, then a roll by ϕ around the (rotated!) $\mathbf{a_1}$ axis, and finally a pitch by θ around the (now twice rotated!) $\mathbf{a_2}$ axis. A translation $\mathbf{r}$ then produces the coordinate system $\mathcal{B}$, 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 Fig. 1. The inertial frame, $\mathcal{A}$, is defined by the triad $\mathbf{a}_1$, $\mathbf{a}_2$, and $\mathbf{a}_3$ with $\mathbf{a}_3$ pointing upward. The body frame, $\mathcal{B}$, 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 Fig. 1). These vectors are parallel to the principal axes. More formally, the coordinates of a vector $\mathbf{x}$ that is expressed in $\mathcal{A}$ as $\mathbf{x} = \sum_i {}^{\mathcal{A}}x_i \mathbf{a}_i$ and in $\mathcal{B}$ as $\mathbf{x} = \sum_i {}^{\mathcal{B}}x_i \mathbf{b}_i$ are transformed into each other by the rotation matrix ${}^{\mathcal{A}}R_{\mathcal{B}}$ and translation vector ${}^{\mathcal{A}}\mathbf{T}_{\mathcal{B}}$:

$$
{}^{\mathcal{A}}\mathbf{x} = {}^{\mathcal{A}}R_{\mathcal{B}}\,{}^{\mathcal{B}}\mathbf{x} + {}^{\mathcal{A}}\mathbf{T}_{\mathcal{B}} \ . \tag{1}
$$

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

$$
\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{b}_i = \boldsymbol{\omega} \times \mathbf{b_i} \ . \tag{2}
$$

Note that this equation is coordinate free, meaning $\boldsymbol{\omega}$ 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:

$$
\boldsymbol{\omega} = p\mathbf{b}_1 + q\mathbf{b}_2 + r\mathbf{b}_3. \tag{3}
$$

The heading (yaw) angle of the robot plays a special role since we can choose it freely without directly affecting the robot's dynamics. For this reason we use $Z - X - Y$ Euler angles to describe the transform from $\mathcal{A}$ to $\mathcal{B}$: first a yaw rotation by ψ around the $\mathbf{a}_3$ axis is performed, then a roll by ϕ around the (rotated!) $\mathbf{a}_1$ axis, and finally a pitch by θ around the (now twice rotated!) $\mathbf{a}_2$ axis [1] From the Euler angles one can compute the rotation matrix as follows:

$$
{}^{\mathcal{A}}R_{\mathcal{B}} = \begin{bmatrix} \cos(\psi)\cos(\theta) - \sin(\phi)\sin(\psi)\sin(\theta) & -\cos(\phi)\sin(\psi) & \cos(\psi)\sin(\theta) + \cos(\theta)\sin(\phi)\sin(\psi) \\ \cos(\theta)\sin(\psi) + \cos(\psi)\sin(\phi)\sin(\theta) & \cos(\phi)\cos(\psi) & \sin(\psi)\sin(\theta) - \cos(\psi)\cos(\theta)\sin(\phi) \\ -\cos(\phi)\sin(\theta) & \sin(\phi) & \cos(\phi)\cos(\theta) \end{bmatrix}. \tag{4}
$$

The angular velocity and Euler angle velocities are related by:

$$
\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & -\cos(\phi)\sin(\theta) \\ 0 & 1 & \sin(\phi) \\ \sin(\theta) & 0 & \cos(\phi)\cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{5}
$$

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:

$$
F_i = k_F \omega_i^2. \tag{6}
$$

[1] In aerospace engineering, the direction of rotation for yaw and pitch are opposite to our definition, which follows the conventional right-hand rule.

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

$$M_i = k_M \omega_i^2. \tag{7}$$

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

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

$$\dot{\omega}_i = k_m(\omega_i^{\text{des}} - \omega_i).$$

This motor gain, k_m, is found to be about $20\,\mathrm{s}^{-1}$ by matching the performance of the simulation to the real system. The desired angular velocities, ω_i^{des}, are limited to a minimum and maximum value determined through experimentation.

However, as a first approximation, we can assume the motor controllers to be perfect and the time constant k_m associated with the motor response to be arbitrarily small. In other words, we can assume the actual motor velocities ω_i are equal to the commanded motor velocities, ω_i^{des}.

Notes:

1. In simulation, you will be able to directly command torque and thrust. The thrust limit is available as a parameter inside the simulator, although you don't need it for this project.

2. In the lab, we will provide the code to translate torque and thrust to motor commands. You will however encounter saturation of your motors and may have to tweak your gains.

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 $\mathcal{A}$. 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 $\mathbf{F}_{\mathrm{g}} = m\mathbf{g}$, and the sum of the motor's individual forces $\mathbf{F}_{\mathrm{i}}$:

$$m\ddot{\mathbf{r}} = \mathbf{F}_{\mathrm{g}} + \sum_i \mathbf{F}_{\mathrm{i}}\ . \tag{8}$$

In the coordinates of the inertial frame $\mathcal{A}$ this reads:

$$m\,{}^{\mathcal{A}}\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + {}^{\mathcal{A}}R_{\mathcal{B}} \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix}. \tag{9}$$

where ${}^{\mathcal{A}}R_{\mathcal{B}}$ is the rotation matrix used in Eq (1) constructed via Eq (4). If the robot is tilted, the above equation 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.

Equation (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:

$$u_1 = \sum_{i=1}^{4} F_i\ . \tag{10}$$

2.4 Euler's Equations of Motion for the Attitude

Since from Eq (9) we know that attitude control is necessary to generate horizontal motion, in this section we will examine how the motors affect the orientation of the vehicle.

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

$$\dot{\mathbf{L}} = \mathbf{M} \,. \tag{11}$$

While this equation looks clean and simple in the inertial frame, it is actually hard to work with because the inertial tensor I changes with the attitude of the robot, and is thus time dependent and non diagonal! For 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:

$$\dot{\mathbf{L}} = \sum_i {}^{\mathcal{B}}\dot{L}_i \mathbf{b}_i + \sum_i {}^{\mathcal{B}}L_i \dot{\mathbf{b}}_i = \sum_i {}^{\mathcal{B}}\dot{L}_i \mathbf{b}_i + \sum_i \boldsymbol{\omega} \times {}^{\mathcal{B}}L_i \mathbf{b_i} = \sum_i {}^{\mathcal{B}}\dot{L}_i \mathbf{b}_i + \boldsymbol{\omega} \times \mathbf{L} \,. \tag{12}$$

Combining (11) and (12), and using the fact that I is constant in $\mathcal{B}$ yield's Euler's Equation:

$$I\, {}^{\mathcal{B}}\dot{\boldsymbol{\omega}} = {}^{\mathcal{B}}\mathbf{M} - {}^{\mathcal{B}}\boldsymbol{\omega} \times I\, {}^{\mathcal{B}}\boldsymbol{\omega} \,, \tag{13}$$

where depending on context I means alternatingly a tensor or a 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, the 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 Fig 1.

Using this geometric intuition, and expressing the angular velocity in the body frame by p, q, and r as in Eq (3), the Euler equation (13) becomes:

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} l(F_2 - F_4) \\ l(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix} . \tag{14}$$

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 (why does the distance l to the center of the robot play no role?).

We can rewrite this as:

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} 0 & l & 0 & -l \\ -l & 0 & l & 0 \\ \gamma & -\gamma & \gamma & -\gamma \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix} . \tag{15}$$

where $\gamma = \frac{k_M}{k_F}$ is the relationship between lift and drag given by Equations (6-7). Accordingly, we will define our second set of inputs to be the vector of moments $\mathbf{u}_2$ given by:

$$\mathbf{u}_2 = \begin{bmatrix} 0 & l & 0 & -l \\ -l & 0 & l & 0 \\ \gamma & -\gamma & \gamma & -\gamma \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} . \tag{16}$$

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

$$m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ u_1 \end{bmatrix} \tag{17}$$

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \mathbf{u}_2 - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix} , \tag{18}$$

where (17) is in inertial coordinates, and (18) is in body coordinates. The inputs u_1 and $\mathbf{u}_2$ are related to the motor forces F_i via the linear system of equations formed by (10) and (16). We are thus facing a system governed by two coupled second order differential equations which we will exploit in section 3 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 $\mathbf{z}_T(t)$ consisting of target position $\mathbf{r}_T(t)$ and target yaw $\psi_T(t)$:

$$\mathbf{z}_T(t) = \begin{bmatrix} \mathbf{r}_T(t) \\ \psi_T(t) \end{bmatrix} \tag{19}$$

and its first and second derivatives $\dot{\mathbf{z}}_T$ and $\ddot{\mathbf{z}}_T$. 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 Constant Speed Trajectory

There are many different methods for creating a trajectory function that passes through a given set of waypoint locations. The focus of this project is on the controller and not the trajectory, so a simple constant speed trajectory will suffice and is recommended. You are allowed to use any trajectory of your choice.

The constant speed trajectory assumes that the robot is moving at constant speed. This trajectory has the unfortunate downside of commanding infinite accelerations at the waypoints.

3.2.1 Initialization

The initialization procedure assumes that the designer has specified a speed and the code has access to the set of waypoints. Let $\{\mathbf{p}_1, ..., \mathbf{p}_n\}$ be the collection of n waypoints and v be the speed. The resulting trajectory will have $n-1$ segments.

We start by computing the $n-1$ unit vectors describing the direction of travel for each segment,

$$\hat{\mathbf{l}}_i = \frac{\mathbf{p}_{i+1} - \mathbf{p}_i}{||\mathbf{p}_{i+1} - \mathbf{p}_i||}. \tag{20}$$

The desired velocity for each segment will be $\dot{\mathbf{r}}_{T,i} = v\hat{\mathbf{l}}_i$.

Next calculate the distance of each segment,

$$d_i = ||\mathbf{p}_{i+1} - \mathbf{p}_i||. \tag{21}$$

The duration of each segment will be $T_i = d_i/v$. The start time of each segment will be the sum of the durations of the previous segments.

$$t_{\text{start},i} = \sum_{j=1}^{i-1} T_j. \tag{22}$$

3.2.2 Evaluating

The first step when evaluating the trajectory will be to figure out which segment the robot is in by checking which two segments start times you are between. Lets assume the robot is in segment i.

The yaw, yaw rate, acceleration, jerk, and snap will be 0. The velocity will be

$$\dot{\mathbf{r}}_T(t) = v\hat{\mathbf{l}}_i \tag{23}$$

and the position will be

$$\mathbf{r}_T(t) = \mathbf{p}_i + v\hat{\mathbf{l}}_i(t - t_{\text{start},i}) \tag{24}$$

If the time is greater than the duration of the full trajectory, then the velocity should be zero and the position should be the final waypoint. It is important for the quadrotor to stay in hover after reaching the destination.

3.3 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. $\cos(\phi) = 1$, $\sin(\phi) = \phi$, $\cos(\theta) = 1$, $\sin(\theta) = \theta$. This will allow us to linearize the rotation matrix in Eq (17).

2. The robot's angular velocity is small enough for the cross product term between angular momentum and velocity in (18) to be negligible. This is usually a good assumption for almost any quadrotor.

3. The attitude of the quadrotor can be controlled at a much smaller time scale than the position. This "backstepping" 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 (17) can be commanded instantaneously. This further implies that it is possible to directly command the acceleration $\mathbf{r}^{\text{des}}$. Figure 2 shows how trajectory generator, position, and attitude controller play together.

Define the position error in terms of components by:

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

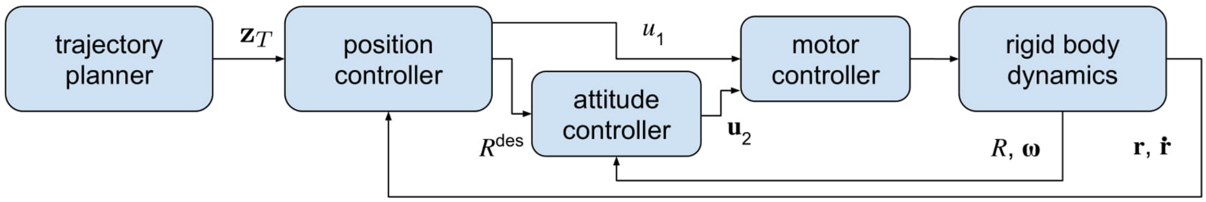

Figure 2: The position and attitude control loops.

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

$$(\ddot{r}_i^{\text{des}} - \ddot{r}_{i,T}) + k_{d,i}(\dot{r}_i - \dot{r}_{i,T}) + k_{p,i}(r_i - r_{i,T}) = 0 \ . \tag{25}$$

In Eq (25), $r_{i,T}$ and its derivatives are given by the trajectory generator, and r_i and $\dot{r}_i$ are provided by the state estimation system, allowing for the commanded acceleration $\ddot{r}_i^{\text{des}}$ to be calculated:

$$\ddot{r}_i^{\text{des}} = \ddot{r}_{i,T} - k_{d,i}(\dot{r}_i - \dot{r}_{i,T}) - k_{p,i}(r_i - r_{i,T}) \ . \tag{26}$$

Now the attitude of the quadrotor must be controlled such that it will generate $\ddot{\mathbf{r}}^{\text{des}}$. For this, the linearized version of (17) is used:

$$\ddot{r}_1^{\text{des}} = g(\theta^{\text{des}} \cos\psi_T + \phi^{\text{des}} \sin\psi_T) \tag{27a}$$
$$\ddot{r}_2^{\text{des}} = g(\theta^{\text{des}} \sin\psi_T - \phi^{\text{des}} \cos\psi_T) \tag{27b}$$
$$\ddot{r}_3^{\text{des}} = \frac{1}{m}u_1 - g. \tag{27c}$$

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 $\psi^{\text{des}} = \psi_T$ is given directly by the trajectory generator.

Linearizing (18), we get:

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \mathbf{u}_2 \ , \tag{28}$$

and by further exploiting that via Eq (5) Euler angle velocities are to linear approximation equal to angular velocities, it becomes evident that we can directly command the acceleration of the Euler angles:

$$I \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \mathbf{u}_2 \ . \tag{29}$$

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

$$\mathbf{u}_2 = I \begin{bmatrix} -k_{p,\phi}(\phi - \phi^{\text{des}}) - k_{d,\phi}\,(p - p^{\text{des}}) \\ -k_{p,\theta}(\theta - \theta^{\text{des}}) - k_{d,\theta}\,(q - q^{\text{des}}) \\ -k_{p,\psi}(\psi - \psi_T) - k_{d,\psi}\,(r - \dot{\psi}_T) \end{bmatrix} \ , \tag{30}$$

where the desired roll and pitch velocities p^{des} and q^{des} can be computed from the equations of motion and the specified trajectory [1], but in practice can be set to zero. [2]

In summary, the controller then works as follows:

1. Use Eq (26) to compute the commanded acceleration $\ddot{\mathbf{r}}^{\text{des}}$.
2. Use Eq (27c) to compute u_1 and the desired angles θ^{des} and ϕ^{des} from (27a) and (27b).
3. Use Eq (30) to compute $\mathbf{u}_2$.

3.4 A Geometric Nonlinear Controller

Nonlinear controllers are generally built based on geometric intuition, i.e. the quadrotor's $\mathbf{b}_3$ axis is tilted to point in the desired direction, and thrust is applied. Such controllers are suitable for very aggressive maneuvers and will allow for faster flight and sharper turns (and inverted loops!). Note that we have considerable freedom to choose a control algorithm so long as it results in a stable system.

The following section is based on the controller developed in [1], which in turn is a simplified version of the controller in [2]. For a thorough treatment and stability analysis please refer to [2].

It turns out that the basic layout of the controller remains as shown in Figure 2, i.e. there is an attitude and a position controller, although we no longer make the backstepping assumption.

We start out from the PD controller in Eq (26), but now write it in more compact vector form:

$$\ddot{\mathbf{r}}^{\text{des}} = \ddot{\mathbf{r}}_T - K_d(\dot{\mathbf{r}} - \dot{\mathbf{r}}_T) - K_p(\mathbf{r} - \mathbf{r}_T) \ , \tag{31}$$

where K_d and K_p are diagonal, positive definite gain matrices.

We are again faced with the question how to compute the inputs u_1, $\mathbf{u}_2$ to generate $\ddot{\mathbf{r}}^{\text{des}}$. We first determine the input u_1. By rearranging Eq (17) one obtains:

$$m\ddot{\mathbf{r}}^{\text{des}} + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} = u_1 R \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \ . \tag{32}$$

Note that the lhs of (32) is just the total commanded force $\mathbf{F}^{\text{des}}$ (including gravity):

$$\mathbf{F}^{\text{des}} = m\ddot{\mathbf{r}}^{\text{des}} + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \ . \tag{33}$$

On the right hand side is u_1, multiplied with the quadrotor's axis $\mathbf{b}_3 = R[0, 0, 1]^T$, expressed in the inertial frame. We obtain the input u_1 by projecting $\mathbf{F}^{\text{des}}$ onto $\mathbf{b}_3$:

$$u_1 = \mathbf{b}_3^T \mathbf{F}^{\text{des}} \ . \tag{34}$$

To compute $\mathbf{u}_2$, we observe that a quadrotor can only produce thrust along its $\mathbf{b}_3$ axis. It makes sense to align $\mathbf{b}_3$ with $\mathbf{F}^{\text{des}}$, and align $\mathbf{b}_1$ to match the desired yaw ψ_T. Please refer to Fig. 3. In the following steps we find a triad

[2] On the RHS of Eq. (30) there should also be the trajectory angular accelerations, analogous to $\ddot{r}_{i,T}$ in Eqn (25). By exploiting "Differential Flatness" [1] the trajectory angular accelerations can be computed from the equations of motion and $\mathbf{z}_T(t)$ and its derivatives. However, in practice these terms can be neglected.

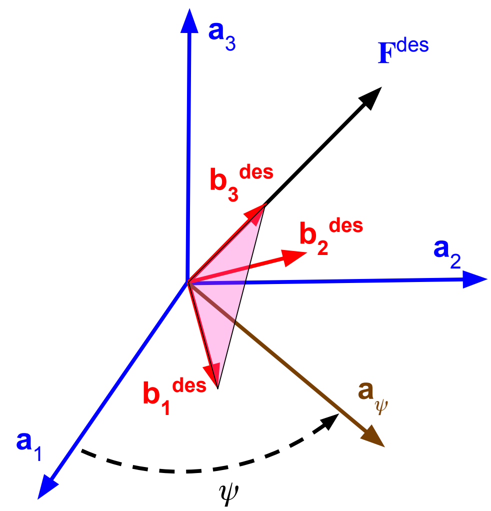

Figure 3: Geometry based attitude control. First ${\mathbf{b}_3}^{\text{des}}$ is aligned along $\mathbf{F}^{\text{des}}$. Then ${\mathbf{b}_2}^{\text{des}}$ is chosen to be *perpendicular* to the plane spanned by the desired yaw heading vector $\mathbf{a}_\psi$ and ${\mathbf{b}_3}^{\text{des}}$. This ensures that ${\mathbf{b}_1}^{\text{des}}$ and ${\mathbf{b}_3}^{\text{des}}$ are *in* the plane containing $\mathbf{a}_\psi$.

$R^{\text{des}} = [{\mathbf{b}_1}^{\text{des}}, {\mathbf{b}_2}^{\text{des}}, {\mathbf{b}_3}^{\text{des}}]$ that has the proper alignment, then proceed to develop a control input $\mathbf{u}_2$ that produces the corresponding torque to drive the attitude towards R^{des}.

As indidicated, the ${\mathbf{b}_3}^{\text{des}}$ axis should be oriented along the desired thrust:

$$ {\mathbf{b}_3}^{\text{des}} = \frac{\mathbf{F}^{\text{des}}}{||\mathbf{F}^{\text{des}}||} \; . \tag{35} $$

Next, we want the ${\mathbf{b}_2}^{\text{des}}$ axis to be perpendicular to both ${\mathbf{b}_1}^{\text{des}}$ and the vector $\mathbf{a}_\psi$ that defines the yaw direction in the plane ($\mathbf{a}_1$, $\mathbf{a}_2$) plane:

$$ \mathbf{a}_\psi = \begin{bmatrix} \cos\psi_T \\ \sin\psi_T \\ 0 \end{bmatrix} \; . \tag{36} $$

This can be accomplished by forming the cross product between ${\mathbf{b}_3}^{\text{des}}$ and $\mathbf{a}_\psi$:

$$ {\mathbf{b}_2}^{\text{des}} = \frac{{\mathbf{b}_3}^{\text{des}} \times \mathbf{a}_\psi}{||{\mathbf{b}_3}^{\text{des}} \times \mathbf{a}_\psi||} \tag{37} $$

which guarantees that the plane formed by ${\mathbf{b}_3}^{\text{des}}$ and the axis ${\mathbf{b}_1}^{\text{des}}$ representing the head of the robot contains the $\mathbf{a}_\psi$. Finally, ${\mathbf{b}_1}^{\text{des}}$ is obtained by cross product and the desired rotation matrix is:

$$ R^{\text{des}} = [{\mathbf{b}_2}^{\text{des}} \times {\mathbf{b}_3}^{\text{des}}, {\mathbf{b}_2}^{\text{des}}, {\mathbf{b}_3}^{\text{des}}] \; . \tag{38} $$

Next, we need a measure for the error in orientation ${R^{\text{des}}}^T R$. The following error vector (see [1])

$$ \mathbf{e}_R = \frac{1}{2}({R^{\text{des}}}^T R - R^T R^{\text{des}})^\vee \tag{39} $$

is obtained from the matrices by taking the $\vee$ operator that maps elements of so(3) to $\mathbb{R}^3$. This error vector is zero when $R^{\text{des}} = R$. It is the rotation vector that generates the error in orientation. Consequently, by applying a torque along its direction it can be decreased, suggesting the control input:

$$\mathbf{u}_2 = I(-K_R\mathbf{e}_R - K_\omega\mathbf{e}_\omega)\ , \tag{40}$$

where $\mathbf{e}_\omega = \boldsymbol{\omega} - \boldsymbol{\omega}^{\text{des}}$ is the error in angular velocities and K_R and K_ω are diagonal gains matrices. The desired angular velocities $\boldsymbol{\omega}^{\text{des}}$ can be computed from the output of the trajectory generator $\mathbf{z}_T$ and its derivatives, but setting them to zero will work for the purpose of this project.

To summarize, the steps to implement the controller are:

1. calculate $\mathbf{F}^{\text{des}}$ from Eq (33), (32), and (31).
2. compute u_1 from Eq (34)
3. determine R^{des} from Eq (38) and the definitions of $\mathbf{b}_i{}^{\text{des}}$.
4. find the error orientation error vector $\mathbf{e}_R$ from Eq (39) and substitute $\boldsymbol{\omega}$ for $\mathbf{e}_\omega$.
5. compute $\mathbf{u}_2$ from Eq (40).

4 System 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 Python and sent to the robot via the CrazyRadio (2.4GHz). Attitude control is performed onboard using the microcontroller, though you will control it in simulation.

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:

(a) mass: $m = 0.030\,\text{kg}$;

(b) the distance from the center of mass to the axis of a motor: $l = 0.046\,\text{m}$; and

(c) the components of the inertia dyadic using $\mathbf{b}_i$ in kg m^2:

$$[I_C]^{\mathbf{b}_i} = \begin{bmatrix} 1.43\times10^{-5} & 0 & 0 \\ 0 & 1.43\times10^{-5} & 0 \\ 0 & 0 & 2.89\times10^{-5} \end{bmatrix}.$$