AER1217: Development of Unmanned Aerial Vehicles
Lab 2: Quadrotor Simulation and Control Design

1 Introduction

This is the second lab in a series designed to complement the lecture material and help you with the course project. This lab requires the design of a quadrotor position controller, and implementation in simulator. It is set up to work in Pybullet, which simulates your interface with the Crazyflie drone and the Vicon motion capture system for position and attitude measurements. To be able to complete this lab within your dedicated time slot, you should have preliminary knowledge of controller design and Python programming.

1.1 Lab Objectives

The lab is split up into three separate sections:

1. Design a position controller
2. Implement the controller in Python
3. Implement the controller to track a time-based circular trajectory.

1.2 Requirements

For this lab, the quadrotor is required to track a circular trajectory about point $(x, y) = (0, 0)$ m with a radius of 3 m, at altitude $z = 1$ m and speed $v = 4$ m/s. The specifications of the circular trajectory are given in `lab2.yaml`. The trajectory is generated in `planner.py`.

You will need to modify the function in `geo_controller.py` to finish three tasks. You are required to demonstrate the trajectory on the demonstration day **February 8, 2024**.

1.3 Code Repository

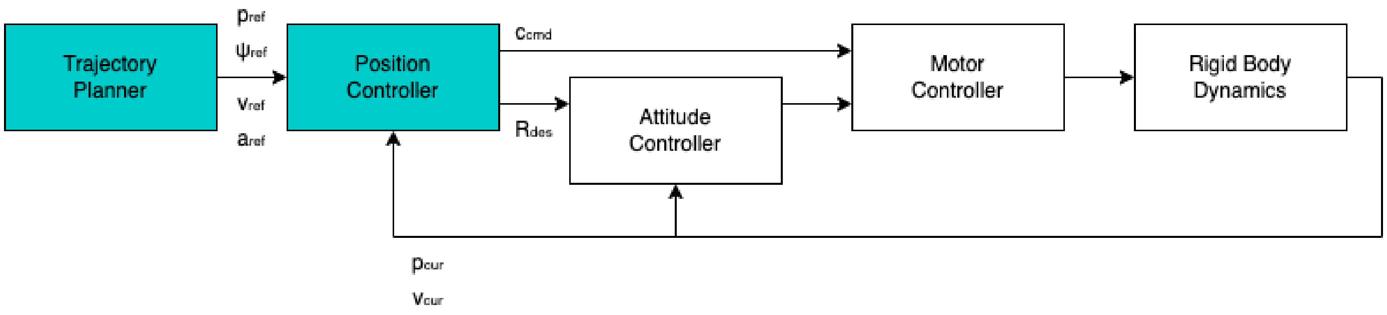
The repository for this lab can be found in the Quercus Assignment Lab 2. The files that will need to be modified is `geo_controller.py`.

2 Lab Components

2.1 World Frame and Body Frame

In this lab, we make use of a world frame W with orthonormal basis $\{\mathbf{x}_W, \mathbf{y}_W, \mathbf{z}_W\}$ represented in world coordinates and a body frame B with orthonormal basis $\{\mathbf{x}_B, \mathbf{y}_B, \mathbf{z}_B\}$ also represented in world coordinates. The body frame is fixed to the quadrotor with an origin coinciding with its center of mass. The quadrotor is subject to a gravitational acceleration g in the $-\mathbf{z}_W$ direction. We denote the position of the quadrotor's center of mass as $\mathbf{p}$, and its derivatives, velocity, acceleration, jerk, and snap as $\mathbf{v}$, $\mathbf{a}$, $\mathbf{j}$, and $\mathbf{s}$, respectively. We represent the quadrotor's orientation as a rotation matrix $\mathbf{R} = [\mathbf{x}_B \ \mathbf{y}_B \ \mathbf{z}_B]$ and its body rates (i.e., the angular velocity) as ω represented in body coordinates.

2.2 Designing a Quadrotor Position Controller



In this lab, in order to track a reference trajectory, we use a controller consisting of feedback terms computed from tracking errors as well as feed-forward terms computed from the reference trajectory using the Crazyflie's differential flatness property. The control algorithm is designed as a classical cascaded structure, i.e., consisting of a high-level position controller and the low-level body-rate controller. The high-level position controller computes the desired orientation R_{des}, the collective thrust input c_{cmd}, the desired body rates ω_{des}, and the desired angular accelerations $\dot{\omega}_{des}$, which are then applied in a low-level controller.

2.2.1 Task 1: Compute the desired acceleration command

As a first step in the position controller, we compute the desired acceleration of the quadrotor's body as

$$\mathbf{a_{des}} = \mathbf{a_{fb}} + \mathbf{a_{ref}} + g\mathbf{z}_w \tag{1}$$

where $\mathbf{a_{fb}}$ are the PD feedback-control terms computed from the position and velocity control errors as

$$\mathbf{a_{fb}} = -\mathbf{K}_{pos}(\mathbf{p} - \mathbf{p}_{ref}) - \mathbf{K}_{vel}(\mathbf{v} - \mathbf{v}_{ref}) \tag{2}$$

where $\mathbf{K}_{pos}$ and $\mathbf{K}_{vel}$ are constant diagonal matrices.

2.2.2 Task 2: Compute the desired thrust command

The desired thrust c_{cmd} then becomes

$$c_{cmd} = m \left\| \mathbf{a_{des}} \right\| . \tag{3}$$

2.2.3 Task 3: Compute the desired attitude command

To enforce a reference heading ψ, we constrain the projection of the $\mathbf{x}_B$ axis into the $\mathbf{x}_W - \mathbf{y}_W$ plane to be collinear with $\mathbf{x}_C$, where

$$\begin{aligned} \mathbf{x}_C &= \begin{bmatrix} \cos\psi & \sin\psi & 0 \end{bmatrix}^T \\ \mathbf{y}_C &= \begin{bmatrix} -\sin\psi & \cos\psi & 0 \end{bmatrix}^T \end{aligned} \tag{4}$$

We compute the desired orientation $\mathbf{R}_{des}$ such that the desired acceleration $\mathbf{a}_{des}$ is respected as

$$\mathbf{z}_{B,des} = \frac{\mathbf{a}_{des}}{\|\mathbf{a}_{des}\|} \tag{5}$$

$$\mathbf{x}_{B,des} = \frac{\mathbf{y}_C \times \mathbf{z}_{B,des}}{\|\mathbf{y}_C \times \mathbf{z}_{B,des}\|} \tag{6}$$

$$\mathbf{y}_{B,des} = \mathbf{z}_{B,des} \times \mathbf{x}_{B,des} \tag{7}$$

2.3 Implementing the Controller

Your goal is to implement the geometric controller to compute the desired thrust input and the desired orientation. In order to do so, you will need to modify the `_compute_desired_force_and_euler` function in `geo_controller.py`.

The arguments are as follows:

- control_timestep (float): The time step at which control is computed.
- cur_pos (ndarray): (3,1)-shaped array of floats containing the current position.
- cur_quat (ndarray): (4,1)-shaped array of floats containing the current orientation as a quaternion.
- cur_vel (ndarray): (3,1)-shaped array of floats containing the current velocity.
- cur_ang_vel (ndarray): (3,1)-shaped array of floats containing the current angular velocity.
- target_pos (ndarray): (3,1)-shaped array of floats containing the desired position.
- target_rpy (ndarray): (3,1)-shaped array of floats containing the desired yaw angle.
- target_vel (ndarray): (3,1)-shaped array of floats containing the desired velocity.
- target_acc (ndarray): (3,1)-shaped array of floats containing the desired acceleration.
- target_rpy_rates (ndarray): (3,1)-shaped array of floats containing the desired body rate.

The returns are as follows:

- desired_thrust (float): The target thrust along the drone z-aixs.
- desired_euler (ndarray): (3,1)-shaped array of floats containing the desired Euler angles.
- pos_e (ndarray): (3,1)-shaped array of floats containing the error between target position and current position.

3 Deliverable and Grading

This lab is worth 15% (Lab demonstration: 5% + Report: 10%). Please submit the code and a PDF report, in a .zip format, to Quercus, by **February 15, 2024, 11:59 PM EST** at the **latest**:

- Lab demonstration (5%):
  - Enable the quadcopter to fly the circular trajectory (5%)
- The required codes in `geo_controller.py` (4%):
  - Task 1 (1%)
  - Task 2 (1%)
  - Task 3 (1%)
  - Overall tracking performance (1%)
- A PDF report of **maximum 3 pages** (5%) that:
  - shows the derivations of the position controller (1%)
  - includes graphs of errors in the x, y, z positions and yaw angle (1%)
  - includes graphs of desired and actual x, y, z positions and yaw angle (1%)
  - provides the control parameters (e.g. PID constants) used in the simulation (1%)
  - comments on the effects of changing control gains and trajectory requirements on the performance of the UAV. (1%)
- Code style (1%):
  - Please comment your codes appropriately and provide a `readme.txt` file with proper explanation of the code structure.