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 ROS. It is set up to work with Gazebo, which simulates your interface with the Parrot AR.Drone UAV 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 ROS, and Python programming.

1.1 Lab Objectives

The lab is split up into four separate sections:

1. Design a position controller (with yaw control)
2. Implement the controller in Python/ROS
3. Implement a ROS node that publishes control commands and subscribes to quadrotor actual and desired positions
4. Implement a ROS node that publishes a time based trajectory to fly linear and circular trajectories.

1.2 Requirements

For this lab, the quadrotor is required to fly in two trajectories:

- **Linear trajectory:** from the point $(x, y, z) = (-1.5, -1.5, 1)$ m to $(1.5, 1.5, 2)$ m and back to the initial position $(-1.5, -1.5, 1)$ m in inertial space, and
- **Circular trajectory:** a circular trajectory about point $(x, y) = (0, 0)$ m with a radius of 1.5 m with altitude varying between a minimum altitude of 0.5 m and a maximum altitude of 1.5 m, similar to the linear trajectory, half the path climbs from 0.5 m to 1.5 m, and the other half descends back to 0.5 m. The heading of the drone must always face the origin of the circle $(x, y) = (0, 0)$ (i.e. non-zero yaw). This requires a yaw-control, and also that the roll and pitch controllers can compensate for a non-zero heading angle ψ.

You are required to demonstrate the two trajectories separately on the demonstration day **February 7, 2022**.

1.3 Code Repository

The repository for this lab can be found here: `https://bitbucket.org/aer1217/`. The files that will need to be modified are:

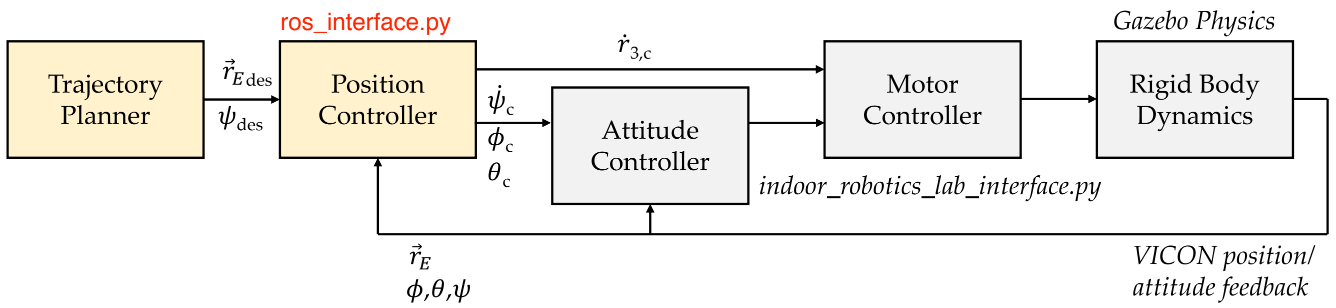
- `scripts/ros_interface.py`,
- `scripts/position_controller.py`,
- `scripts/desired_positions.py`, and

- `launch/ardrone_simulator.launch`,

which can be found in `ardrone_labs/src/aer1217_ardrone_simulator/`.

2 Lab Components

2.1 Designing a Quadrotor Position Controller



In this lab, you are required to write a **position controller** (with yaw control) to track the desired trajectory $\{\vec{r}_{Edes}, \psi_{des}\}$. The overall control architecture for AR.Drone is shown in the figure above. The on-board attitude controller has been written for you in the `indoor_robotics_lab_interface.py` file. It takes in the commanded yaw rate, climb rate, pitch angle, and roll angle from your position controller, as described in Section 2.4. Note that this is slightly different than the typical control shown in the lecture notes, as the outputs of the position controller for this lab are $\dot{r}_{3,c}$, ϕ_c, θ_c, and $\dot{\psi}_c$. In addition, the VICON motion capture system only provides position and attitude data, hence velocity or angular rates have to be obtained through numerical differentiation.

The AR.Drone reference frame is defined such that the x-axis points towards the forward direction, y-axis points towards the left, and z-axis points upwards. A similar coordinate frame is used for the inertial frame of reference, i.e. z-axis of the inertial frame points upwards such that altitude is positive and the gravitational acceleration is -9.8 m/s^2.

As a general guide, the process of designing the position controller involves the following:

1. Express the dynamics of the UAV in the inertial (Earth) frame of reference
2. Simplify by assuming yaw angle $\psi = 0°$
3. Express $\ddot{x}$, $\ddot{y}$, $\ddot{z}$ in terms of the roll angle ϕ, pitch angle θ and motor thrust f
4. Express the commanded roll angle ϕ_c, commanded pitch angle θ_c, and commanded thrust f_c in terms of commanded accelerations $\ddot{x}_c$, $\ddot{y}_c$, $\ddot{z}_c$
5. Calculate the commanded accelerations $\ddot{x}_c$, $\ddot{y}_c$, $\ddot{z}_c$ by designing a second-order system based on the error in positions and velocities (PD controller)
6. For non-zero yaw, transform the commanded roll ϕ_c and pitch θ_c from the inertial frame of reference to the quadrotor body frame of reference

2.2 Implementing the Quadrotor Position Controller

The position controller should be written in the `position_controller.py` file. Ensure that there is a method (member function) that can be called that will output the desired pitch angle, roll angle, yaw rate and climb rate commands for the drone.

2.3 Overview of Quadrotor Dynamics and Control

As shown in lecture, the dynamics of the quadrotor can be expressed as:

$$m\ddot{\vec{r}}_E = -mg\vec{z}_E + R_{EB}\begin{bmatrix} 0 \\ 0 \\ F_1+F_2+F_3+F_4 \end{bmatrix} \quad (1)$$

$$I\dot{\vec{\omega}}_B = \begin{bmatrix} L(F_2-F_4) \\ L(F_3-F_1) \\ M_1-M_2+M_3-M_4 \end{bmatrix} - \vec{\omega}_B \times I\vec{\omega}_B \quad (2)$$

where:

$$R_{BE} = R_\phi R_\theta R_\psi \quad (3)$$

$$R_{EB} = R_\psi^T R_\theta^T R_\phi^T \quad (4)$$

$$= \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \quad (5)$$

$$= \begin{bmatrix} \cdot & \cdot & \sin\phi\sin\psi+\cos\phi\sin\theta\cos\psi \\ \cdot & \cdot & -\sin\phi\cos\psi+\cos\phi\sin\theta\sin\psi \\ \cdot & \cdot & \cos\theta\cos\phi \end{bmatrix} \quad (6)$$

If we simplify it to have $\psi = 0$:

$$R_{EB} = \begin{bmatrix} \cdot & \cdot & \cos\phi\sin\theta \\ \cdot & \cdot & -\sin\phi \\ \cdot & \cdot & \cos\theta\cos\phi \end{bmatrix} \quad (7)$$

Hence:

$$\ddot{x} = f\cos\phi\sin\theta \quad (8)$$

$$\ddot{y} = -f\sin\phi \quad (9)$$

$$\ddot{z} = f\cos\theta\cos\phi - g \quad (10)$$

Using feedback linearization, we can express the desired angles and body z-axis forces:

$$\phi = \sin^{-1}\frac{-\ddot{y}}{f} \tag{11}$$

$$\theta = \sin^{-1}\frac{\ddot{x}}{f\cos\phi} \tag{12}$$

$$f = \frac{\ddot{z}+g}{\cos\theta\cos\phi} \tag{13}$$

Controller design steps:

1. Calculate current mass-normalized thrust using $\ddot{z}$, θ and ϕ from measurements (VICON data and some numerical differentiation):
$$f = \frac{\ddot{z}+g}{\cos\theta\cos\phi} \tag{14}$$
2. Determine desired x-/y-accelerations by designing a second-order system with controller parameters ω_n (natural frequency) and ζ (damping ratio)
$$\ddot{x}_c = 2\zeta_x\omega_{n,x}(\dot{x}_d - \dot{x}) + \omega_{n,x}^2(x_d - x) \tag{15}$$
$$\ddot{y}_c = 2\zeta_y\omega_{n,y}(\dot{y}_d - \dot{y}) + \omega_{n,y}^2(y_d - y) \tag{16}$$
3. Calculate the commanded roll and then pitch:
$$\phi_c = \sin^{-1}\frac{-\ddot{y}}{f} \tag{17}$$
$$\theta_c = \sin^{-1}\frac{\ddot{x}}{f\cos\phi_c} \tag{18}$$
4. For non-zero yaw, transform the commanded roll and pitch in the inertial frame to the quadrotor frame
$$\phi_{c,B} = \phi_c\cos\psi + \theta_c\sin\psi \tag{19}$$
$$\theta_{c,B} = -\phi_c\sin\psi + \theta_c\cos\psi \tag{20}$$

2.4 A ROS Node for Communicating Control Commands

Create a ROS node that subscribes to the following topic:

- `/vicon/ARDroneCarre/ARDroneCarre` which communicates using a `geometry_msgs TransformStamped` message, where quadrotor translation and rotation are contained in `Transform`.

Also get the ROS node to publish to the following topic:

- `cmd_vel_RHC` with a `geometry_msgs Twist` message containing the desired roll angle in radians in `linear.x`, the desired pitch angle in radians in `linear.y`, the desired yaw rate in radians per seconds in `angular.z`, and the desired climb rate in meters per second in `linear.z`. These values saturate at -1 and 1. E.g. the quadrotor can pitch a maximum of 1 rad.

You may or may not want to publish or subscribe to other topics depending on the overall architecture you choose to implement, or different information you want to communicate between multiple nodes if you have them.

Create an instance of the class you made in the Section 2.2. The member function of this instance can then be called to return the desired control inputs, which the node can then publish to the onboard controller. Note that you can work on this section simultaneously, while the controller class is being written.

2.5 A ROS Node for Communicating Desired Trajectory

Create either a ROS node or class that generates desired positions for the position controller in Section 2.1. Ensure that the position controller receives the desired position. You should ensure that this is updated regularly such that the desired position changes with time. Similar to the previous sections, this section can be developed in parallel.

2.6 Putting it Together

Finally, ensure that the `launch` file launches all the ROS nodes. The indoor robotics lab interface and Gazebo is already be completed for you. To run the Gazebo simulator, type the following in Terminal: `roslaunch aer1217_ardrone_simulator ardrone_simulator.launch`. A Gazebo window will open up, and at the bottom bar, click the "play" button to begin the simulation. In situations where the simulator freezes up or crashes, close the window, hit `Ctrl+C` in the Terminal, and relaunch it using the `roslaunch` command.

Tune the controllers' parameters such that the quadrotor can track the desired trajectory satisfactorily, feel free to change the desired flying speed and angular velocity to complete the linear and spiral trajectory respectively. You may also want to interpolate points in your reference signal. i.e. rather than having 3 reference points with 3 step functions for the linear path, interpolate points in between to generate a ramp signal. This will make the tracking smoother.

To include meaningful results in your report, you may want search `rosbag`, or an alternative way of logging your data such as `csv` in Python.

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 14, 2022, 11:59 PM EST** at the **latest**:

- Lab demonstration (5%):
  - Enable the quadrotor to fly the linear trajectory (1%)
  - Enable the quadcopter to fly the circular trajectory (2%)
  - Enable the quadcopter to handle non-zero yaw command in the circular trajectory (2%)
- The required codes (4%):
  - `ros_interface.py` (0.5%)
  - `position_controller.py` (2%)
  - `desired_positions.py` (1%)
  - `ardrone_simulator.launch` (0.5%)
  - Additional scripts that you wrote for the simulator, if any
  - A `readme.txt` file to briefly explain how to run your code.

- 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 (speed of flying a linear trajectory or angular velocity of flying a spiral trajectory) 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.