16-665: Robot Mobility
Fall, 2024

Author: Prof. Guanya Shi
Instructor: Prof. Guanya Shi

Air Mobility
Project Description
Due: Oct 31, 2024

Overview and Objectives

This project emphasizes the dynamics modeling, feedback control design, and trajectory planning for aerial robots in simulation. In particular, you are expected to:

- Build dynamics models for a 2d drone and a quadrotor;
- Implement linear, cascaded nonlinear, and adaptive control methods covered in class;
- Implement basic trajectory planning methods covered in class.

Extra credit questions are provided for those individuals interested in pursuing more advanced concepts.

Project Report Format and Grading Criteria

Each individual is expected to submit a project report that will be graded out of a total of 100 points based on content correctness (40%), quality (20%), completeness (20%), and clarity of presentation (20%). Individual point values are indicated next to each question in square brackets (e.g., [1] indicates a value of 1 point). Moreover, there are 15 extra points for more advanced concepts.

Collaboration: The project is expected to be completed by individuals (and is not a team project). Collaboration with one other person is acceptable. However, such collaboration is limited to discussion and direct code/report/result copy is strictly prohibited. Collaborators must be acknowledged explicitly on the report by name. Failure to report collaborators or collaborate with more than one person will be viewed as a violation of the CMU Academic Integrity Policy (as detailed in the course syllabus).

Code Structure and Dependencies

Code Structure: There are three Python code files. "2d.py" includes all problems for 2d drone in Part A and "quadrotor.py" is for Part B. "math_utils.py" has several useful helper functions for quaternion and rotation matrix computations. "crazyflie2.stl" is a model file for the meshcat animation.

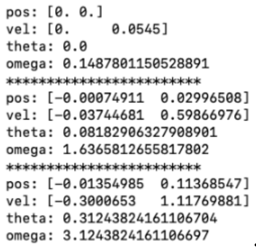
Dependencies: You can simply run "pip install -r requirement.txt" to install all the dependencies or install the following libraries yourself:

- Basic libraries: "matplotlib", "numpy", "time", "argparse" and "scipy"
- Python control systems library "control": https://github.com/python-control/python-control
- 3d viewer "meshcat" for animations: https://github.com/meshcat-dev/meshcat-python

Guideline for Figures: For figures in the report, please add titles or captions to point out which question it is for.

Part A: 2d Drone ([50] + [10] extra points)

1. *System Modeling [10]:* Based on lecture 2, complete the "dynamics" function in the "Quadrotor_2D" class. Run "python 2d.py 1" to observe the output. We have disabled randomness for this question so you should see results same or similar (up to numerical errors) to this:



```
pos: [0. 0.]
vel: [0.      0.0545]
theta: 0.0
omega: 0.1487801150528891
************************
pos: [-0.00074911  0.02996508]
vel: [-0.03744681  0.59866976]
theta: 0.08182906327908901
omega: 1.6365812655817802
************************
pos: [-0.01354985  0.11368547]
vel: [-0.3000653   1.11769881]
theta: 0.3124382416110670
omega: 3.1243824161106697
```

. Take a screenshot of your output.

2. *Cascaded Setpoint Control [10]:* Based on lecture 3, complete the “cascaded_control” function in the “Quadrotor_2D” class. Design PD gains for position and attitude control. Run “python 2d.py 2” to track two setpoints: from initial condition [0, 0] to [1, 1], and from [0, 0] to [1, 0]. Answer the following questions:

    - [5] Report the figure for both setpoints, and comment on their differences. For both setpoints, take three screenshots of the meshcat animation at different time steps.

    - [5] What PD gains do you choose? Compare the performance of the gain values you choose and one set of other gain values, in terms of the rise time (90%), maximum overshoot, and average control energy (the average of $||u||^2$).

3. *Linear Control and LQR [10] (and [5] extra points):* Based on lecture 3, complete the “linear_control” function in the “Quadrotor_2D” class. Choose diagonal and positive definite Q, R matrix and use the “lqr” function in “control” library to compute the K matrix. Like problem 2, run “python 2d.py 3” to track two setpoints: [1, 1] and [1, 0]. Answer the following questions:

    - [10] Report the figure for both setpoints. Quantify and comment on the difference between LQR control and the cascaded setpoint control in problem 2, in terms of the rise time (90%), maximum overshoot, and average control energy.

    - ([5] extra points) What Q and R do you choose? Scale the R matrix by 0.1 and 10, and elaborate what happens. Change the ratio between pos/vel error and theta/omega error in the Q matrix, and elaborate what happens. Moreover, initialize the drone with a large angle ($\theta = \pi/3$), is LQR still good (compare to the cascaded control method in problem A-2) and why?

4. *Cascaded Tracking Control [10]:* Based on lecture 3, use the “cascaded_control” function in the “Quadrotor_2D” class to track a figure-8 trajectory: $p_x = \sin(t)\,, p_y = \frac{1}{2}\cos\left(2t + \frac{\pi}{2}\right)$. Run “python 2d.py 4” and answer the following questions:

    - [5] Report the figure, and compute the controller performance, using position RMSE (rooted mean squared error) as the metric.

    - [5] Remove the acceleration feedforward term a_d in the controller. Elaborate what happens and explain why. Further set $v_d = 0$ in the controller and elaborate and explain what the difference is. For both cases, report figures and RMSE numbers.

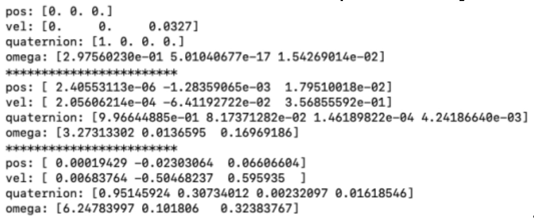
5. *Trajectory Generation and Differential Flatness [10] (and [5] extra points):* Based on lecture 5, design a polynomial reference trajectory from [0, 0] to [1, 0]. Then using the “cascaded_control” function in the “Quadrotor_2D” class to track this trajectory. Use differential flatness to compute ω_d, τ_d for “cascaded_control”. Run “python 2d.py 5” for this question. Answer the following question:

    - [5] Report the figure and RMSE results with two terminal times: 2*pi seconds (the default value) and 2 seconds, by changing the “total_time” variable. Compare the tracking results with the result from tracking a “naïve” reference trajectory with constant velocity ($p_x = t/total\_time$), for both terminal times. What do you observe and why?

    - [5] For both terminal times (2*pi seconds and 2 seconds), use the “cascaded_control” function with $\omega_d = 0, \tau_d = 0$ to track the designed polynomial reference trajectory. Report the figure and compare the result with the differential-flatness-based method used in the last question. What is the difference and why?

    - ([5] extra points) Instead of using state-dependent ω_d, τ_d , use open-loop $\omega_r(t), \tau_r(t)$ in the “cascaded_control” function to track the designed trajectories. What do you observe and why? Further, instead of using the cascaded controller, directly use open-loop trajectories $T_r(t), \tau_r(t)$ for control (no feedback) and analyze the results with and without system randomness (by setting “self.sigma_t” and “self.sigma_r” to be zero).

Part B: Quadrotor ([50] + [5] extra points)

1. *System Modeling [15]:* Based on lecture 2, complete the "dynamics" function in the "Quadrotor" class. Run "python quadrotor.py 1" to observe the output. We have disabled randomness for this question so you should see results same or similar (up to numerical errors) to this: 

```
pos: [0. 0. 0.]
vel: [0.     0.     0.0327]
quaternion: [1. 0. 0. 0.]
omega: [2.97560230e-01 5.01040677e-17 1.54269014e-02]
************************
pos: [ 2.40553113e-06 -1.28359065e-03  1.79510018e-02]
vel: [ 2.05606214e-04 -6.41192722e-02  3.56855592e-01]
quaternion: [9.96644885e-01 8.17371282e-02 1.46189822e-04 4.24186640e-03]
omega: [3.27313302 0.0136595  0.16969186]
************************
pos: [ 0.00019429 -0.02303064  0.06606604]
vel: [ 0.00683764 -0.50468237  0.595935  ]
quaternion: [0.95145924 0.30734012 0.00232097 0.01618546]
omega: [6.24783997 0.101806   0.32383767]
```

. Take a screenshot of your output.

2. *Cascaded Control [20]:* Based on lecture 3, complete the "cascaded_control" function in the "Quadrotor" class. Design PD gains for position and attitude control. Run "python quadrotor.py 2" to track a setpoint [1, 1, 1], and a spiral trajectory $p_x = \sin(2t), p_y = \cos(2t) - 1, p_z = 0.5t$. Answer the following questions:

    - [10] For the setpoint tracking task, first set the desired yaw angle (ψ_d) to be 0 and try a time-variant desired yaw angle ($\psi_d = \frac{t}{total\ time} * \frac{\pi}{3}$). Report both figures and compare the position RMSE and average control energy. For both cases, take three screenshots of the meshcat animation at different time steps.

    - [10] For the trajectory tracking task ($\psi_d = 0$), what PD gain values do you choose? Compare the performance of the gain values you choose and *two* sets of other gain values, in terms of the position RMSE. Report all three figures.

3. *Integral Control and Adaptive Control [15] (and [5] extra points):* In this question we have added some disturbance to the robot ("robot.d"). Based on lecture 4, complete the "adaptive_control" function in the "Quadrotor" class. Run "python quadrotor.py 3" to track a setpoint [1, 1, 1], and a spiral trajectory $p_x = \sin(2t), p_y = \cos(2t) - 1, p_z = 0.5t$. $\psi_d = 0$ for both cases. Answer the following questions:

    - [5] Use the default constant disturbance value ([0.5, 0.5, 1.0]). Implement integral control as the most simple adaptive control. Choose *two* different I gains and compare their performance, for both setpoint tracking (in terms of rise time, position RMSE, and maximum overshoot) and trajectory tracking (in terms of position RMSE).

    - [5] Change the disturbance to be time-variant ([0.5, sin(t), cos(\sqrt{2}t)]). Apply the best controller in the last question for both setpoint and trajectory tracking. Comment on the difference from the constant disturbance case.

    - [5] Apply the time-variant disturbance [0.5, sin(t), cos(\sqrt{2}t)]. Implement the L-1 adaptive control method taught in lecture 4 for both setpoint and trajectory tracking. Is it better than integral control and why?

    - ([5] extra points) Change the disturbance to be state dependent ($d = \phi(x)a$). Design some interesting feature function $\phi(x)$ (e.g., air drag model, neural networks, polynomials) and implement the adaptive nonlinear control method taught in lecture 4, for both setpoint and trajectory tracking.