AMR HW #4

Nicola Bezzo

UVa 11/12/2019

HW#4 Due Wed. 11/26 Midnight (same rules as usual)

1. **(50pt)** In this assignment, you will get started with a **1D quadrotor simulator** and and implement a controller for it. We will be testing your controller with two cases. In the first test case, the quadrotor simply needs to stabilize at a height of 0. The second test case gives the quadrotor a step input of 1 meter; that is, your quadrotor will be asked to rise to a height of 1 meter. The aim of this assignment is to get your controller to achieve a desired response for each test case.

2. **(50pt)** In this assignment, you will design and implement a controller for **a planar (2D) quadrotor**. We will be testing your controller with two trajectories, the first one is a horizontal line trajectory and the second one is a more complex sine wave shaped trajectory. The aim is to get your controller to follow each trajectory such that the error between the desired position from the trajectory and the actual position of the robot is small.

Instructions for Problem #1 – 1D Quadrotor

The goal of this programming exercise is to get you familiar with working with the quadrotor simulator and implementing a Proportional Derivative (PD) controller.

Before implementing your own function, you should first try running runsim.m in your MATLAB setup. If you see a quadrotor falling from the height 0, then the simulator is running smoothly on your computer and you may continue with other tasks. The controller code, that you need to modify (controller.m) produces robot inputs which are all zero thrust and thus the quadrotor falls due to gravity.

- **PD Controller**

As you have seen in the lecture, the dynamic equation for the motion of the quadrotor in the z direction is

$$\ddot{z} = \frac{u}{m} - g$$

Hence the control input for a PD controller is

$$u = m(\ddot{z}_{\text{des}} + K_p e + K_v \dot{e} + g)$$

where e and $\dot{e}$ can be calculated from the current and desired states (z, zdes, $\dot{z}$, $\dot{z}$ des).

Instructions

- **Files included**

[⋆] controller.m - Controller for a quadrotor.
runsim.m - Test script to be called for testing.
height_control.m - Simulation code that will be called by runsim.
fixed_set_point.m - Step response function.
utils - Helper functions for quadrotor simulator
⋆ indicates files you will need to implement

- **Tasks**
  - You will need to first implement a PD controller for height control of the quadrotor. Then, tune the proportional gain (Kp) and derivative gain (Kv) in the file controller.m until the quadrotor converges quickly and smoothly to a step response input.
  - In the first test case, the quadrotor simply needs to stabilize at a height of 0.
  - The second test case gives the quadrotor a step input of 1 meter; that is, your quadrotor will be asked to rise to a height of 1 meter. The response to this input should have a rise time of less than 1s and a maximum overshoot of less than 5%. Remember that rise time is the time it takes to reach 90% of the steady-state value, so in this case you must reach 0.9 meters in under one second.

Programming Hints

- **Accessing states and robot params**

The position and velocity are stored in the s variable, which is a 2 × 1 vector. They can be accessed stored into the variables pos and vel as follows:

- pos = s(1);
- vel = s(2);

In the same way, the desired position and velocity can be stored into the pos_des and vel_des variables by accessing them in the s_des variable:

- pos_des = s_des(1);
- vel_des = s_des(2);

The robot parameters are stored in the params structure. These parameters can be accessed as follows:

- params.gravity - acceleration due to gravity
- params.mass - robot's mass
- params.arm_length - robot's arm length
- params.u_min - minimum thrust
- params.u_max - maximum thrust

- **Setting the reference height**

The reference height can be controlled in runsim.m by setting the z_des variable:

- z_des = 0; - sets the reference height to zero
- z_des = 1; - sets the reference height to one

Instructions for Problem #2 – 2D Quadrotor

In this exercise, you will be implementing the PD controller discussed in lecture to control the motion of the quadrotor in the Y-Z plane.

- **Dynamics**

For a quadrotor modeled in the Y − Z plane, its orientation is defined by a roll angle, ϕ. It is assumed that its pitch and yaw angles are 0. You will need the rotation matrix for transforming components of vectors in B to components of vectors in A:

$$ {}^{\mathcal{A}}[R]_{\mathcal{B}} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} $$

We will denote the components of angular velocity of the robot in the body frame by $\dot{\phi}$:

$$ {}^{\mathcal{A}}\omega_{\mathcal{B}} = \dot{\phi}. $$

The quadrotor has two inputs: the thrust force (u1) and the moment (u2).

$$ u_1 = \Sigma_{i=1}^{2} F_i \qquad\qquad u_2 = L(F_1 - F_2) $$

where L is the arm length of the quadrotor

Let r=[y,z]^T denote the position vector of the quadrotor in A.
The linear acceleration is given by Newton's Equation of Motion

$$m\ddot{\mathbf{r}} = m\begin{bmatrix}\ddot{y}\\ \ddot{z}\end{bmatrix} = \begin{bmatrix}0\\ -mg\end{bmatrix} + {}^{\mathcal{A}}[R]_{\mathcal{B}}\begin{bmatrix}0\\ u_1\end{bmatrix} = \begin{bmatrix}0\\ -mg\end{bmatrix} + \begin{bmatrix}-u_1\sin(\phi)\\ u_1\cos(\phi)\end{bmatrix}$$

The angular acceleration is determined by Euler's equation of motion

$$I_{xx}\ddot{\phi} = L(F_1 - F_2) = u_2$$

As a result the system model can be written as

$$\begin{bmatrix}\ddot{y}\\ \ddot{z}\\ \ddot{\phi}\end{bmatrix} = \begin{bmatrix}0\\ -g\\ 0\end{bmatrix} + \begin{bmatrix}-\frac{1}{m}\sin(\phi) & 0\\ \frac{1}{m}\cos(\phi) & 0\\ 0 & \frac{1}{I_{xx}}\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix}$$

- **Controller**

The inputs u_1 and u_2 can be derived as

$$u_1 = mg + m\ddot{z}_c = m\{g + \ddot{z}_T(t) + k_{v,z}(\dot{z}_T(t) - \dot{z}) + k_{p,z}(z_T(t) - z)\}$$
$$u_2 = I_{xx}\ddot{\phi}_T(t) = I_{xx}(\ddot{\phi}_c + k_{v,\phi}(\dot{\phi}_c - \dot{\phi}) + k_{p,\phi}(\phi_c - \phi))$$
$$\phi_c = -\frac{\ddot{y}_c}{g} = -\frac{1}{g}(\ddot{y}_T(t) + k_{v,y}(\dot{y}_T(t) - \dot{y}) + k_{p,y}(y_T(t) - y))$$

- **Hover Controller**

Hovering is the special case of which the desired position is constant and the desired roll is zero.

$$u_1 = m[g - k_{v,z}\dot{z} + k_{p,z}(z_0 - z)]$$
$$u_2 = I_{xx}(\ddot{\phi}_c + k_{v,\phi}(\dot{\phi}_c - \dot{\phi}) + k_{p,\phi}(\phi_c - \phi))$$
$$\phi_c = -\frac{1}{g}(k_{v,y}(-\dot{y}) + k_{p,y}(y_0 - y)$$

Instructions

- **Files included**

[⋆] controller.m - Controller for a planar quadrotor.
runsim.m - Test script to be called for testing.
simulation_2d.m - Simulation code that will be called by runsim.
trajecories/ - Folder includes example trajectories for testing your controller.
utils/ - Folder containing helper functions which you can use.
⋆ indicates files you will need to implement

- **Tasks**

- You will need to complete the implementation of a PD controller in the file controller.m so that the simulated quadrotor can follow a desired trajectory. To test different trajectories, you will need to modify the variable trajhandle inside the runsim.m to point to the appropriate function.
- The evaluation is based on how well your controller can follow the desired trajectories.

Gain Tuning

The controller in this assignment is slightly more complex, containing four gains (two for position and two for attitude)

- **Ziegler-Nichols method**

The Ziegler-Nichols method is a method used to tune P, PI, PD, and PID controllers. We will only cover this method as applied to PD controllers. The method is outlined as follows:

1. Set the proportional and derivative gains to zero
2. Turn the proportional gain up until the system outputs sustained oscillations.
3. Note the value of the proportional gain. This value is called the ultimate gain, K_u.
4. Measure T_u, which is the period of the oscillations.
5. Set the proportional gain to $K_p = 0.8K_u$.
6. Set the derivative gain to $K_d = T_u/8$.

If you follow the above procedure and the controller response is too slow, try scaling the proportional and derivative gains up by the same constant until you get a sufficiently quick response.

Because you are implementing a hierarchical control system in which the attitude controller forms the “inner loop”, it is recommended that you first tune the attitude controller and then the position controller. You can tune the attitude controller by setting all of your position gains to zero and setting a desired roll angle to a certain value, say $\pi/6$.

Submission

In Collab: upload your code and a brief instruction on how to run it, together with a pdf document containing data and plots that demonstrate that you achieved the desired tracking asked in the two problems. Please upload a zip file containing all of these files. The file name should contain your family_name and first_name (e.g. Bezzo_Nicola_HW3.zip).