Programming Exercise 1:
Quadrotor Simulator and PD Controller

1 Introduction

The goal of this programming exercise is to get you familiar with working with the quadrotor simulator and implementing a Proportional Derivative (PD) controller. In Week 1, we provided you with a quadrotor GUI in which to tune PD control gains. In this exercise, you will have to implement your own PD controller to control the height of a quadrotor, as well as tune its gains.

Before starting on this programming exercise, we strongly recommend watching the video lectures, completing the review questions for the associated topics, and reading through this handout.

To get started, you will need to download the starter code and unzip its contents into the directory in which you wish to complete the exercise.

2 Quadrotor Simulator

We utilize one of MATLAB's ODE solvers, called `ode45`, to simulate the behavior of the quadrotor.You can read more details at Mathworks or other online resources. We then use the function `plot/plot3` to help visualize the current state of the quadrotor at each time step. You may take a look at file `height_control.m` for the simulation code.

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 supplementary segment "Supplementary Material: Getting Started With the First Programming Assignment" walks through these steps. The starter code for the controller (`controller.m`) produces robot inputs which are all zero thrust and thus the quadrotor falls due to gravity.

3 PD Controller

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

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

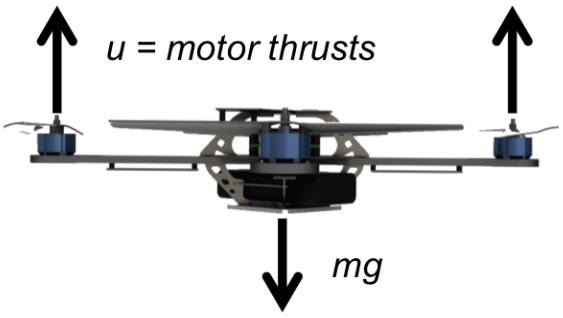

Figure 1: 1D quadrotor model.

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 $(z, z_{\text{des}}, \dot{z}, \dot{z}_{\text{des}})$.

4 Assignment

4.1 Files included in this exercise

[$\star$] `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.

`submit.m` - A script to be called for generating the submitted files

`evaluate.p` - Evaluation script to be called by `submit.m`

`fixed_set_point.m` - Step response function.

`utils` - Helper functions for quadrotor simulator

$\star$ indicates files you will need to implement

4.2 Tasks

You will need to first implement a PD controller for height control of the quadrotor. Then, tune the proportional gain (K_p) and derivative gain (K_v) in the file `controller.m` until the quadrotor converges quickly and smoothly to a step response input.

4.3 Submission and Grading

To submit your results to our server, you need to run the command `submit` in your MATLAB command window. A script will then evaluate your controller on two test cases and generate output files (files with the type `.mat`) to be uploaded to the web UI. There will be one output file for each test case. 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. Please note that the step response is different from the exercises you have already completed. Thus, while using your position and derivative gains from a past assignment might be a good place to start, you might need to tune them further to successfully complete this assignment.

Part	**Submitted Files**	**Points**
Hover Control	hover.mat	10
Step Response	step.mat	20
Total Points		30

You may submit your results multiple times, and we will count only the highest score towards your grade.

5 Programming Hints

5.1 Accessing states and robot parameters

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

5.2 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

5.3 Computing feedforward acceleration (optional)

The reference acceleration, $\ddot{z}_{\text{des}}$, is not given explicitly. you may set it to zero and still get a working controller. However, since we have the the desired velocity over time and we have the time step (it is 0.01 seconds, per `height_control.m`), then we may estimate it as follows:

$$\ddot{z}_{\text{des}} \approx \frac{\dot{z}_{\text{des},k} - \dot{z}_{\text{des},k-1}}{\Delta t} \tag{1}$$

where $\dot{z}_{\text{des},k}$ and $\dot{z}_{\text{des},k-1}$ are the desired velocities at times k and $k-1$, respectively, and Δt is the time step. However, in order to do compute this quantity, we must have a way of storing $\dot{z}_{\text{des},k-1}$ for use in our `controller.m` function. Persistent variables are used for this purpose. Persistent variables are variables declared in a function whose values remain stored in between function calls. The persistent variable `vel_des_prev` can be declared:

```
persistent vel_des_prev;
```

One can then check whether a value has been assigned to this variable using the `isempty` function. If a value was not assigned, then you can assign one:

```
if isempty(vel_des_prev)
       vel_des_prev = 0;
end
```