ECE 8 - Lab 5: PID Controller Introduction

November 30, 2022

1 Introduction

The proportional-integral-derivative (PID) control is the most common control method used in industry due to its functional simplicity. PID controllers have the goal of taking some error in our system and reducing it to zero. PID control consists of three basic coefficients – proportional(P), integral(I) and derivative(D) – to find optimal response. These parameters can be weighted, or tuned, to adjust their effect on the process. In this lab, you will be introduced to the PD control, which uses only P and D parameters of the PID control, for controlling the motion of a quadcopter.

You will be taking data for an experiment that will explore PID tuning (in this case PD tuning), a important skill for implementing this control model, and writing a report about your experience.

1.1 Lab Objectives

By the end of this lab you should be able to:

1. Understand PD control.
2. Identify the states of a quadcopter that are being used to design a PD controller.
3. Identify the parameters of the proposed PD controllers and know how to adjust them to achieve optimal results

1.2 Necessary Files, Tools, and Equipment

- MATLAB
- Phone or camera to take video

1.3 Grading Rubric

- Exercise 1, 2, and 3 are worth 4 points total.
- Exercise 4 is worth 5 points.
- Exercise 5 is 4 bonus points
- Properly commented code is worth 1 point.

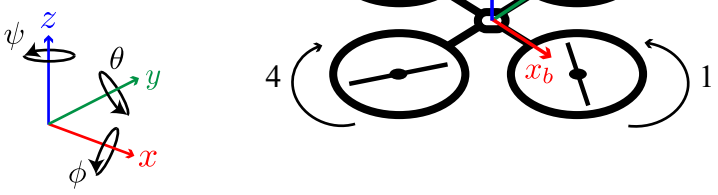
1.4 Due Date

This lab is due at 12/09/2022 at 6pm!

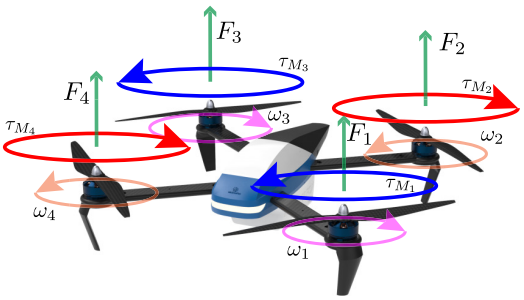
2 Controller Design

2.1 Quadcopter Model

Our goal is controlling the motion of a quadcopter to reach the target position. We first understand the model of a quadcopter to develop proper methods to calculate appropriate control inputs for stabilization and trajectory control of the quadcopter.



(a) Inertial reference frame and body reference frame of a quadcopter.



(b) Angular velocities, torques, and forces created by four rotors in a quadcopter's body frame.

Figure 1: Reference frames and angular velocities, torques, and forces created by four rotors.

We recall coordinate systems (i.e., *inertial* frame and *body* frame) and the state variables of a quadcopter (Figure 1). A quadcopter has six *degrees of freedom*; namely, three degrees of *translational* motion x, y, and z and three degrees of *rotational* motion ϕ(roll), θ(pitch), and ψ(yaw). The absolute linear position of the quadcopter is defined in the *inertial* frame X,Y,Z-axes with ξ, and the angular position is defined in the *inertial* frame with three Euler angles η as follows:

$$\xi = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \eta = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}, \quad q = \begin{bmatrix} \xi \\ \eta \end{bmatrix}. \tag{1}$$

Here, we note that the *inertial* frame is not moving with the quadcopter, but the *body* frame is moving with the quadcopter since it is attached to the quadcopter and the origin of the *body* frame is in the center of mass of the quadcopter. The coordinates can only be compared in the same coordinate systems (i.e., either *inertial* or *body* frame) and the coordinates in both coordinate systems are transformable from one to the other using a transformation matrix. For example, if we have the position of the quadcopter in the *body* frame, we transform the position into the *inertial* frame to compare the position of the quadcopter with the position of the target in the *inertial* frame. For convenience, the orientation of the *body* frame is the same as the orientation of the *inertial* frame at the initial position.

The angular velocities, torques and forces created by the four rotors (numbered from 1 to 4) in Figure 1. The angular velocity of rotor i, denoted by ω_i, creates force F_i in the direction of the rotor axis.

$$F_i = k_F \omega_i^2, \tag{2}$$

where k_F is a lift constant. The combined forces of rotors create thrust F in the direction of Z-axis in the *body* frame. The state τ_B consists of the torques τ_ϕ, τ_θ, and τ_ψ in the direction of the corresponding angles in the *body* frame

$$F = \sum_{i=1}^{4} F_i = k_F \sum_{i=1}^{4} \omega_i^2, \tag{3}$$

$$\tau_B = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} \ell(F_4 - F_2) \\ \ell(F_3 - F_1) \\ \ell(F_1 + F_2 + F_3 + F_4) \end{bmatrix}, \tag{4}$$

where ℓ is the distance between the rotor and the center of mass of the quadcopter.

2.2 PD Controllers

As explained in Lecture 17, the general form of the PD controller is given by

$$\begin{aligned} e(t) &= q_d(t) - q(t), \\ u(t) &= K_P e(t) + K_D \dot{e}(t), \end{aligned} \tag{5}$$

where $u(t)$ is the control input, $e(t)$ is the difference between the desired state $q_d(t)$ and the present state $q(t)$, and K_P and K_D are the parameters for the proportional and derivative elements of the PD controller. The main process of the PD controller is to reduce the errors between $q_d(t)$ and $q(t)$.

In the following, as an example of PD control design, we introduce PD controllers that are being used for your quadcopter in CoppeliaSim. Using the difference between the desired position and the current position of the quadcopter, the total stabilization of the quadcopter is achieved. The required force corresponding to each rotor F_i is determined by

$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} = f \begin{bmatrix} 1 - \alpha + \beta + \gamma \\ 1 - \alpha - \beta - \gamma \\ 1 - \alpha - \beta + \gamma \\ 1 - \alpha + \beta - \gamma \end{bmatrix}, \tag{6}$$

where f is the control input for the total trust on Z-axis in the *inertial* frame and the control input α, β, and γ is the θ, ϕ, and ψ correction, respectively. Each control input is composed by PD control.

$$\begin{aligned} f &= f' + K_{P,f} e_z + K_{D,f} \dot{e}_z + (\star), \\ \alpha &= (K_{P,\theta} e_\theta + K_{D,\alpha} \dot{e}_\theta) + (K_{P,y} e_y + K_{D,y} \dot{e}_y), \\ \beta &= (K_{P,\phi} e_\phi + K_{D,\phi} \dot{e}_\phi) + (K_{P,x} e_x + K_{D,x} \dot{e}_x), \\ \gamma &= (K_{P,\psi} e_\psi + K_{D,\psi} \dot{e}_\psi), \end{aligned} \tag{7}$$

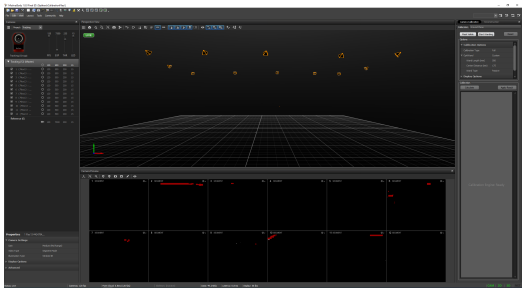
where f' is the required thrust to lift the quadcopter for each rotor, $(\star)$ indicates a term related to a tuning constant, and $K_{i,j}$, e_j are the controller parameters and errors, respectively, for $i \in \{P, D\}$, $j \in \{f, x, y, z, \phi, \theta, \psi\}$.

3 Lab Information

3.1 E2-206

This section will tell you all you need to know in order to complete the Exercises and successfully run an experiment involving quad copters. E2-206 is a lab space under the Hybrid Systems Lab here at UC Santa Cruz. We utilize a array of Infrared (IR) cameras from OptiTrack (Figure: 2a) in order to have sub millisecond recordings of the positon and orientation of rigid bodies in a area of space.

(a) OptiTrack Camera



(b) Screenshot of Motive

Figure 2: Overview of the hardware and software for tracking robotic systems

We use a program called Motive in order to communicate with OptiTrack and from Motive (Figure 2b), we can utilize its API with MATLAB in order to save the incoming stream of data. A few things to keep in mind about the data coming from Motive.

- The origin point (0,0), is where the duct tape "X" is in the room, and the rest of the field represents the four quadrants
- The positional data points are in the measurement of millimeters, whereas the orientation is represented in degrees.

3.2 Lab Procedure

Your objective is to take data in E2-206, there are set procedures in order to ensure everyone is safe during the experiment, as well as take effective data.

Figure 3: Photo of Drone

1. The drones have 3.7 V Battery, before you start the experiment, make sure the battery reads 3.7V.

2. Place the drone in the origin with the "white arrow" facing away from the computers, shown in Figure 3.

3. Turn on the drone by plugging in the battery, make sure to have the connector in the correct orientation (Red wire goes to +, Black wire goes to -) or you will destroy the drone.

4. Press the reset button on the drone shown in Figure 3, and a red LED will flash, this indicates the sensors are calibrated.

5. Make sure one person has safety glasses and gloves to catch the drone in case the experiment fails.

6. Another person will start the experiment by running the MATLAB code.

7. Wait until the drone fully lands and the code fully executes, if the drone does unexpected behavior, have the person with gloves catch the drone and press the reset button.

8. Once a successful experiment has completed, save the workspace as a .mat file and transfer it to a usb drive or email it to yourself.

3.3 Source Code - PID Controller

Take a look at the project file we give you, this is what you will be using to run the experiments. Its the source files for controlling the drone in E2-206. You don't have to understand every line of code, but try to understand what it does objectively.

```
function [x_acc_d, pid_output, X_pid_err] = Xcontroller(X_pid, x_d, x, x_dot, dt, tau, k)
    MAX_ACC = 900; % Max acceleration using thrust=value(0,255) assuming thrust>=thrustTrim(140)

    %% ------------------- FIRST PID BLOCK -----------------------------
    % Gains
    K_p = 350;
    K_i = 5;
    K_d = 200;

    % Proportional
    X_pid_err.x_curr_error = x_d - x;
    pid_p = K_p * X_pid_err.x_curr_error;

    % Derivative
    X_pid_err.deriv = -K_d*x_dot; % taking off minus sign
    pid_d = X_pid_err.deriv;


    %Integral
    X_pid_err.x_cumm_error = X_pid.x_cumm_error + K_i*dt*(X_pid_err.x_curr_error);
    pid_i = X_pid_err.x_cumm_error;

        % Integral saturations
%     if(pid_i > MAX_OUT)
%         pid_i = MAX_OUT;
%     end
%     if(pid_i < -MAX_OUT)
%         pid_i = -MAX_OUT;
%     end

    % Store previous error
    X_pid_err.x_prev = X_pid_err.x_curr_error;

    % Final PID result
    pid_output = [pid_p, pid_i, pid_d];

    x_acc_d = pid_p + pid_i + pid_d;
    x_acc_d = min(max(-MAX_ACC, x_acc_d), MAX_ACC);


end
```

Figure 4: PID Controller in Xcontroller.m

From Figure 4, we can see from the implementation of the controller, we can identify x_d as part of the error calculation before multiplying the Proportional Gain constant. The controller is designed with the reference of x_d (as you can see is one of the inputs for the function) to be the equilibrium

point for the x-axis. This specific controller is only for the x-axis, the entire code-base also implements y, and z PID blocks which are very similar to the one above. For the upcoming exercises, you will only tune proportional and derivative gains.

4 Exercises

There are two exercises to this lab, one is taking the data, the other is the analysis in MATLAB.

1. Form teams of 6, and pick a time slot during lecture time on Thursday. You will have approximately 15 mins time slots to tune the PD controller. You must do Exercise 4 and 5 on your own.
   (a) The location of where the experiments are taking place is E2-206
2. As you run experiments, tune the Z-axis proportional (K_p) and derivative gain (K_d) (which is located in Zcontroller.m) to see how the drone behaves. Change the values and see if you can get the drone to fly as smoothly as possible.
3. After conducting the three experiments listed below, pick your best one and save the workspace, make sure you also take a recording of the flight on a device.
   (a) Experiment 1: (P: 300, D: 50)
   (b) Experiment 2: (P: 50, D: 150)
   (c) Experiment 3: (P: 60, D: 30)
4. Use MATLAB to plot the following graphs with the data from the best experiment your team recorded:
   (a) a 3D Plot of the complete path the drone takes during the experiment
   (b) a figure with the subplots of x,y and, z position vs. time
   (c) a figure with the subplots of the yaw, pitch, and roll vs. time
5. **Optional (4 bonus points):** A written analysis of what you observed, and answering the following questions:
   (a) What is the equilibrium point for the drone seen in the experiments?
   (b) How does changing P and D affect the drone?