EECS C106B / 206B Robotic Manipulation and Interaction **Spring 2020**

Lecture 6: Linear Practical Control Methods

Scribes: Katelyn Biesiadecki, Shreyas Agarwal

6.1 Lecture Overview and Motivation

So far, we've learned several control techniques for linear and nonlinear systems, such as state feedback control and control Lyapunov functions. However, there are some practical considerations that we haven't taken into account yet: how do we actually choose our gains, especially in more complicated systems? How do we account for not having a perfect model of our system dynamics? In this lecture, we will cover the following techniques for mitigating these problems:

1. Cascaded Control
2. PID Control
3. Gain Scheduling

Recall from our previous lectures, the goals of control:

- **tracking**; follow a trajectory
- **robustness**; reject disturbances and noise
- **speed**; converge within a time constraint
- **stability**

These are the considerations that we'll keep in mind when implementing our control techniques.

6.2 Tuning a Linear Controller

Given a driven system, we can define our control input u as follows:

$$\dot{x} = Ax + Bu$$
$$u = -kx$$

Methods of tuning the gain, k, of this linear controller include the following:

1. **Pole Placement:** $A_{CL} = A - Bk$
   Set k to get desired (stable) eigenvalues, which correspond to the poles of your transfer function.
2. **Linear Quadratic Regulator (LQR):**
   Specify a cost function (matrices Q, R) to minimize. You can use the Matlab command, lqr, to do this.

3. **Randomly Picking Things Until It Works™:**
   Ultimately, you'll still have to manually tune your gains.

Both pole placement and LQR, however, require *perfect model knowledge* of A and B. (And the last method isn't much of a "method.")

6.2.1 Controller Tuning

$$\begin{bmatrix} Y \\ U \end{bmatrix} = \begin{bmatrix} \frac{PC}{1+PC} & \frac{P}{1+PC} & \frac{-PC}{1+PC} \\ \frac{C}{1+PC} & \frac{-PC}{1+PC} & \frac{-C}{1+PC} \end{bmatrix} \begin{bmatrix} R \\ D \\ N \end{bmatrix}$$

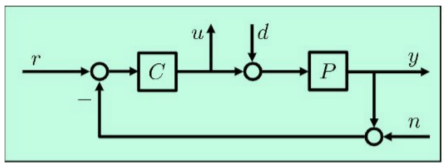

Figure 6.1: Transfer Function Representation of Input-Output Relations

Note that this example uses transfer functions, which characterize the relationship between the output, Y, the control input, U, and the reference signal, disturbance, and noise (R, D, and N, respectively). If the system's response to noise is exactly the opposite of that of the reference signal, the noise signal will be adversely amplified along with your reference signal. This indicates the trade-off between noise rejection and (reference) tracking speed.

However, since transfer functions are represented in the frequency domain, we can try to design our controller to filter out this noise and allow control signals to pass based on their frequencies. To do this, you can construct a bandpass filter that attenuates higher-frequency noise frequencies (e.g. 60 Hz) and amplifies lower-frequency control signals.

6.3 Cascaded Control

In **cascaded control**, we design our controller by breaking up a more complex system with several controllable states into a 'cascade' of simpler linear systems. This makes it easier to choose gains that can influence a subset of the states. Two common linear systems for modeling real-world behavior are:

1. **First order:** $\tau\dot{x} + x = k_{dc}u$, where τ is the time constant, k_{dc} is the DC gain
2. **Second order:** $\ddot{x} + 2\zeta\omega_N\dot{x} + \omega_N^2 x = \omega_N^2 k_{dc}u$, where ζ is the damping constant, ω_N is the natural freq

In these systems, it is easy to adjust the parameters (e.g. damping ratio) and understand how the system will respond to this change. The caveat is that each stage of the cascaded control system needs to be "faster"

than the previous ones. In the first order case, this is the time constant; in the second order case, this is the natural frequency.

Example: Velocity Controller on Baxter. In Project 1b, we'll be implementing a velocity controller for Baxter – why will the velocity controller likely perform better than the torque controller in this project?

ANSWER: It's less complicated. It's harder to tune gains for a second order controller (acceleration). As long as your application doesn't require really fast motions (in which case, the torque controller would be more useful), the velocity controller will be sufficient.

6.4 PID Control

"*How do you actually control 1st and 2nd order systems?*
Just throw PID at it."

— Valmik Prabhu

$$u = K_p e + K_d \dot{e} + -K_i \int_0^t e \, dt$$

Unlike the previous nonlinear control techniques we've learned so far – linearization, control Lyapunov, feedback linearization, and model predictive control – a proportional integral derivative controller (PID) is entirely model-free. It only takes the state error into account, e.g. $e(t) = q_d(t) - q_a(t)$. By augmenting these techniques with PID, we can correct the error due to the unknown or unmodeled parts of our system dynamics (as long as the system model brings the state close enough to equilibrium).

6.4.1 Augmented Torque Control Law

$$\tau = M\ddot{q}_d + C\dot{q} + G - K_p e - K_d \dot{e}$$

As mentioned above, we can augment our nominal torque control equation with PD to correct leftover errors (without potentially introducing instability with the integral term, in this example).

6.4.2 Quasi-stable approximation

Under the quasi-stable approximation, we assume that the above $C\dot{q} = 0$, or V = 0. By ignoring the velocity term, we're assuming that at any given instant of time, the system is basically static (or quasi-stable). This approximation fails if your system is moving very quickly.

6.5 Gain scheduling

In **gain scheduling**, we design our controller to have different gains in different regions of your configuration space and tune within those regions. In this architecture, your controller gains will change depending on the value of a "scheduling variable" (some measure of the system state or output). This is useful if one set of gains is insufficient for controlling the system under all conditions.

Be careful at boundaries between these regions: if your gain rapidly changes when crossing over, it could cause your system to become unstable. The solution is to try and nicely interpolate the gains between these boundaries.

6.6 PID Control

6.6.1 The Proportional Term

Term that does most of the work. Causes oscillations. A virtual spring, pulling the system towards the desired state.

$$\ddot{e} = -K_p e$$

Increasing K_p decreases rise time and increases system oscillation.

6.6.2 The Derivative Term

Virtual damper, stopping system from changing too quickly.

$$\ddot{e} = -K_d \dot{e}$$

Increasing K_d stabilizes response, decreases overshoot and ringing, and increases rise time.

A problem with derivative control is that it can potentially destabilize system due to noise amplification. Most of the time we don't know the actual derivative and so need to approximate it with samples:

$$\dot{e} \approx \frac{e(t) - e(t - \delta t)}{\delta t}$$

Unfortunately, this is a high pass filter which can amplify sensor noise. One fix is to apply low pass filter on top of the measurements but this decreases effectiveness of derivative term.

6.6.3 The Integral Term

Controlling derivative of control input.

$$\ddot{e} = -K_i \int_0^t e \, dt \;\Rightarrow\; \dddot{e} = K_i e$$

Increasing K_i rejects constant disturbances and can destabilize the system.

Term	Proportional Term	Derivative Term	Integral Term
Equation	$\ddot{e} = -K_p e$	$\ddot{e} = -K_d \dot{e}$	$\ddot{e} = K_i \int_0^t e dt$
Physical analog	Virtual spring	Virtual damper	n/a
Increasing K	Decreases rise time Increases system oscillation Increases rise time* Destabilizes system?	Stabilizes system response Decreases overshoot	Rejects disturbances Destabilizes system?

6.6.3.1 Windup

Windup is a problem which can cause destabilization. Windup occurs when the control input saturates, i.e. hits a physical limit. For example, if a drone is flying up to a desired height of 50m, then the motors will get the command to increase speed (rpm). Eventually, the integral term may give a command such as 2000 rpm but the motors can only spin at a max of 1000 rpm. Then, as the drone nears and passes above 50 m, the error decreases causing the integral term to begin to drop. However, it is slowly dropping from a value of 2000 rpm so the drone will continue to fly past 50 m at a rate of 1000 rpm and keep flying upwards until the command drops below 1000 rpm at which point the motors then begin to slow down. Thus, the inability for the motors to achieve the command causes windup and can cause the system to become unstable. The picture below demonstrates the above discussion:

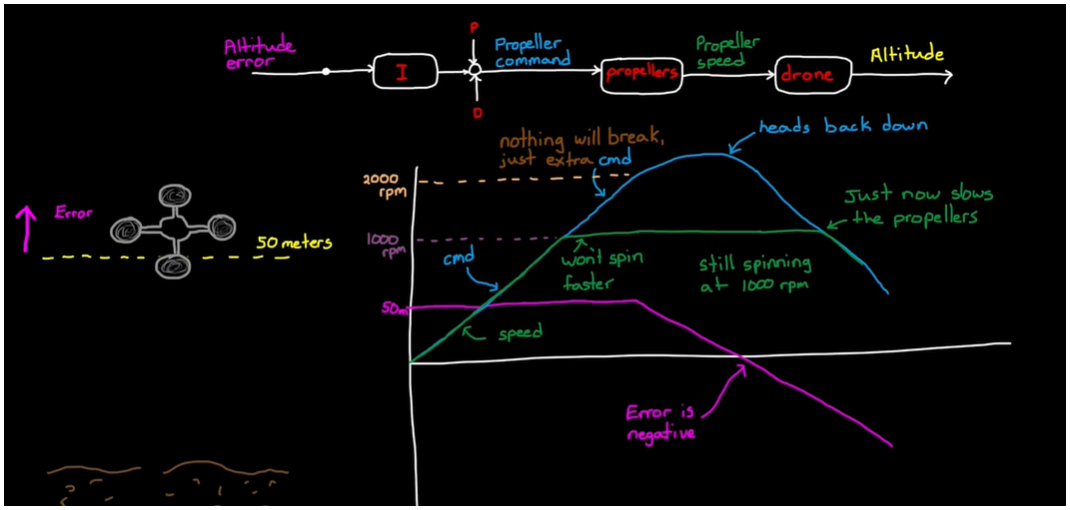

Figure 6.2: Image taken from https://www.mathworks.com/videos/understanding-pid-control-part-2-expanding-beyond-a-simple-integral-1528310418260.html

Some fixes for the windup issue include the following:

- Finite horizon integrator
- Weighted horizon integrator
- Bounds on the integral term
- Turn off the integrator when the error is high

6.7 Feedforward Control

The feedforward/"ballistic term" (K_{ff}) does not depend on error.

$$u = K_{ff} + K_{fb}\, e$$

The feedforward term does most of the work while the feedback term provides small corrections. The benefit of feedforward is that it doesn't result in any oscillations and improves system response without causing stability issues.