EE106A Discussion 10: Control

1 Intro to control of linear systems

Now that we've learned how to represent the dynamics of systems we can construct controllers to make these systems follow joint angle trajectories. We start with linear time-invariant systems, which are the easiest to analyze and control. For these systems, the dynamics have the form

$$\dot{x(t)} = f(x) + g(x, u) = Ax(t) + Bu(t) \tag{1}$$

Before introducing a controller, let's check the behavior of the system $\dot{x} = Ax$. As we've seen previously, the solution to this ODE is $x(t) = x(0)e^{At}$. It's possible to prove that if all of the eigenvalues of A are strictly negative (i.e., their real parts are < 0), then $x(t) \to 0$ as $t \to \infty$ for any initial condition $x(0)$. This property is called "global asymptotic stability." If that's what we want, then there's no need to apply a control input, we can just leave the system alone!

Problem 1: *We define the equilibrium point of a dynamical system to be a point x_e such that $\dot{x(t)} = 0$. What are the equilibrium points of equation 1 if $u = 0$ and the system is asymptotically stable?*

1.1 Controllability

If the system in equation 1 is not stable, or if we want to force it to follow a particular trajectory that isn't simply staying at an equilibrium point, we can use the control input u to change how the dynamics behave. The first question we might ask is whether it's possible to choose control inputs to make a particular system achieve the trajectory we want. It turns out that for LTI systems, there's a simple test we can use that's based on the "controllability matrix" Q:

$$Q = \begin{bmatrix} B & AB & \dots & A^{n-1}B \end{bmatrix} \in \mathbb{R}^{n \times nn} \tag{2}$$

If and only if $\text{rank}(Q)$ is equal to n, the system is "completely controllable," and we can choose a u to achieve *any* trajectory!

Problem 2: *What kind of B matrix will make **any** LTI system completely controllable? What would the corresponding control inputs look like in the dynamics?*

1.2 Linear state feedback

So we have a controllable system - great, how do we choose the actual control inputs? First, let's define the error $e(t) = x_d(t) - x(t)$ for some desired trajectory $x_d(t)$. Plugging this into our system, we get

$$\begin{aligned} -\dot{e} &= \dot{x} - \dot{x}_d \\ &= Ax + Bu - \dot{x}_d \\ &= A(x_d - e) + Bu - \dot{x}_d \\ &= A(-e) + Bu + (Ax_d - \dot{x}_d) \end{aligned}$$

We can see that these "error dynamics" are similar to a regular linear system, but with an extra affine term that depends on the desired trajectory. The simplest case is when we want the system to stay at an equilibrium point - the affine term disappears (check!). We want to choose u so that the error e goes to 0. If we choose a *linear state feedback policy* $u(t) = K_p e(t)$, we can see that the error dynamics become

$$\begin{aligned} -\dot{e} &= A(-e) + B(K_p e) + (Ax_d - \dot{x}_d) \\ &= (A - BK_p)(-e) + (Ax_d - \dot{x}_d) \end{aligned}$$

If the affine term is 0, we can choose the K_p so that the eigenvalues of $A - BK_p$ are strictly negative, which as we saw above will force the error to go 0 asymptotically! If the system is completely controllable, it's guaranteed that we can find a K_p that satisfies this. Because the control we're providing here is proportional to the current error $e(t)$, this is called a proportional or P controller.

Problem 3: *Let's say you want to force the system to rest at a point x_d which is not an equilibrium point. Will the state error converge to 0?*

2 Linearization

Most interesting dynamical systems are unfortunately not linear, which makes analysis and control much more difficult. Fortunately, we can often obtain a good linear approximation of nonlinear systems in practice. Since the linearized system is only an approximation of the true nonlinear dynamics, we lose most of our nice stability and tracking guarantees, but it's still extremely helpful for understanding and controlling the system.

Problem 4: *You've found that the equations of motion for a 2-joint robot are given by* $\Upsilon = M(\theta)\ddot{\theta}$*, where the manipulator inertia matrix* $M(\theta)$ *is given by* $M(\theta) = \begin{bmatrix} m\theta_1^2 & 0 \\ 0 & m\theta_2^2 \end{bmatrix}$*. You decide to provide torque control to both joints, and there are no other external forces. What is the linearized system?*

3 PID Control

We can often do better than P control by adding additional terms to our feedback policy. The full PID controller has the form

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \dot{e}(t) \tag{3}$$

Where the state error is defined as $e(t) = x_d(t) - x(t)$ and the velocity error is defined as $\dot{e}(t) = \dot{x}_d(t) - \dot{x}(t)$. These error terms encode how far away the system state is from the desired trajectory at any given time. K_p, K_i, and K_d are called your *control gains* and are each $n \times n$ diagonal matrices of weights. By varying these gains, you can change how the controller responds to the various error terms. This is called *tuning* the controller.
Note that while we developed the P controller in the context of linear system theory, where we can use the dynamics model to calculate a K_p that guarantees trajectory tracking, both the P and the PID controller can actually be used with *any* dynamical system. In the following sections we'll go into more detail about the terms we've added to equation 3.

3.1 The Derivative Term

The derivative term is the stabilizing term in a PID controller. Essentially what the D term does is add a damper (with viscosity K_d) between the state and its desired trajectory. A damper you see almost every day are the one-way dampers mounted to the tops of doors which stop them from closing too fast. Remember that the dynamics of a damper are $F = -b\dot{x}$, so the damper exerts higher force the faster the state is changing. Similarly the derivative term in a PID controller will exert more control input the faster the error changes (in the direction opposite the change).

The derivative term provides stabilization, decreasing oscillation and overshoot, but will also slow down the system response.

3.1.1 Implementation Pitfalls: Noise

The derivative term has a major implementation-related pitfall. Often, a system's sensors will only measure the position of the system x, not its velocity $\dot{x}$. This means that in order to estimate $\dot{x}$ you need to approximate the derivative $\dot{x}(t) \approx \frac{x(t)-x(t-\delta t)}{\delta t}$. If you've taken EECS 16B, you'll recognize this difference equation as a *high-pass filter.* A high-pass filter amplifies high frequency components of an input signal; the faster the input changes the higher the output will be.
The problem emerges when you consider sensor noise. Sensor noise tends to be rather low in magnitude, but high in frequency. A common frequency for noise is 60 Hz (the frequency of AC current in the US), but the resonant frequencies of any RLC circuits or mechanical components you're using are also quite common. A derivative term will naturally amplify any sensor noise and potentially cause instability in your controller. Thus, you generally want to apply a *low-pass filter* such as a moving average on your error derivative before feeding it into your derivative term. For example, the PID controller in lab averages the past three measurments of $\dot{e}$ before feeding it into the controller. The low pass filter will attenuate the high frequency signals such as sensor noise and give you a cleaner control signal.

3.2 The Integral Term

The integral term doesn't have as clear an effect as the proportional and derivative terms, nor does it have a neat physical analog. In order to appreciate the effect of the integral term, we need to consider the effect of disturbance forces on the system.
When there are constant disturbance or drift forces, it's impossible for a PD controller to settle at $x(t \to \infty) = x_d(t \to \infty)$. This is where the integral term is useful. By adding up error over time, the integrator can chip away at this constant disturbance and cause the state to converge to the desired trajectory. However, in doing so integral controllers tend to decrease system stability (even proving stability for a system with integrators becomes much harder). If tuned improperly, they can easily cause significant oscillation.

3.2.1 Implementation Pitfalls: Windup

Integral control also has a major implementation pitfall, which is called windup. Imagine that you're controlling the same mass-rail system as before, but now you have actuation constraints. Real-world actuators usually can't produce arbitrary torques or velocities at any given time. There are often constraints on the maximum velocity, acceleration, and jerk of a given actuator. If you start with a very high error, your proportional term will likely saturate your actuators, and you'll end up taking a longer time to reach the goal state. During this time, the integral term will keep increasing, even though this increase won't change u. When it finally hits the goal state the integral term will be very large, and the system will overshoot until the integral winds down again. Then on the way back the integrator will wind up again, this time in the opposite direction, and you'll get overshoot again. If the wind up gets bad enough, the integrator can easily destabilize the system, overshooting the goal more and more each time.

Problem 5: *Brainstorm some potential anti-windup methods. What are their drawbacks?*

4 Other control strategies

4.1 Feedforward Control

Imagine that we have a second order (force-controlled) system that we want to move from point A to B. We can generate a trajectory $x_d(t)$ that will navigate the system from A to B. Since this is a function of time, we can take the derivative to find $\dot{x}_d(t)$ and do so again to find $\ddot{x}_d(t)$ (we could also use Euler differentiation to approximate the derivative of a set of waypoints). If we have a good estimate of the system's inertia $M(x)$, we can plug in the trajectory to find $M(t)$, then find $F_d(t) = M(t)\ddot{x}_d(t)$. Now we have an estimate of the input force required at every time step, assuming that we start where the plan starts, and no disturbances occur. We can incorporate this information into our PID control law by adding a feedforward term

$$u(t) = u_f f(t) K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \dot{e}(t)$$

In this example $u_f f(t) = F_d(t)$. Feedforward control doesn't oscillate and can't cause instability (assuming that the system itself is stable). Because of this, we want the feedforward term to do a majority of the control work, and use our feedback terms for small corrections. A good feedforward term will *drastically* improve performance and require significantly less tuning, with essentially no downsides.

4.2 Feedback Linearization

Sometimes, we can do better than plain feedforward control when we're dealing with highly nonlinear systems, like robot manipulators. If you look at a general control-affine (the dynamics have an affine dependence on the control) nonlinear system

$$\dot{x} = f(x) + g(x)u$$

you have two nonlinear terms. The *drift vector* $f(x)$ incorporates all the internal forces in the system (coriolis and gravitational forces for example) while the *control vectors* $g(x)$ determine how your control inputs affects your dynamics. If we can figure out a set of inverse dynamics

$$u = a(x) + b(x)\dot{x}_d = -g^{-1}(x)f(x) + g^{-1}(x)\dot{x}_d$$

we can use this as our controller and exactly cancel out the nonlinearities of our system to get

$$\dot{x} = \dot{x}_d$$

It's not always possible to invert the dynamics like this (if you want to figure out if it's possible, you'll need to take a graduate nonlinear controls class), but it so happens that the dynamics of any open-chain robot manipulator are invertible. You'll examine the feedback linearization of these systems in your homework, and should you choose to take EECS C106B, you'll be implementing it on hardware.