C106B/206B Discussion 12: Semester Review

1 Introduction

We've covered a lot of ground this semester! Let's look back on the range of topics we've done and see how it all fits together.

1. **Systems and state:** We started talking about systems, which determine how the state evolves over time. This course focuses specifically on robotic systems, so the state usually comprises of some set of variables that represent physical qualities of the robot.
2. **Control:** Robotic systems usually have some way to provide input, allowing us to control how they behave. We know that linear systems are particularly nice to work with, so feedback linearization allows us to figure out how to set our inputs to drive our state in the way we want. We also discussed Lyapunov stability, a technique to check whether our state will remain bounded.
3. **Constraints:** Before we plan our path, we want to ensure that certain constraints are followed because our robot might not be able to reach certain places or velocities. These come in the form of Pfaffian Constraints on our velocity $\dot{q}$, which may also limit our positions if they are integrable (holonomic).
4. **SLAM:** The map of the space we are in might not be very accurate or even nonexistent. Also, the sensors on our robot's wheel's aren't great. We want to figure out the obstacles around us while keeping an accurate estimate of our machine's location, thereby simultaneously mapping and localizing - SLAM!
5. **Path Planning:** We now have 3 important pieces of our puzzle: the limitations of our robot, how to control it, and any obstacles around us. Now, we need to actually decide how to get to our goal! This is the problem of path planning. Optimization planners, RRTs, and sinusoids all return a series of inputs that take our system from our starting position to the goal. MPC builds on this by replanning every few time steps, adjusting the input as needed and accounting for error (hence the title of controller).
6. **Grasping:** A huge problem in robotics is how to actually hold things. Grasping covers the mathematics of ensuring our object remains properly secured and how we can move it around once it is.
7. **Reinforcement Learning:** On the frontier of robotics, data-driven methods apply machine learning to train our robots to behave in a desired fashion. This approach stems from the foundations established by optimal control, and many branches of research explore alternative formulations of this problem.

2 Systems

A system tells us how the state evolves over time. It is described using a combination of the state itself (the smallest set of variables to completely describe the current configuration), the input variables, and the output vector (what we control or measure with sensors).

Problem 1: The Van der Pol oscillator ~~is used in many circuits~~ is used in many electric circuits because of its stable oscillations. It has applications to biology as well, modeling flow in the aorta, which carries blood out of our heart. It follows the below model:

$$\ddot{x} - \mu(1 - x^2)\dot{x} + x = 0 \tag{1}$$

where μ is a scalar damping coefficient. By choosing a good set of state variables, write the above model in state space form.

$q = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$ $\quad \dot{q} = A q$ $\quad x = -\ddot{x} + \mu(1-x^2)\dot{x}$

$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} \mu(1-x^2) & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix}$

$\dot{q} \quad A \quad q$

3 Control

3.1 Feedback Linearization and Dynamic Extension

Nonlinear systems can be tricky to control. We might generate a trajectory where we know exactly how we want our state to evolve. That is, we know what we want our output values or their derivatives to be. However, figuring out the inputs that lead to these can be quite difficult because the system is not perfectly invertible. As a result, we use a technique called *dynamic extension*, which involves taking the derivatives of the output until our input values appear, in order to achieve feedback linearization.

This works for systems like the planar quadrotor:

$$\begin{bmatrix} \dot{y} \\ \ddot{y} \\ \dot{z} \\ \ddot{z} \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} \dot{y} \\ 0 \\ \dot{z} \\ -g \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ -\frac{1}{m}\sin\phi & 0 \\ 0 & 0 \\ \frac{1}{m}\cos\phi & 0 \\ 0 & 0 \\ 0 & \frac{1}{I_{xx}} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{2}$$

Our output f is $[y, z]^T$, the position of the planar quadrotor (We can use f instead of y to avoid confusion with the state variable). We know what we want our accelerations $\ddot{y}$ and $\ddot{z}$ to be (perhaps from path planning). However, we're not able to control our output directly with these inputs in the form of feedback linearization:

$$u = A^{-1}(y^d - b)$$

Therefore, we use dynamic extension, controlling the 4th derivative of the output. We can then integrate to ensure the position of the quadrotor will be where we would like it.

3.2 Lyapunov Stability

Lyapunov stability formally states that an equilibrium point x_e is stable if the state starts within some distance δ of that point and remains within ϵ of it for all time. (This is a weaker definition of stability because it is local, having a particular starting condition - a stronger version exists for global exponential stability).

The Basic Theorem of Lyapunov uses the idea of a positive definite *energy function* for the system $V(x, t)$ such that if its time derivative $\dot{V}(x, t) \leq 0$ for all t, then the origin of the system is locally stable.

In your homework, you constructed a *control Lyapunov function* to drive a quadrotor along a desired trajectory. It ensures error converges to 0 rather than the position of the quadrotor. An optimization problem can be used to find inputs over time while ensuring the derivative of the Lyapunov function remains negative (or remains below a negative value, a stricter condition that speeds up convergence).

Problem 3

Now, we will directly try to drive the position to the origin. Consider the following model for a unicycle model robot. The state is (x, y, θ) which represents the position of the center of the robot relative to some fixed origin along with its current heading. The control inputs are the linear velocity v and the angular velocity ω.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \tag{3}$$

In this problem, we will explore a technique called *point-offset* control for controlling unicycle model robots like the Turtlebot. Consider a point p attached rigidly to the robot at a distance δ from the center, in front of the robot. So, the position of p is given by:

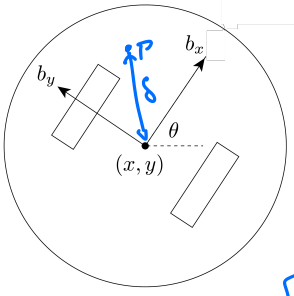
$$\begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} x + \delta\cos\theta \\ y + \delta\sin\theta \end{bmatrix} \tag{4}$$

Now consider the problem of driving the turtlebot to some neighbourhood of the origin. Instead of driving the turtlebot directly, we will instead attempt to control the robot so that the point p goes to the origin. Then, the turtlebot will be in a neighbourhood of radius δ around the origin. In the next few problems, we will develop a control law to drive p to the origin, and prove its stability.

1. Let the body frame axes of the turtlebot be $b_x = (\cos\theta, \sin\theta)^T$ and $b_y = (-\sin\theta, \cos\theta)^T$, as shown in the figure below. Show that

$$\dot{p} = v b_x + \delta\omega b_y \tag{5}$$

→ Velocity of point p wrt body frame axes of turtlebot



$$p = \begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} x + \delta\cos\theta \\ y + \delta\sin\theta \end{bmatrix} \longrightarrow \dot{p} = \begin{bmatrix} \dot{x} - \delta\dot{\theta}\sin\theta \\ \dot{y} + \delta\dot{\theta}\cos\theta \end{bmatrix}$$

$$= \begin{bmatrix} v\cos\theta - \delta\omega\sin\theta \\ v\sin\theta + \delta\omega\cos\theta \end{bmatrix}$$

$$= v\begin{bmatrix}\cos\theta \\ \sin\theta\end{bmatrix} + \delta\omega\begin{bmatrix}-\sin\theta \\ \cos\theta\end{bmatrix} = v b_x + \delta\omega b_y$$

✓

2. Say we apply the following feedback control law on the robot:

$$v = -b_x^T p, \qquad \omega = -\frac{1}{\delta} b_y^T p \tag{6}$$

Using the Lyapunov function

$$V = \frac{1}{2} p^T p \tag{7}$$

show that the point p converges asymptotically to the origin. Is the stability global?

Find velocity of p in terms of control law:

$$\dot{p} = (-b_x^T p) b_x + \delta\left(-\frac{1}{\delta} b_y^T p\right) b_y$$

$$\text{proj}_{b_x} p = \frac{b_x^T p}{\|b_x\|} b_x$$

$\dot{p}$ = negative projection of p onto axes of Turtlebot

$= -p$

$V = \frac{1}{2} p^T p \rightarrow$ PD function ✓

$\dot{V} = \frac{d}{dt}\left(\frac{1}{2} p^2\right) = \frac{1}{2} \cdot 2\, p\dot{p}$

$= -p^T p \rightarrow$ negative definite ✓

3.3 Control Barrier Functions

Control Barrier Functions account for safety in addition to simply control. They are used for critical systems where a controller must provably avoid entering danger zones. A constraint function $h(q)$ of the state encodes the safety region; if the function is positive, then the region is safe. We apply the following condition:

$$\dot{h} \geq -\gamma h$$

to ensure this remains true (assuming the system starts from a safe state). Then, we can solve for u (a function of $\dot{x}$) to determine the condition on the input. We can apply this condition when generating trajectories!

4 Constraints

We can't always hit the velocities or positions that we want it to for a particular system. As a result, the path we plan and the control law we create must follow some constraints.

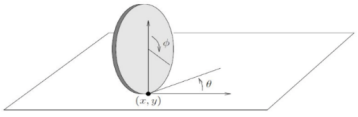
A Pfaffian constraint on the velocity is expressed as $A(q)\dot{q} = 0$. The velocities that we can reach are therefore in the nullspace of A - these vectors meet the given condition. These constraints are *holonomic* if they are integrable and thereby also form position constraints, where $h(q) = 0$.

The Lie bracket of 2 vector fields $f(q)$ and $g(q)$ is given by

$$\frac{\partial g}{\partial q} f(q) - \frac{\partial f}{\partial q} g(q)$$

If the nullspace of A is given by $g_{1\ldots n}$, then the span of the g vectors, their Lie brackets, and higher order Lie brackets is the Lie algebra. If the dimension of the Lie algebra equals the dimension of the state, our system is controllable.

Problem 4: Find the Pfaffian constraints of a rolling disk. Are these holonomic? Why or why not?



→ Roll w/o slipping

$\dot{x} = r\cos\theta\,\dot{\phi}$ — $\cos\theta$: x-direction; $\dot{\phi}$: speed of the wheel; r: scaling by radius

$\dot{x} - r\cos\theta\,\dot{\phi} = 0$

$\dot{y} = r\sin\theta\,\dot{\phi}$

$\dot{y} - r\cos\theta\,\dot{\phi} = 0$

Nonholonomic → the constraints don't restrict the position of disk

5 Path Planning

We've covered multiple different types of path planners. An optimization planner meets some constraints on the motion of our robot to avoid obstacles while calculating the best possible inputs to form our path. RRTs generate small motion primitives that are chained together to move our robot towards our goal. The sinusoidal planner uses the oscillation of the sine function to generate inputs that move each part of our state, one at a time.

Problem 5: In your project, you've seen that optimization planners tend to form the best path. Would they work just as effectively on a 7-DOF robot arm in an obstacle-rich workspace? Why or why not?

Yes... eventually

(Depending on computational resources, may take a while)

→ can use for ex. RRT

6 SLAM

If our environment is unknown, we want to map out our obstacles and keep an estimate of where our robot is. However, our sensor readings on both fronts will have noise, which is assumed to be Gaussian in nature. SLAM performs the tasks of generating a plot of our space and accounting for the errors in our measurements. The front end finds features of interest, and the back end eliminates noise with a Kalman Filter.

7 Grasping

Last semester, we started our discussion of robotic arm movement talking about *kinematics*. This deals with the different positions and angles our body frame can potentially reach. We then discussed *kinetics*, which deal with velocities and accelerations. This semester, we moved to dynamics for an arm - analyzing the relationship between forces applied on the body and its motion! A *wrench* follows the same kind of linear/angular form as twists:

$$\Gamma = \begin{bmatrix} f \\ \tau \end{bmatrix}$$

where f is a linear force component, and τ is a torque.

We used the Jacobian for wrenches as well. If we desire a particular torque in our body frame, we can use the matrix to determine the torques we would need to apply on each of the joints in our robot.

To discuss actually grabbing objects, we must combine multiple contacts together. These are the positions where our robot is grabbing our objects. To check whether the grasp is successful, we combine the contacts into a single frame in the form of a grasp map. We can then multiply this map by the amount of torque each contact is applying to determine the wrenches the grasp can resist. A grasp that can resist any wrench is in *force closure*. A *soft finger* grasp can resist torques as well as forces because of its distributed nature.

Many techniques exist to optimize grasp points so that an object is picked up effectively. These include metrics like Ferrari-Canny, gravity resistance, etc.

8 Optimal Control and Reinforcement Learning

Optimal control comes up with an input that is the theoretical optimal satisfying a cost function and given constraints. We've seen OC in many forms, including optimization and CLFs. If our cost function is quadratic and our system follows an affine difference equation form, we can apply dynamic programming to solve for the optimal input at every timestep, creating a linear quadratic regulator (LQR) controller.

Reinforcement learning applies the concepts of machine learning to the space of control. Many paradigms of reinforcement learning exist. On the most general level, examples of a particular task are given, and the computer must figure out how to best accomplish its goals. Variations of RL include a model-based approach (where the computer must learn to predict the next state given actions), actor-critic (involving two separate networks and a Q-function), offline learning (using a dataset that doesn't have examples of the final task at hand), etc.