Assignment 4: Introduction to Tracking Controllers

Tufts ME 149: Optimal Control For Robotics

Assigned: Feb 6 — Due: Feb 15

Introduction

[**ToDo:** *Finish this draft!*]

[**ToDo:** *The requirements for the assignment will remain unchanged, but I will add more clarification around the details. Final version will be posted before Thursday Feb 8. Please email me with any questions or comments regarding the assignment.*]

Part One: Cubic Spline Math

Cubic splines are commonly used to represent trajectories. Cubic splines are useful because they are easy to work with and can be constructed with a continuous first derivative (unlike linear splines). Rather than construct an entire spline, we will focus on a single cubic segment $x(t)$, shown below.

$$x(t) = \sum_{i=0}^{3} c_i t^i = c_0 + c_1 t + c_2 t^2 + c_3 t^3 \tag{1}$$

A cubic Hermite spline is a cubic spline that is fully defined by the value and slope at each knot point. These boundary conditions are given below.

$$x(0) = x_0 \qquad x(h) = x_h \tag{2}$$

$$\dot{x}(0) = v_0 \qquad \dot{x}(h) = v_h \tag{3}$$

1a: Given the boundary conditions and spline equation above, compute the spline coefficients c_i in terms of the boundary conditions.
1b: Cubic splines are often constructed with a constraint that the acceleration at each boundary is continuous. Compute the acceleration at each boundary: $\ddot{x}(0)$ and $\ddot{x}(h)$. You may write acceleration in terms of either the boundary conditions or the spline coefficients.
Note: You may perform these calculations either by hand or using the Matlab symbolic toolbox. If working by hand, then please scan your calculations (this can typically be done using a simple app on your phone) and then upload them as a pdf. If using the Matlab symbolic toolbox, them upload your derivation script. The script should print the solution to the command prompt; please copy this output directly into the comments of the script.

Part Two: Pendulum Swing-Up

Construct a piecewise-polynomial reference trajectory for a simple pendulum. The trajectory should be in the form of three pp splines: `qRef` (angle), `dqRef` (rate), and `ddqRef` (accel). The trajectory should move the system from the stable equilibrium (`qRef(0) = 0`) to the unstable equilibrium configuration (`qRef(T) = pi`). The initial and final rates should be zero. This "swing-up" trajectory should take 3 seconds to complete.

You may use any method that you like to construct the pp-spline, provided that it can be evaluated using the `ppval()` function in Matlab. A few possible options include: `spline`, `pchip`, `pwch`, and `ME149/codeLibrary/splines/pwqh`.

Make sure that your splines have *consistent* derivatives: the the rate reference should be the derivative of the angle reference, and the acceleration reference should be the derivative of the rate reference. One way to achieve this is by constructing `qRef` first and then using the `ME149/codeLibrary/splines/ppDer` function in the code library to compute the derivatives for you.

Design a proportional-derivative (PD) controller to track the set of reference trajectories. You may use any method that you like to tune the gains. The function `ME149/codeLibrary/control/secondOrderSystemGains` in the code library might be useful, although you are not required to use it. In addition to the feed-back controller, compute the feed-forward reference torque using the inverse dynamics for the pendulum.

Use the simple pendulum dynamics model (`ME149/codeLibrary/simplePendulum/`) from the code library, with `param.freq = 1.0` and `param.damp = 0.0`. The library provides functions for both forward and inverse dynamics.

It is often desirable to minimize the "actuator effort" along a trajectory. There are many models for actuator effort, but here we will use the torque-squared model. Note that the objective function $J()$ is a *functional*: it is a function that takes another function as input.

$$J\big(u(t)\big) = \int_0^T g(\tau)\, d\tau = \int_0^T u^2(\tau)\, d\tau \tag{4}$$

This type of objective function is a path-integral. When doing trajectory optimization, the correct way to evaluate a path integral is to solve it using the same method that you use for propagating the dynamics. This ensures that your results are self consistent, which will be important later. To do this, you construct a modified dynamics function at the start of your simulation, where the integrand $g()$ and the system dynamics $f()$ are stacked to form a single combined dynamics function that is passed to the simulator.

2a: Compute the torque-squared objective function for your reference trajectory. This should be done by calling your simulation function with the `dynFun` $= g()$. **2b:** Compute the torque-squared objective function for the simulation of the full closed-loop system using the controller (with both feed-back and feed-forward terms). Introduce an intentional perturbation to the initial conditions: the initial angle for the simulation should be 0.2 radians. This will allow the simulation to test the performance of your controller.

[**ToDo:** *Requirements for plots and other output for submission.*]

Part Three: Double Pendulum Swing-Up

Same as the single pendulum swing-up, but now apply those techniques to a more complicated system. The initial conditions now start with both links of the pendulum in stable equilibrium (minimum potential energy) and end with both links in the unstable equilibrium (maximum potential energy). The path integral can be applied to both torque (which are added), and you can simply write a PD controller for each joint. As with the previous assignment, set all parameters in the double pendulum dynamics (link lengths, link masses, and gravity) to one.

Implementation Details

You should be able to create one (or more) functions that work for both the single and double pendulum. For example, the objective function should work for any number of torque inputs.