Homework 2:
LQG and Trajectory Optimization

E. Todorov, AMATH/CSE 579

due May 24

1 Acrobot dynamics

The dynamical system we will be working with is a deterministic double pendulum, also known as the acrobot. See Figure 1. Only the second joint (elbow) is actuated. The dynamics are in the form

$$\dot{x} = a(x) + B(x)\,u$$

$x = \left[\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2\right]^T$ where θ_1, θ_2 are the two joint angles; u is the scalar control signal (torque acting on the second joint).

The following Matlab code (acrobot.m) computes the dynamics:

```
function [xdot, a, B] = acrobot(x, u)
friction = 0.1;
g = 9.8;
M = [3 + 2*cos(x(2)), 1+cos(x(2)); 1+cos(x(2)), 1];
c1 = x(4)*(2*x(3)+x(4))*sin(x(2)) + 2*g*sin(x(1)) + g*sin(x(1)+x(2));
c2 = -x(3)^2*sin(x(2)) + g*sin(x(1)+x(2));
a = [x(3:4); M \ [c1-friction*x(3); c2-friction*x(4)]];
B = [0;0; M \ [0;1]];
xdot = a + B*u;
```

On the course website you can also find the file testacrobot.m which shows an animation of the passive dynamics. Note that the dynamics have two parameters: the friction and gravity coefficients. Feel free to experiment with different values.

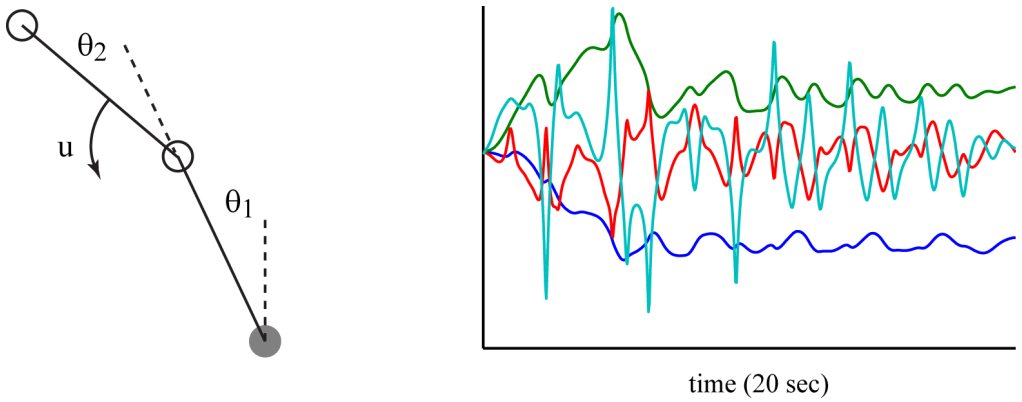

Figure 1: Illustration of the acrobot system. The plot on the right shows joint positions and velocities over time for the passive system.

2 Tasks

We want to design control laws which make the acrobot stay in the upright position ($\theta_1 = \theta_2 = 0$) with minimal control effort. Thus we will use the cost function

$$\ell(x, u) = \frac{r}{2}u^2 + 1 - \exp(k\cos(x_1) + k\cos(x_2) - 2k)$$

where r sets the relative importance of conserving control energy vs. staying upright, and k is the width of the position cost. You should adjust r, k so as to generate interesting behaviors. Note that the angles should be kept inside the interval $[-\pi; +\pi]$. If they ever leave this interval you should add/subtract 2π.

We will consider two versions of the task. The first is a regulation task: keep the acrobot in the goal state assuming the initial state is near the goal state. The second task is harder: get the acrobot to swing up from an arbitrary initial state, and then keep it there.

2.1 Regulation near the goal state

To solve the regulation problem, linearize the dynamics around state $x_0 = [0, 0, 0, 0]^T$ and approximate it in the form

$$\dot{x} = A_0 x + B_0 u$$

Also compute a quadratic approximation to the cost function in the form

$$\ell_0(x,u) = \frac{r}{2}u^2 + \frac{1}{2}x^T Q x$$

Now that you have a linear system and a quadratic cost function, solve the corresponding continuous-time LQQ problem (using "care" in Matlab, and setting the noise covariance terms from the lecture notes to 0). This will give you a linear feedback control law in the form

$$u = Lx$$

which should make the linearized system go to the goal state from any initial state (test this numerically). But what about the actual nonlinear system? Plug in your linear feedback control law in the nonlinear dynamics:

$$\dot{x} = a(x) + B(x)Lx$$

and integrate the latter forward in time from a variety of initial states. Use one of Matlab's integrators (say "ode45"). For some initial states this linear control law will work well, while for others it will not. Find a way to characterize (numerically) the region of initial states around x_0 for which it works.

2.2 Control from any initial state

The control problem here will be defined as a finite horizon optimal control problem. The final cost is the same as the running cost but without the control term. You have to adjust the overall duration and time step; note that this is a complicated dynamical system, thus you need to use small time steps (around 10 msec) and large horizon (at least 3 sec). We want to control the system from any initial state, and not just from states near the goal. We will do this by optimizing a trajectory that is specific to each initial state x_0. This involves two subtasks: implementing the code that computes the gradient and calls an optimizer, and then adjusting parameters and initial conditions (see below) so as to make the optimizer find sensible solutions.

The objective function J being minimized is the total cost of the trajectory. Its argument is the sequence of controls $U = \{u_0, u_1, \cdots u_{N-1}\}$. To compute the total cost J, integrate the dynamics forward in time to obtain the state trajectory and then evaluate $\ell(x,u)$ at each point in time.

Note that J also depends on the starting state x_0, however this dependence should be made implicit because the optimizer should not change x_0. Use the discrete-time maximum principle from the lecture notes to compute the gradient $\partial J/\partial U$. Once you have a MATLAB function that computes both J and its gradient, call an off-the-shelf optimizer to minimize it. Try to compute the necessary derivatives analytically; finite-differencing is also fine but it may slow you down. You can use the MATLAB optimization toolbox, or minFunc by Mark Schmidt (Google it). Note that you can select different optimization algorithms, and which algorithm is best tends to be problem-specific. BFGS is usually a good starting point.

Now that you can optimize J automatically, in an ideal world you would be done. In the real world however you are not done, because hard optimization problems often require good initialization as well as continuation methods to guide them to the minimum. In this case it is not clear how to obtain good initialization. You can try different random initializations for U, perhaps smoothed over time.

If it turns out that random initialization of U is not sufficient, you can explore a continuation method. Continuation generally means solving an easier problem first, and using the solution as initialization for the harder problem. To use continuation, you have to define a continuum of problems such that your (hard) problem is at one end, and some easy-to-solve problem is at the other end. In this case, you can create a related but easier problem by allowing actuation at both joints. To create intermediate problems, you can still allow actuation at both joints but use a larger control cost at the first joint (shoulder). If you make this cost very large, the problem eventually becomes indistinguishable from our original problem with actuation only at the elbow joint. You may need multiple steps of continuation, gradually increasing the control cost on the shoulder joint.

In solving higher-dimensional problems it is important to visualize the results and get an intuition as to what the optimizer is doing. To plot the acrobot trajectories in static figures, you can create multiple subplots showing frames from the animation. You can also plot the state and control variables over time - to check that they change smoothly over time.

WHAT TO SUBMIT: your code, along with a PDF file showing figures of (hopefully successful) control and summarizing your observations. For the second part, test your trajectory optimizer with different initial states.