CS4610/CS5335: Homework 4

Out: 4/9/16, Due: 4/19/16

Please turn in this homework to Rob Platt via email on the due date. HW PA Q1, Q2, and Q3 should be submitted in the form of a set of three files named Q1.m, Q2.m, and Q3.m. All this should be zipped up into a single file and emailed to me.

Have a look at the accompanying zip file. Stub files for Q1.m, Q2.m, and Q3.m are provided to you. You should implement each of these. Once implemented, you should be able to run "hw4(X)" in order to run code for question "X". hw4.m is given to you and should not need to be modified. The only thing you need to do is to insert code into the stub functions in Q1.m, Q2.m, and Q3.m. NOT ALL OF THE STUB FUNCTIONS IN Q1.m, Q2.m, and Q3.m NEED TO BE MODIFIED. Please see the code.

PA Q1: In this question, you must implement a finite horizon discrete time LQR for the damped mass system described in class and illustrated in the course slides. The time horizon, T, and the A and B matrices that encode the system dynamics are already encoded in hw4.m and you don't need to change them. Also, the QT, Q, and R cost matrices as well as the initial state, $x0$, are already encoded in hw4.m. What you need to do is to implement two functions: FH_DT_Riccati and getControl as described in Q1.m. FH_DT_Riccati should do the Riccati equation recursion. It should return a cell array, $Pseq$, where each cell is a 4×4 P matrix and the cell index is the same as the time index ($Pseq\{i\}$ denotes the 4×4 P matrix at time i). getControl should calculate the control action when the system is in state x at time i.

PA Q2: Exactly the same as Q1 except that you should now implement a receeding horizon controller. Whereas in Q1 you were to find the optimal control for a fixed time horizon T, now you must execute a receeding horizon controller where you calculate a control action at each time step by optimizing over the *next* T time steps.

PA Q3: In this question, we are going to use the LQR framework in a new way. As we have studied it so far, LQR can generate large control inputs that can change quickly. For example, the control output calculated in Q1 and Q2 is very large on the first few time steps. It is sometimes desirable to find a trajectory that minimizes the *change* in control input rather than the magnitude of the control input itself. It turns out that we can use LQR to calculate these sorts of control policies as well. Suppose that we want to minimize a cost function of the form:

$$J(X,U) = x_T^T Q_T x_T + \sum_{t=1}^{T-1} x_t^T Q x_t + u_t^T R u_t + \Delta u_t^T \hat{R} \Delta u_t,$$

where the last term in the summation imposes a cost on change in control input. We can achieve this behavior by defining a new system:

$$\begin{pmatrix} x_{t+1} \\ u_{t+1} \end{pmatrix} = \begin{pmatrix} A & B \\ 0 & I \end{pmatrix} \begin{pmatrix} x_t \\ u_t \end{pmatrix} + \begin{pmatrix} B \\ I \end{pmatrix} \Delta u_t$$

with cost function

$$\begin{aligned} J(X,U) &= \begin{pmatrix} x_T \\ u_T \end{pmatrix}^T \begin{pmatrix} Q_T & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_T \\ u_T \end{pmatrix} \\ &+ \sum_{t=1}^{T-1} \begin{pmatrix} x_t \\ u_t \end{pmatrix}^T \begin{pmatrix} Q & 0 \\ 0 & R \end{pmatrix} \begin{pmatrix} x_t \\ u_t \end{pmatrix} + \Delta u_t^T \hat{R} \Delta u_t. \end{aligned}$$

You should ask yourself the following questions: what is the new state vector representation? What are the new "A" and "B" matrices? Use this new representation to calculate the optimal trajectory in this scenario. You need to create three functions in Q3.m: FH_ DT_Riccati, getControl, and getSyntheticDynamics. However, FH_DT_Riccati and getControl should be exactly the same functions as you created in Q1.m. The only new function is getSyntheticDynamics. This takes the underlying parameters of the system as input and produces as output the new modified parameters.