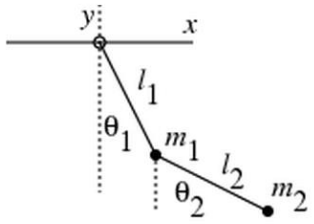
ME 454: Numerical Methods in Optimal Control

Final Project

Torque control of double pendulum

I was initially inclined to simulate the optimal control of a walking biped. I was not able to find a good paper that described the equations of motion of the system [1]. In the interest of time, I decided to model the simpler problem of optimizing the motions of a double pendulum, which can be assumed as a very simplified model of a leg. The power of control using optimal control theory is its ability to specify the control goal for both fully and under actuated systems, be it linear or nonlinear. What makes optimal control attractive is being able to analytically solve intractable control problems. Here, I am considering a fully actuated double pendulum, which does not harness the full computational power of optimal control theory. However, I still think this project was immensely valuable for my understanding of the algorithm. I ended up first investigating a double pendulum (mass focused at the end of the arm, without taking inertias into consideration), and then a two link RR robotic arm (mass concentrated at the center of the arm, taking into account the mass matrix, vector of Coriolis effects, and vector of gravitational forces). I ended up optimizing the computed-torque control law.

Dimensionless Double Pendulum with Mass Concentrated at the Ends:



Equations of Motion were derived by solving the Euler Lagrange from the following (full derivation can be found in my Mathematica code).

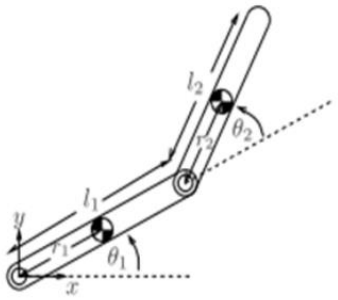
$$\mathrm{x1} = \mathrm{l1Sin}(\theta 1);\ \mathrm{y1} = \mathrm{l1Cos}(\theta 1)$$

$$\mathrm{x2} = \mathrm{x1} + \mathrm{l2Sin}(\theta 2);\ \mathrm{y2} = \mathrm{y1} + \mathrm{l2Cos}(\theta 2)$$

$$\mathcal{L} = 0.5\mathrm{m1}(\dot{x}1^2 + \dot{\mathrm{y}1}^2) + 0.5\mathrm{m1}(\dot{x}1^2 + \dot{\mathrm{y}1}^2) - \mathrm{m1}g\mathrm{y1} - \mathrm{m2}g\mathrm{y2}$$

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right) = \frac{\partial \mathcal{L}}{\partial q};\qquad q = \begin{pmatrix}\theta 1\\ \theta 2\end{pmatrix}$$

Cylindrical Double Pendulum Arm with Mass Concentrated at the Centre:



Equations of Motion:

Derived using Lagrange's equations of motion similar to above, gives us the following equation for the torque:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau$$

$$\begin{pmatrix} \alpha + 2\beta\mathrm{Cos}[\theta 2[t]] & \delta + \beta\mathrm{Cos}[\theta 2[t]] \\ \delta + \beta\mathrm{Cos}[\theta 2[t]] & \delta \end{pmatrix} \cdot \begin{pmatrix} \theta 1''[t] \\ \theta 2''[t] \end{pmatrix} +$$

$$\begin{pmatrix} \mathrm{b1} - \beta\mathrm{Sin}[\theta 2[t]]\theta 2'[t] & -\beta\mathrm{Sin}[\theta 2[t]](\theta 1'[t] + \theta 2'[t]) \\ \beta\mathrm{Sin}[\theta 2[t]]\theta 1'[t] & \mathrm{b2} \end{pmatrix} \cdot \begin{pmatrix} \theta 1'[t] \\ \theta 2'[t] \end{pmatrix} +$$

$$\begin{pmatrix} g\mathrm{m1r1Cos}[\theta 1[t]] + g\mathrm{m2}(\mathrm{l1Cos}[\theta 1[t]] + \mathrm{r2Cos}[\theta 1[t] + \theta 2[t]]) \\ g\mathrm{m2r2Cos}[\theta 1[t] + \theta 2[t]] \end{pmatrix} = \begin{pmatrix} \tau 1 \\ \tau 2 \end{pmatrix}$$

Where:

$$\alpha = \mathrm{Iz1} + \mathrm{Iz2} + \mathrm{m1r1}^2 + \mathrm{m2}(\mathrm{l1}^2 + \mathrm{l2}^2)$$

$$\beta = \mathrm{m2l2r2}$$

$$\delta = \mathrm{Iz2} + \mathrm{m2r2}^2$$

m1 = m2	0.5 kg	Mass
l1 = l2	0.5 m	Link length
r1 = r2	0.25 m	Distance to CoM
$I_{z1} = I_{z2}$	0.01 Kg m^2	Moment of Inertia about the axis of rotation
b1 = b2	0	Damping coefficients

The system has four states, the angular position and angular velocity of each of the two joints. The control inputs are the applied torques, one for each joint. In this way, the double pendulum is fully actuated, and should be able to follow any trajectory within the workspace.

State: $X = \{\{\theta1[t]\}, \{\theta2[t]\}, \{\theta1'[t]\}, \{\theta2'[t]\}\};$

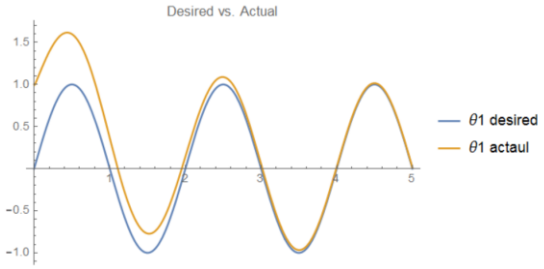
Control: $U = \{\{\mathrm{u1}[t]\}, \{\mathrm{u2}[t]\}\};$

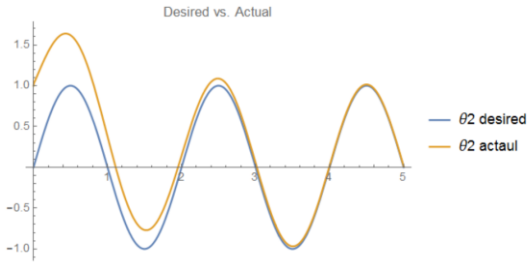
The importance in using this approach is that a somewhat complicated non-linear control problem is transformed into a simple design-problem of a linear system. My understanding is that the LQR scheme implemented calculates the feedforward control that minimizes the cost function while ensuring an optimal gain that guarantees that all states and control inputs of the *closed-loop* system goes to zero in finite time. My initial objective was to minimize the control energy while tracking a reference trajectory. As it stands, the control scheme does not minimize the torque, rather it tries to follow the trajectory regardless of required input torque. I made an attempt to set torque constraints using barrier functions [4] [5]. Essentially, I added the term$((U^6)^T.M.U^6)$ to the cost function, and its directional derivative to the derivative of the cost function, such that the input torque is capped because the cost function increases as the input torques increase. However, I ran into some issues during implementation, where my solution would not converge, even though it started out diverging. My first assumption was that either I am getting the wrong descent direction (variables not being updated correctly) in Armijo, or that the descent direction was correct, but I was taking steps that were too large and ended up increasing my cost again. However, upon investigation I found that my step sizes were being reduced. My next guess was that since I have essentially changed the cost function from what we have learned in the class, I may need to make appropriate changes to "a" and "b" as they depend on the partial derivatives of the cost function. It may also be the case that my barrier function is incorrect, and I need to define softer bounds.

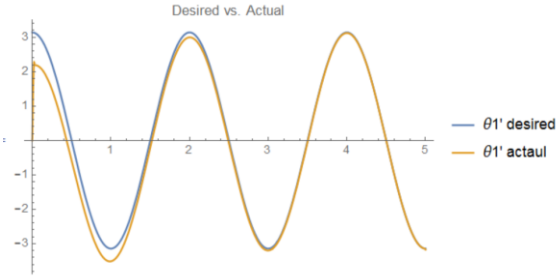
I used the solution set for Homework 8 as a basis for my code. The main changes that needed to be made were in the trajectories, the linearized equation of the system dynamics f[x,u], and the dimensionality was also altered. I use the Euclidean norm similar to the solution set. I also tried changing z(0) from equating 0 to be $-P^{-1}(0).r(0)$, but I could not get that working either. My guess is that is because of errors in my programming rather than anything else. I found that as the iterations increased more than 4, my code ran increasingly slower, where it would take more than ten minutes before the next iteration. This was because interpolating functions were being compounded for each iteration. I added a function using the Mathematica "Interpolation[]" function that samples the different values and generates one single interpolation function. This significantly improved the performance of the code. Additionally, using Clear[] at the end of my while loop to clear the variable from memory (of course with the exception of ζ, ξ and norm) also prevented some crashes due to memory running out. Furthermore, the "i" subscript takes a lot of memory, so I removed that from all variables, with the exception of ζ, ξ, and norm because it makes plotting them much easier.

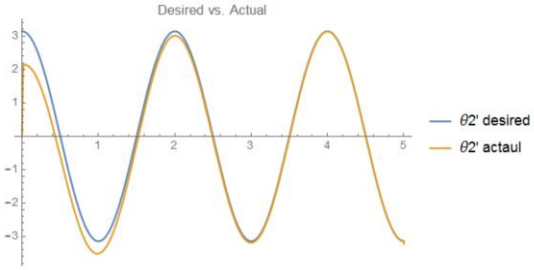
Even though I am using the solution set, my code is still very brittle. I have found that for this control scheme, the trajectories must be feasible, otherwise the controller with diverge infinitely.

Desired Trajectories: $\begin{pmatrix} \theta1[t] \\ \theta2[t] \\ \theta1'[t] \\ \theta2'[t] \end{pmatrix} = \begin{pmatrix} \mathrm{Sin}[\pi t] \\ \mathrm{Sin}[\pi t] \\ \pi\mathrm{Cos}[\pi t] \\ \pi\mathrm{Cos}[\pi t] \end{pmatrix}$

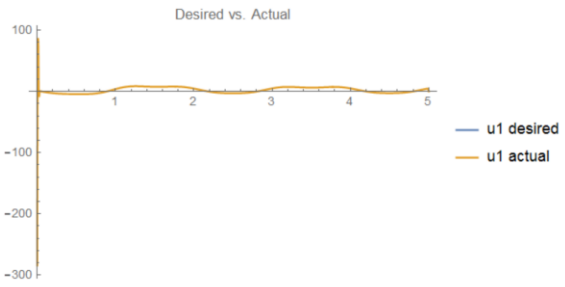


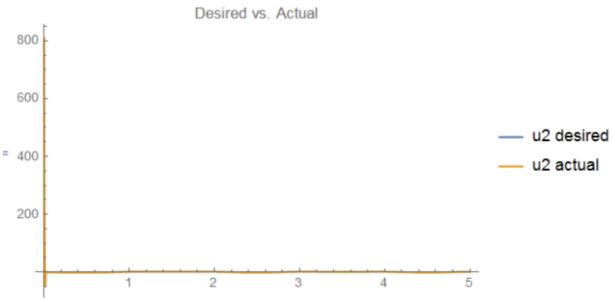


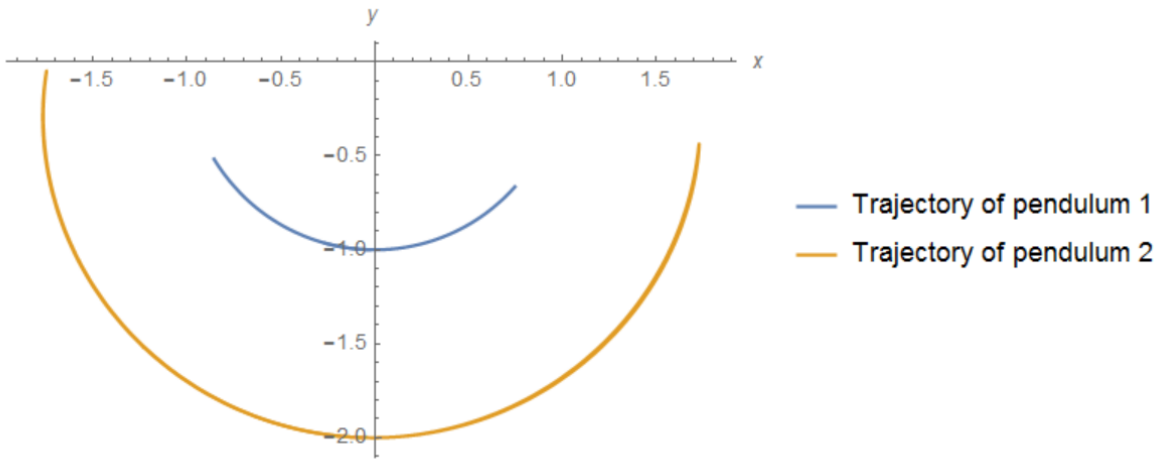




I found a trade-off between accurately tracing the desired the trajectory, and reducing the spike for the initial torque input. Currently, my Q and R matrices are constants. I believe if R was a function of time, such that the weight at t=0 was greater decreased as function of time, I would be able to get less spiking for the actual torque inputs u1 and u2, while also being able to trace the trajectory nicely as time goes on. However, tuning the constant matrices is challenging on its own, and I could not find the correct function of time the elements of R need to be. The way the Q, R, and P matrix are chosen have a great effect on whether a solution is found or not. In this way, optimal control feels like an art form.







Also, if the total simulation time is increased to 10 seconds, the optimization with begin to diverge. This may be because it starts to look too far ahead and gets too far away from the local minimum.

Future Work:

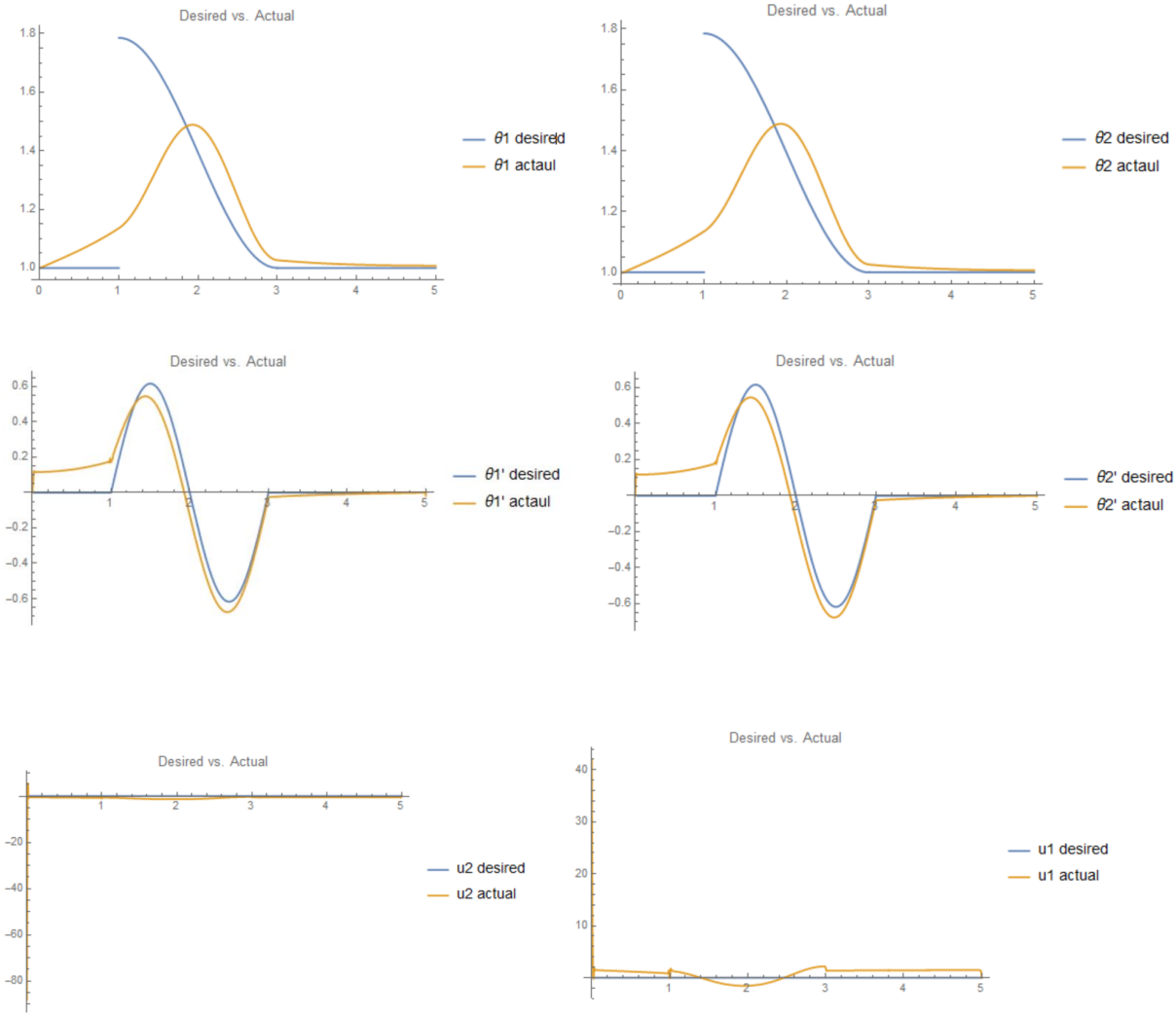
I would like to be able to fully implement torque constraints and angular velocity constraints in the system. This would make practical sense because joint motors impose bounds on the angular velocity. I think a good way to approach this would to make sure the penalty function makes sense, such that the values I get for zeta lie on the tangent space and really do satisfy $z' = Az+Bv$. Once this is ensured, I can manually choose a small step size, multiply it by zeta and add to xi, and find the projection. The cost of this projection should be less than before. After this investigation, I think I can get a better idea of how the system is operating and be able to formulate a solution.

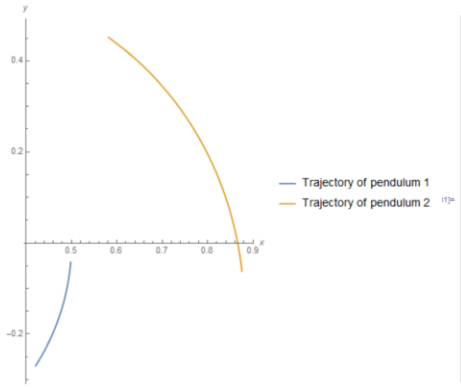
This was a very intellectually challenging class, and I really enjoyed it. Thanks for a great quarter!

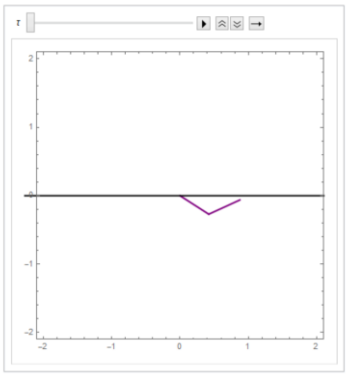
Another example trajectory:

$$Xd[t] = \begin{pmatrix} \begin{cases} 1 & 0 < t < 1 \\ 1 + \frac{1}{8}\pi\left(1 - \text{Cos}\left[\frac{1}{2}\pi\,(-3+t)\right]\right) & 1 \le t < 3 \\ 1 & 3 \le t < 5 \\ 0 & \text{True} \end{cases} \\ \begin{cases} 1 & 0 < t < 1 \\ 1 + \frac{1}{8}\pi\left(1 - \text{Cos}\left[\frac{1}{2}\pi\,(-3+t)\right]\right) & 1 \le t < 3 \\ 1 & 3 \le t < 5 \\ 0 & \text{True} \end{cases} \\ \begin{cases} 0 & 0 < t < 1 \\ \frac{1}{16}\pi^2\,\text{Sin}[\pi\,(-3+t)] & 1 \le t < 3 \\ 0 & \text{True} \end{cases} \\ \begin{cases} 0 & 0 < t < 1 \\ \frac{1}{16}\pi^2\,\text{Sin}[\pi\,(-3+t)] & 1 \le t < 3 \\ 0 & \text{True} \end{cases} \end{pmatrix}$$

Finds local minimum after 5 iterations.




y
0.4
0.2
−0.2
0.5
0.6
0.7
0.8
0.9
x
Trajectory of pendulum 1
Trajectory of pendulum 2


τ
2
1
0
−1
−2
−2
−1
0
1
2

References:

[1] "ArXiv.org Cs ArXiv:1311.1839v1." *[1311.1839v1] An Efficiently Solvable Quadratic Program for Stabilizing Dynamic Locomotion*. N.p., n.d. Web. 12 June 2015.

[2] Iida, Prof. Dr. Fumiya. "Optimizing Dynamic Motions of Two-link Pendulum." (n.d.): n. pag. Web.

[3] "Underactuated Robotics: Learning, Planning, and Control for Efficient and Agile Machines Course Notes for MIT 6.832." (n.d.): n. pag. Web

[4] Delchamps, David F. "The Discrete-Time Linear Quadratic Regulator Problem." *State Space and Input-Output Linear Systems* (1988): 393-405. Web.

[5] "Numerical Optimization." (2006): n. pag. Web.