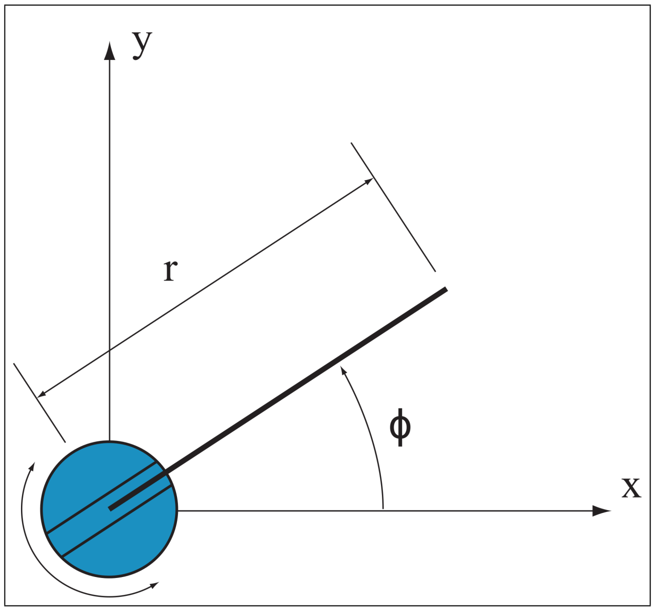
Project #5

ES_APPM 346-0

Code due Midnight, Mar. 17, 2021
Write-up due 5pm, Mar. 18, 2021

The final project for the quarter is a culmination of much of what we have studied over the course of the quarter. On the final day of class, we will have a timed competition to see who can solve this problem the quickest. The implementation details will be discussed in the assignment section.

Introduction



In this project, you will build a robotic path planner for a simple rotate plus extend robotic arm. The objective at all times is to determine the optimal path for moving the end of the arm from one position and speed to another. The arm is depicted in the Figure above.

We begin by defining the state of the robot. The state of the robot will be determined by four quantities: two for position and two for velocity. Let φ be the rotation of the robot and r be the length of the extension from the axis of rotation. Now define the four state variables to be

$$\begin{aligned} x_1 &= \varphi \\ x_2 &= r \\ x_3 &= \dot{\varphi} \\ x_4 &= \dot{r} \end{aligned}$$

where the dot notation is used to represent the time derivative.

The two controls for this robot are the torque u_1 for rotation, and the strength u_2 for extension. From this, we see that the change in angular momentum is given by the torque, and the change in radial momentum is given by the strength along with the centripedal force as shown in the following equations:

$$\frac{d}{dt}((I + mx_2^2)\dot{x}_1) = u_1 \tag{1}$$
$$\frac{d}{dt}(m\dot{x}_2) = u_2 + mx_2\dot{x}_1^2. \tag{2}$$

where m is the mass of the arm and I is the moment of inertia about the center of mass of the arm (this could also include whatever mass is being transported by this robot). Carrying out these derivatives and replacing $\ddot{x}_1$, $\ddot{x}_2$ with $\dot{x}_3$, $\dot{x}_4$ respectively, we get evolution equations for the position and velocity of the robot:

$$\dot{x}_1 = x_3 \tag{3}$$
$$\dot{x}_2 = x_4 \tag{4}$$
$$\dot{x}_3 = \frac{1}{I + mx_2^2}(u_1 - 2mx_2x_3x_4) \tag{5}$$
$$\dot{x}_4 = \frac{u_2}{m} + x_2x_3^2 \tag{6}$$

Note here that u_1, u_2 are functions of t and uniquely determine the path that the robot takes. The objective is to choose $u_1(t)$, $u_2(t)$ in such a way that the path is optimal according to some measure.

In our case, we choose to minimize the total expended energy during the transition from the initial to the terminal state. For this mechanical system, the expended energy is

$$E = \frac{1}{2}(u_1^2 + u_2^2)$$

The optimal path can be determined by using the Pontryagin maximum principle. Generically speaking, suppose we have a system of equations which model the motion of the robot given by

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}, \mathbf{u}, t), \quad 0 \le t \le T.$$

In this example, $\mathbf{f}$ represents the right hand side of the system (3)–(6). Also, suppose the chosen path has a performance score, in this case the total expended energy,

$$P = \int_0^T E(\mathbf{x}, \mathbf{u}, t)\, dt. \tag{7}$$

Then the optimal controller is found by using Lagrange multipliers in an equation of the form

$$H(\mathbf{x}, \mathbf{u}, t) = E(\mathbf{x}, \mathbf{u}, t) + \boldsymbol{\lambda}^T \mathbf{f}(\mathbf{x}, \mathbf{u}, t).$$

According to the Pontryagin maximum principle, the Lagrange multipliers for the optimal path satisfy the following pair of equations:

$$-\dot{\boldsymbol{\lambda}} = \frac{\partial H}{\partial \mathbf{x}}^T = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}^T \boldsymbol{\lambda} + \frac{\partial E}{\partial \mathbf{x}}^T \tag{8}$$
$$0 = \frac{\partial H}{\partial \mathbf{u}}^T = \frac{\partial E}{\partial \mathbf{u}} + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}^T \boldsymbol{\lambda} \tag{9}$$

Equation (8) results in an additional four evolution equations:

$$\dot{\lambda}_1 = 0 \tag{10}$$

$$\dot{\lambda}_2 = \frac{2m}{I + mx_2^2}\left(x_3x_4 + x_2\frac{1}{I + mx_2^2}(u_1 - 2mx_2x_3x_4)\right)\lambda_3 - x_3^2\lambda_4 \tag{11}$$

$$\dot{\lambda}_3 = -\lambda_1 + \frac{2m}{I + mx_2^2}x_2x_4\lambda_3 - 2x_2x_3\lambda_4 \tag{12}$$

$$\dot{\lambda}_4 = -\lambda_2 + \frac{2m}{I + mx_2^2}x_2x_3\lambda_3. \tag{13}$$

Equation (9) results in two more equations which express the control functions u_1, u_2 in terms of the x_i's and λ_i's:

$$u_1 = \frac{\lambda_3}{I + mx_2^2} \tag{14}$$

$$u_2 = \frac{\lambda_4}{m} \tag{15}$$

Substituting the expressions for the u_i's into the evolution equations for the x_i's and λ_i's, we can now formulate our problem as a two point boundary value problem. To find the optimal path, we must solve the eight differential equations (3)–(6), (10)–(13). The initial and terminal values for the arm, $x_1, \ldots, x_4$, are known, but none of the initial or terminal values are known for the λ_i's.

Of course, for a real robotic arm, this calculation would be done in real-time. Time is of the essence in computing these paths in order to make the robot maximally efficient. Your task will be to build a Matlab code that will take as input the requested initial and terminal conditions, and return the optimal control functions u_1, u_2.

Assignment

As discussed in class, this assignment has a competitive component to it. With this in mind, there are a few groundrules. First and foremost, **none of the canned solvers in Matlab may be used, e.g. `ode45`**, however, you **are** allowed to use the backslash for solving linear systems. The remainder of the rules concern the formatting and submission of your program which are detailed below. In addition to the competition, your written assignment answering the following questions must be submitted.

1. **CONTEST PROBLEM:** For the contest, you must submit a Matlab function which takes the following form:

```
function [t,x,u] = robotname(T,x0,xT)
% Written entirely and originally by Your Name
```

   You may choose any name for the `robotname` (preferrably something unique and not more than 20 characters that does not reveal your own name because it will be seen on the screen when we run the contest). However, your name must appear in the comments of the file, as the first line after the function line as shown above. This way, if I type "help robotname" at the command prompt, it will reveal the contest winner. Note: as stated in class, **you may not name your robot "terminator" or "terminator2", those names are reserved.**

   For input, `T` is the terminal time (the initial time will always be assumed to be zero), `x0` is a row vector specified as `[x1,x2,x3,x4]` which is the initial state of the robot, and `xT` is the row vector of requested terminal state of the robot. For output, `t` is a vector of arbitrary length N which is the time step locations, `x` is an $N \times 8$ matrix where `x(i,:)` is the state of the robot at time `t(i)`, and `u` is a $N \times 2$ matrix where `u(i,:)` is the amount of the force being applied (u_1, u_2) at time `t(i)`. Note that `u` can be computed via Equations (14), (15).

Contest Rules:

(a) There are two methods available for you to use for this final project, which we will cover in class. You may use either the shooting method or the finite difference method, it is entirely your choice. No matter which method you choose, you must still use the same input and output parameters as indicated above. You may assume that $m = 5$ and $I = 1.5$ in the model equations.

(b) **Your code must be your own creation**, if I find any suspicious similarities between your code and a competing code, both codes will be disqualified prior to the competition, not to mention it will be reflected in your grade for the project.

(c) Codes using any of the packaged ODE solvers available in Matlab will be disqualified (and will receive a poor grade, for that matter).

(d) Your main iteration loop **must** contain a safety valve so that it will not get into an infinite loop for any given initial conditions. In particular, if your Jacobian is nearly singular, your code must abort. This can be tested by checking `rcond(J) < 1e-15`. If this condition is met, your code must stop and return the most recent solution.

(e) The contest will be run as a March Madness style bracket. I am going to have to send a doodle poll to find a time to hold the contest, because our scheduled exam time is too early on Monday March 15.

(f) The initial pairings will be drawn at random from all the robots submitted by the deadline of midnight, March 17. You may submit your code up until the write-up deadline of 5pm March 18 to receive full credit for the project, but robots submitted after the first deadline will be ineligible for the competition. No revisions, resubmissions, or other modifications after the deadline will be accepted for the competition, but will be accepted for grading purposes.

(g) The winner of a bracket will be determined by the best scored solution for a randomly chosen set of time interval, initial conditions, and terminal conditions specified as `T`, x0 and `xT` above. Both robots in the bracket will be given the same conditions, but a new set of random conditions will be chosen for each bracket.

(h) The score of a run will be the combined sum of the elapsed time in seconds for the solution times 1000, plus the total expended energy as computed in (7), plus 1000 times the error in the terminal state computed as a 2-norm of the 4 component vector.

(i) To assist you with making sure your robot is compliant with the rules, there is a code validator called `validate.m` that is uploaded to Canvas. Run the code at the command prompt by typing:

```
[t,x,u] = robotname(T,x0,xT)
[ok,reason] = validate(t,x,u,x0,xT)
```

The function will return `ok==1` if it was acceptable. If not, then `reason` will give the reason why it failed. Also, you can test your robot under competition conditions by using the battle test program uploaded to canvas called `battletest.m`. Run the code at the command prompt by typing:

```
battletest('robotname')
```

(j) This is a single elimination tournament.

2. Instructions for the write-up:

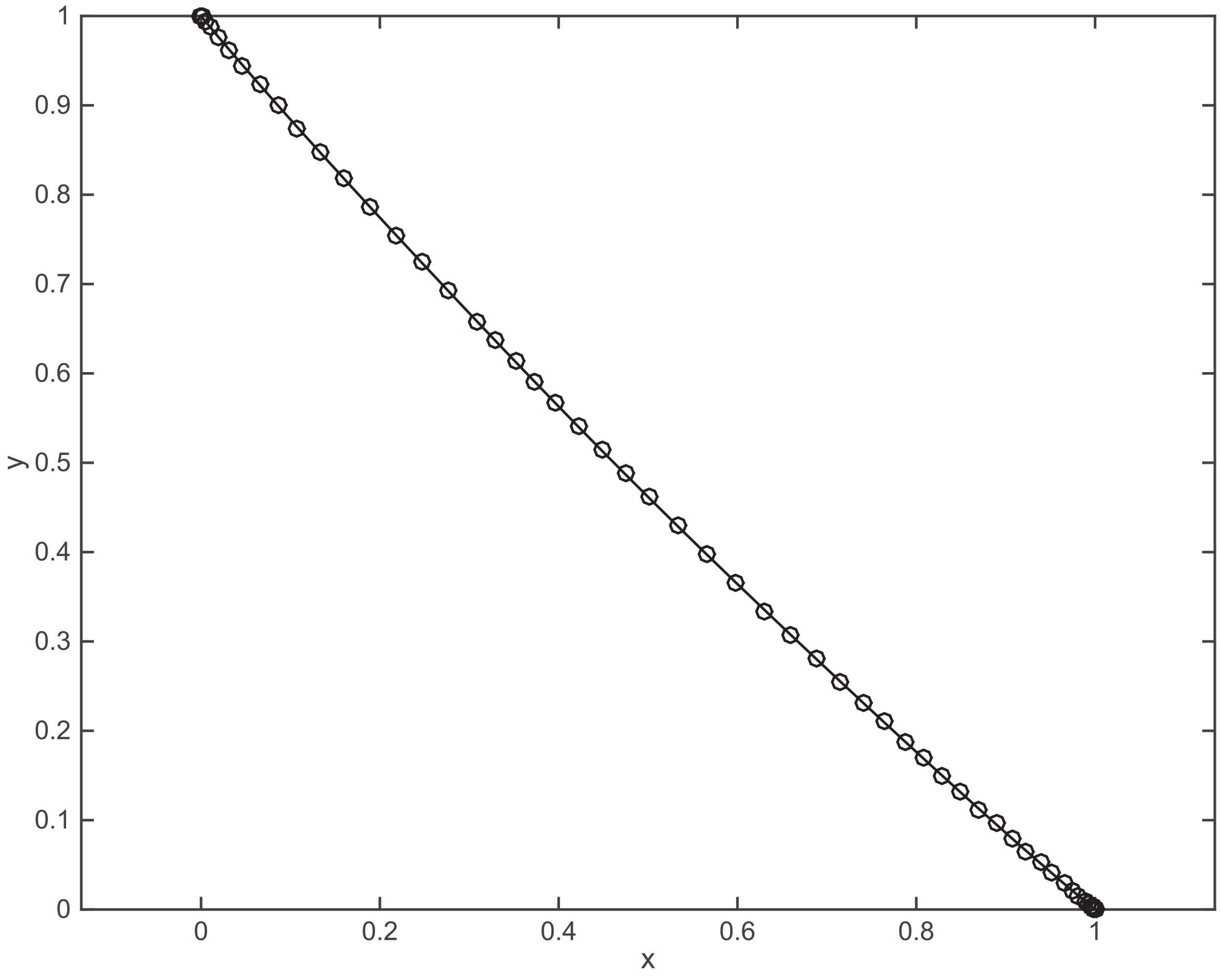
(a) I am not sure whether I will have time to make a Matlab Grader test for this problem. If I do I will let you know via Canvas Announcements. In that case, make sure the format of the function or functions satisfy the instructions on Grader (the function name will be given not, chosen by you). It should require only minor modifications to submit.

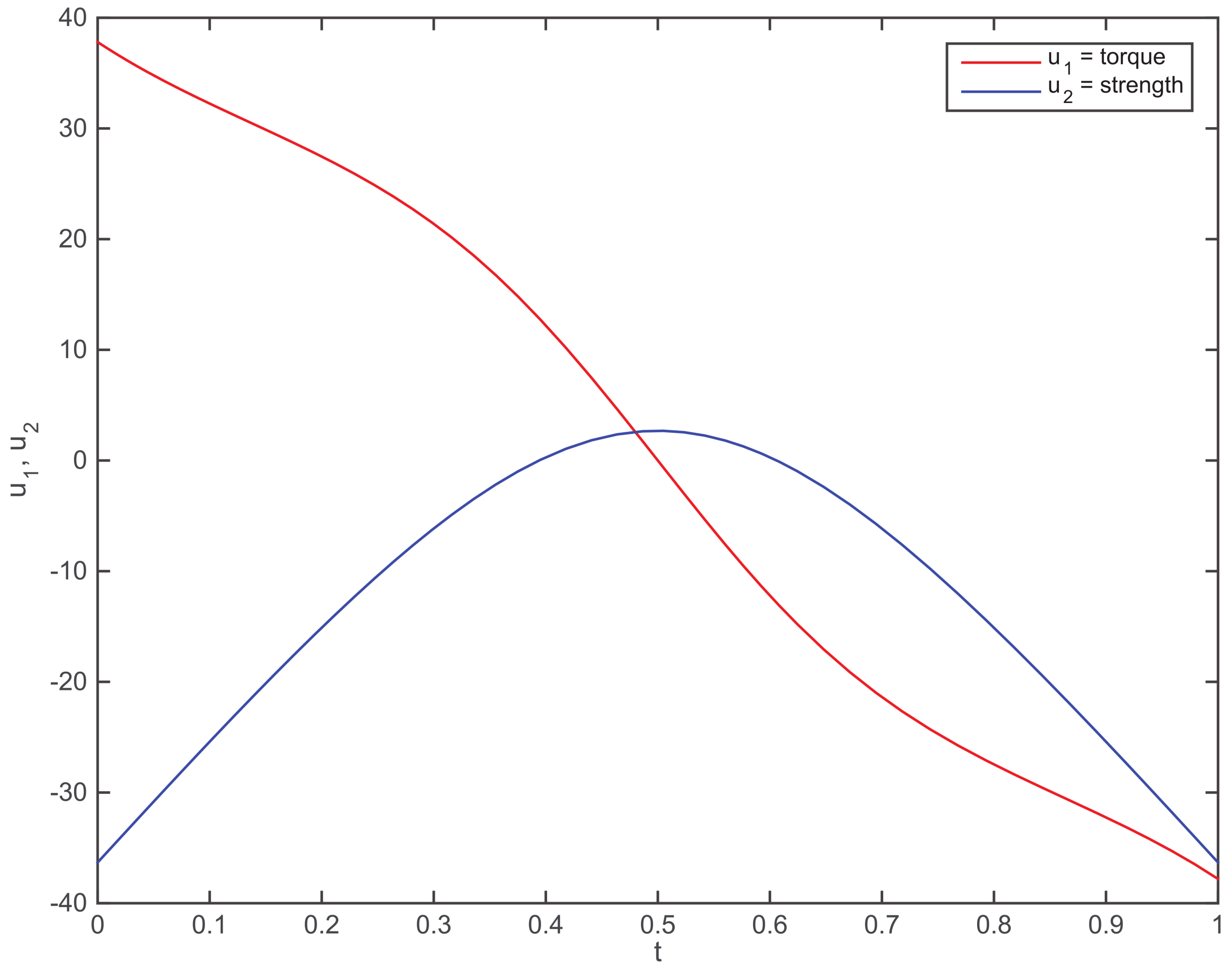
(b) State which method(s) you chose to implement and give reasons for making the choice, e.g. what are the advantages and disadvantages of the choice you made?

(c) Test your code for the following input:

```
[t,x,u] = robotname(3,[0,1,0,0],[pi/2,1,0,0]);
subplot(1,2,1)
plot(x(:,2).*cos(x(:,1)),x(:,2).*sin(x(:,1)),'k-o');
axis equal
subplot(1,2,2)
plot(t,u(:,1),'r-',t,u(:,2),'b-');
```

To help you test, below is the result for my code (without using the subplot commands):





Print the corresponding plots for your results for this problem. Note that location of the time points will almost certainly be different between my results and yours, but the shape of the plots should be similar.

(d) Suppose the starting point of the robot is $(1, 0)$ corresponding to `x0 = [0,1,0,0]` for your robot. The terminal points

$$\theta_T(\phi) = \tan^{-1}\left(\frac{2\sin\phi}{1+2\cos\phi}\right)$$
$$r_T(\phi) = \sqrt{5+4\cos\phi}$$

define a circle of radius 2 about the starting point. Plot this circle and indicate for which points along the circle your algorithm is able to successfully converge. You can check convergence by running your robot with $T = 10$, and $x_T = (\theta_T(\phi), r_T(\phi), 0, 0)$ for different values of ϕ. The method can be said to converge if `x(end,1:4)-xT` is within a tolerance. Do you notice any kind of pattern?

Good luck everyone!!!