DEPARTMENT OF MECHANICAL ENGINEERING
RICE UNIVERSITY

MECH 498/598: INTRODUCTION TO ROBOTICS

Lab #3
Dynamic Simulation

Due Friday, 20 April 2018

Assignment Description
For this assignment, you will once again be able to work in groups of two. Only one submission (electronic only) per group is required. In this lab you will build a simple dynamic simulation of a 2DOF manipulator and then implement control of a 3DOF manipulator moving an object around in space. Some code has been provided for you on Canvas as well, with additional instructions in the comments.

Submission Instructions
Your submission for this assignment will only have an electronic component. For the electronic submission, just as for previous labs, you should create a folder named **`[your_net_id]_AND_[teammate_net_id]_lab3`**, where your Rice ID is your shortened email address, e.g. "cgm4." Then, upload a compressed form of this folder named **`[your_net_id]_AND_[teammate_net_id]_lab3.zip`** to the Canvas assignment, as you did before. This submission folder should contain all the files provided to you that you did not modify, the following files you with your modifications with these exact names, and any additional files needed to run your simulation:

- **`RRInit.m`**
- **`simulateRR.m`**
- **`robController.m`**
- **`createRobTrajectory.m`**

1. In this lab, you will create software for the dynamic simulation of chain link manipulators. Begin by considering the 2-link, RR, nonplanar, manipulator shown below. **Important:** In addition to the two point masses m_1 and m_2 shown at the ends of the two links, the two links should be considered to be rigid, thin rods with masses m_{r1} and m_{r2} , respectively, which are evenly distributed throughout their volume. The dynamic equations are as follows:

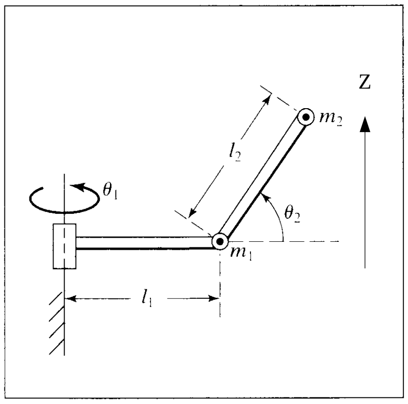
$$M_1 L_{c1}^2 \ddot{\theta}_1 + M_2 (L_1 + L_{c2}\cos\theta_2)^2 \ddot{\theta}_1 + (I_1 + I_2)\ddot{\theta}_1 - 2M_2 L_{c2}\sin\theta_2\,(L_1 + L_{c2}\cos\theta_2)\dot{\theta}_1\dot{\theta}_2 = \tau_1$$

$$M_2 L_{c2}^2 \ddot{\theta}_2 + I_2\ddot{\theta}_2 + M_2 L_{c2}\sin\theta_2\,(L_1 + L_{c2}\cos\theta_2)\dot{\theta}_1^2 + M_2 g L_{c2}\cos\theta_2 = \tau_2$$

$$M_1 = m_1 + m_{r1} \qquad M_2 = m_2 + m_{r2}$$

$$L_{c1} = \frac{\left(m_1 + \frac{1}{2}m_{r1}\right)L_1}{M_1} \qquad L_{c2} = \frac{\left(m_2 + \frac{1}{2}m_{r2}\right)L_2}{M_2}$$

$$I_1 = \frac{1}{12}m_{r1}L_1^2 + m_1\left(\frac{L_1}{2}\right)^2 \qquad I_2 = \frac{1}{12}m_{r2}L_2^2 + m_2\left(\frac{L_2}{2}\right)^2$$



A simple approach to building a dynamic simulation consists of the following steps:

- Specify and store the initial state of the system. The state includes θ and $\dot{\theta}$ for each joint.
- Next, repeat the following three steps. During repetition, time is incremented from 0 until some final time t_f in small increments dt.
  - Sum the torques on each link, τ, at the current instant in time. If there are no torques (that is, there is nothing applied externally and no control feedback), $\tau = 0$.
  - To compute the acceleration of each joint, $\ddot{\theta}$, start with the dynamic model of the system given in class: $\tau = M(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}} + C(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) + G(\boldsymbol{\theta})$. Next, solve for the vector $\ddot{\boldsymbol{\theta}}$ using the inverse of the inertia matrix $M(\boldsymbol{\theta})$.
  - Calculate the new state of the system by numerical integration of the acceleration for each joint. A simple numerical integration technique is the trapezoidal rule: at time step i, the velocity is $v_i = v_{i-1} + 0.5(a_{i-1} + a_i)dt$, and the position is $p_i = p_{i-1} + 0.5(v_{i-1} + v_i)dt$. (This formula implies that the updated derivative a_i or v_i has been calculated using the states from the previous iteration i-1.)

  The sample MATLAB script simulateRR.m provided on Owl-Space does all of these steps, with empty sections where you need to fill in the blanks. [30 points]

  a. Modify the sample scripts given to create a dynamic simulation of the manipulator above. The simulation should last 10 seconds, with increments of 0.01 seconds. Play with the values of the physical parameters in the system, as well as the initial conditions. Watch the effects in the video output of your simulation.

  b. A good way to check that your simulation is working properly is to plot the total energy of the system over time. Find the kinetic and potential energy of the RR robot and generate the plot within the loop of your simulation code.

2. Now you will add a simple controller to your simulation. Use the following constants: $m_1 = 1, m_2 = 5, m_{r1} = 2.3, m_{r2} = 2.3, l_1 = 1, l_2 = 1.41,$ with the manipulator initially at rest in the position $\theta_1 = \pi/3$ and $\theta_2 = \pi/2$. Use a proportional-derivative feedback controller for each joint, so that $\tau = -K_p(\theta - \theta_d) - K_v\dot{\theta}$., where $\theta_{1d} = 0$ and $\theta_{2d} = \pi/2$, and the feedback gains are the same for both joints. To more closely represent physical systems, your joint torques should be limited to $-20 < \tau < 20$. Torques outside this limit should be limited to the max or min value. Select values of K_p and K_v that give an underdamped response. Demonstrate the effect of this controller on the system by providing plots of θ_1 and θ_2 versus time. [20 points]

3. Now it's time to meet Rob the 3-link robot. Rob looks an awful lot like the first three links of the FANUC robot you encountered in Lab 2. The MATLAB code to simulate interaction with Rob will be provided to you in the

files **`robInit()`**, **`dhtf()`**, **`robFK()`**, **`drawRob()`**, **`setRob()`**, **`robIK()`**, **`robJacobian()`**, and **`simulateRob()`**. Rob has a single task to accomplish – he needs to pick up a heavy magnetic ball, move it to a new location, and then return to his home configuration. Fortunately, Rob's end effector is equipped with a powerful electromagnetic that can be flipped on and off, so picking and placing the ball is as simple as having the end effector within a certain distance of the target position while the end effector velocity is below some threshold. You need to design the robot controller to move Rob smoothly and stably from one end effector position to another. Do this with the function **`[trajectory] = createRobTrajectory(via, rob)`** You will be given the Cartesian coordinates of the ball's starting and ending location in the variable **`via`** ; your first task is to generate a trajectory of joint positions and velocities that moves the robot slowly enough through the following positions: home position, ball start, ball end, home position. To move the robot slowly you will need to make a discretized path in Cartesian space as you did for Lab 2 where the points are close together. Then, make use of the inverse kinematics function to calculate joint space trajectories. You should use a PD controller with gravity compensation, and implement your controller by editing the function **`[tau] = robController(trajectory, Theta, Theta_dot, t, rob)`**. The output tau is a vector of the joint torques to be applied by the motors. Try completing the task without gravity compensation first, and you will notice why it is necessary. [50 points]

Notes:

- You do not need to hand in printouts of your MATLAB code, only the plots requested of you.
- Test your files carefully, and add comments. Also, place your name in a comment in each file.
