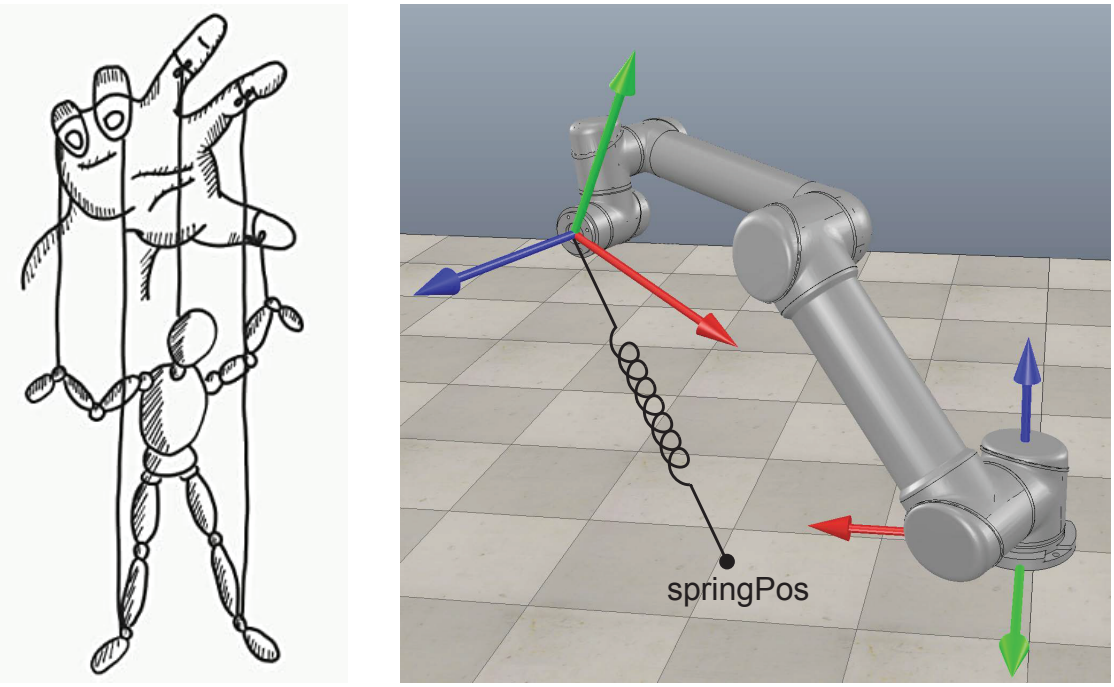
ME 449 Assignment 3
Puppet on a String
Due 1:30 PM, Wednesday November 10, 2021

You've probably seen animation and physics engine tools that allow you to interactively grab a link of a virtual mechanism and apply forces to it, to make it move like a marionette on a string. In this assignment, you will create the underlying software that allows you to do this for a serial-chain robot.



Left: A puppet on a string. Right: A robot on a spring.

Create the new function `Puppet`. It should take as input the following:

- thetalist: an n-vector of initial joint angles (units: rad)
- dthetalist: an n-vector of initial joint rates (units: rad/s)
- g: the gravity 3-vector (units: m/s^2)
- Mlist: the configurations of the link frames relative to each other at the home configuration. (There are eight frames total: {0} or {s} at the base of the robot, {1} ...{6} at the centers of mass of the links, and {7} or {b} at the end-effector.)
- Slist: the screw axes $\mathcal{S}_i$ in the space frame when the robot is at its home configuration
- Glist: the spatial inertia matrices $\mathcal{G}_i$ of the links (units: kg and kg m^2)
- t: the total simulation time (units: s)
- dt: the simulation timestep (units: s)
- damping: a scalar indicating the viscous damping at each joint (units: Nms/rad)
- stiffness: a scalar indicating the stiffness of the springy string (units: N/m)
- springPos: a 3-vector indicating the location of the end of the spring not attached to the robot, expressed in the {s} frame (units: m)
- restLength: a scalar indicating the length of the spring when it is at rest (units: m)

As output, it produces

- thetamat: an $N \times n$ matrix where row i is the set of joint values after simulation step $i-1$
- dthetmat: an $N \times n$ matrix where row i is the set of joint rates after simulation step $i-1$

To turn `Puppet` into an interactive animation tool, you would allow the user to interactively change springPos, but in this assignment we will keep the end of the spring fixed in space. The spring connects the location springPos (represented in the {s} frame) to the origin of the {b} frame. If the distance between the springPos and {b} is more than restLength, the spring pulls the end-effector toward springPos with a magnitude (stiffness $\times$ (distance $-$ restLength)). If the distance is less than restLength, the spring pushes the end-effector away from springPos.

If your total simulation time t is 5 s and dt is 0.01 s, you will have $N = 500$ or 501 simulation steps that you will animate. Your code will iteratively call the MR function `ForwardDynamics`, and you can use simple first-order Euler numerical integration, as implemented by `EulerStep`. (You are welcome to experiment with other integrators, such as using the one-half acceleration times dt^2 term to get a more accurate change in position each timestep, but `EulerStep` is fine for this project.) The MR function `ForwardDynamicsTrajectory` also simulates the motion of a robot, so you can look at that if you wish, but it takes different inputs and handles the simulation timesteps differently.

For the robot, we will use the UR5, a popular 6-dof industrial robot arm. The robot has geared motors at each joint, but in this project, we ignore the effects of gearing, such as friction and the increased apparent inertia of the rotor.

The relevant kinematic and inertial parameters of the UR5 are:

$$M_{01} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.089159 \\ 0 & 0 & 0 & 1 \end{bmatrix}, M_{12} = \begin{bmatrix} 0 & 0 & 1 & 0.28 \\ 0 & 1 & 0 & 0.13585 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, M_{23} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -0.1197 \\ 0 & 0 & 1 & 0.395 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$M_{34} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0.14225 \\ 0 & 0 & 0 & 1 \end{bmatrix}, M_{45} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.093 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, M_{56} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.09465 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$M_{67} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.0823 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, G_1 = \texttt{diag}([0.010267495893, 0.010267495893, 0.00666, 3.7, 3.7, 3.7]),$$

$$G_2 = \texttt{diag}([0.22689067591, 0.22689067591, 0.0151074, 8.393, 8.393, 8.393]),$$
$$G_3 = \texttt{diag}([0.049443313556, 0.049443313556, 0.004095, 2.275, 2.275, 2.275]),$$
$$G_4 = \texttt{diag}([0.111172755531, 0.111172755531, 0.21942, 1.219, 1.219, 1.219]),$$
$$G_5 = \texttt{diag}([0.111172755531, 0.111172755531, 0.21942, 1.219, 1.219, 1.219]),$$
$$G_6 = \texttt{diag}([0.0171364731454, 0.0171364731454, 0.033822, 0.1879, 0.1879, 0.1879]),$$

$$\texttt{Slist} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & -0.089159 & -0.089159 & -0.089159 & -0.10915 & 0.005491 \\ 0 & 0 & 0 & 0 & 0.81725 & 0 \\ 0 & 0 & 0.425 & 0.81725 & 0 & 0.81725 \end{bmatrix}.$$

For your convenience, these parameters are given in Python, Mathematica, and MATLAB at http://hades.mech.northwestern.edu/index.php/Modern_Robotics#Supplemental_Information.

Your code will be tested in stages to help ensure correctness. **Important:** You are welcome to talk to classmates about concepts at the development phase, but you are not allowed to share your

code nor look at anyone else's code. AI-based software easily detects shared and altered code with common origins, so do not do it.

Part 1: Simulating a falling robot. In the first part, the robot will fall in gravity without damping or the external spring (damping and stiffness are both set to zero). Since there is no damping or friction, the total energy of the robot (kinetic plus potential) should remain constant during motion. Gravity is $g = 9.81$ m/s^2 in the $-\hat{z}_s$-direction, i.e., gravity acts downward.

Simulate the robot falling from rest at the home configuration for five seconds. The output data should be saved as a .csv file, where each of the N rows has six numbers separated by commas. This .csv file is suitable for animation with the CoppeliaSim UR5 csv animation scene. Adjust the animation scene playback speed ("Time Multiplier") so it takes roughly five seconds of wall clock time to play your csv file. You can evaluate if your simulation is preserving total energy by visually checking if the robot appears to swing to the same height (same potential energy) each swing. Choose values of dt where the energy appears nearly constant (without choosing dt unnecessarily small) and where the energy does not appear constant (because your timestep is too coarse). Capture a video for each case and note the dt chosen for each case.

Part 2: Adding damping. Now experiment with different damping coefficients as the robot falls from the home configuration. Damping causes a torque at each joint equal to the negative of the joint rate times the damping. Create two videos showing that when you choose damping to be positive, the robot loses energy as it swings, and when you choose damping to be negative, the robot gains energy as it swings. Use t = 5 s and dt = 0.01 s, and for the case of positive damping, the damping coefficient should almost (but not quite) bring the robot to rest by the end of the video. Do you see any strange behavior in the simulation if you choose the damping constant to be a large positive value? Can you explain it? How would this behavior be affected if you chose shorter simulation timesteps?

Part 3: Adding a spring. Make gravity zero and choose a springPos at $(0, 0, 1)$ in the {s} frame with a restLength of zero. All initial robot joint angles are zero except for joint 2, which is -1 rad (the arm is pointing up). Experiment with different stiffness values, and simulate the robot for t = 10 s and dt = 0.01 s starting from the home configuration. Capture a video for a choice of stiffness that makes the robot oscillate a couple of times and record the stiffness value. Does the motion make sense to you? Should total energy be conserved? Does the total energy appear to be conserved? Do you see any strange behavior if you choose the spring constant to be large?

Now add a small positive damping to the simulation that makes the arm nearly come to rest by the end of the video. For both videos, record the stiffness and damping you used.

What to turn in: You should turn in a single zip file named FamilyName_GivenName_asst3.zip. It should contain a pdf file FamilyName_GivenName_asst3.pdf; videos named part1a.mp4, part1b.mp4, part2a.mp4, part2b.mp4, part3a.mp4, and part3b.mp4; corresponding csv files named part1a.csv, etc., that were animated by CoppeliaSim to create your videos; and a directory called "code." The pdf file should have four sections: Introduction, Part 1, Part 2, and Part 3. In the Introduction, provide any explanation needed to understand your overall submission. (This may be very short.) In Parts 1-3, answer all questions posed above and provide information explaining the corresponding videos and csv files, including the dt, t, damping, and stiffness values used for the animations. In the code directory, provide your commented function(s) and sample code that we can execute to generate the csv files you turned in.