Introduction to Robotics (CS223A) Homework #6

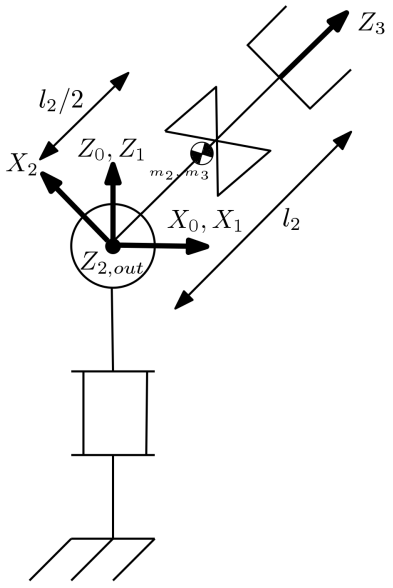
(Winter 2018/2019) Due: **Wednesday, March 13**

Name _____ **SUNet ID** _____

Some tips for doing CS223A problem sets:

- Use abbreviations for trigonometric functions (e.g. $c\theta$ for $\cos(\theta)$, s_1 or $s\theta_1$ for $\sin(\theta_1)$) in situations where it would be tedious to repeatedly write sin,cos, etc.
- Use common sense for decimals – – if the question states a = 1.34, then don't give answers like 2*a = 2.680001245735.
- If you give a vector as an answer, make sure that you specify what frame it is given in (if it is not clear from context). The same rule applies to rotation and transformation matrices.
- Circle your final answers.

1. **Inverse Dynamics and Gravity Compensation**
   Consider the RRP manipulator shown below, where l_2 is 0.3m and m_2 and m_3 are located at 0.15m along that link:



   The joint configuration and mass information is given to you in `main.py`.

(a) The system is controlled by a joint-space dynamics decoupling control, $\tau = \alpha\tau' + \beta$, which compensates the non-linear part of the system, decouples the dynamics, and tracks a desired trajectory separately for each joint. The control law is given as below:

$$\tau = M(q)\ddot{q} + V(q, \dot{q}) + G(q) \tag{1a}$$

$$\ddot{q} = \ddot{q}_d - k'_p(q - q_d) - k'_v(\dot{q} - \dot{q}_d) \tag{1b}$$

Find the values for the gains k'_p, k'_v such that the closed-loop system for all joints are critically damped with natural frequency of 7 rad/sec.

(b) Determine the configuration of the robot where the magnitude of gravity felt by joint 1 is at a maximum. Similarly, determine the configuration for joint 2 and joint 3. How do you explain this intuitively?

(c) Implement `inverse_dynamics()` in `controllers.py` with the control law given in Eq. (1). For stability reasons, we ignore centrifugal and Coriolis forces (if our estimate of $V(q, \dot{q})$ is inaccurate, this term might inject energy into the system, making it unstable in the Lyapunov sense). In practice, dropping $V(q, \dot{q})$ is okay because centrifugal and Coriolis forces are small when the robot is moving slowly. Thus, we pretend our dynamics model looks like $\tau = M(q)\ddot{q} + G(q)$. This question will be tested by the autograder.

(d) Suppose the desired joint trajectory is given as a function of time $q(t) = [sin(t), t, 0.25t]$. Use the inverse dynamics controller you implemented to track this trajectory, with the gains you computed in part (a). In `main.py`, the desired trajectory and robot configuration can be accessed with the functions `q1_trajectory()` and `q1_q2_robot()`. Attach a plot of the joint values q over time t during the simulation. Comment on the system response to your controller.

For this homework, main.py is set up to run the simulation for you in each part and then generate plots, but you need to make a few small changes. First, you must fill in the correct controller gains in the gains dictionary at the beginning of the main function. When you want to run a simulation, make sure the appropriate trajectory and controller are uncommented in the code sections near line 165. Finally, the function by default creates a plot of only the actual joint positions over time. You may need to change this plotting code in the section that begins at line 220 for each problem so that it shows the quantities that we ask for. The tutorials for matplotlib found at `https://matplotlib.org/users/index.html` *will give you a basic idea of the syntax if you are unfamiliar with the package*

Hint: You can see the robot moving in action by running `visualizer.py` in a separate terminal window, going to `localhost:8000` in a web browser, and clicking the "Simulation" link. The robot will move when you run `main.py`.

(e) If we don't compensate for the gravity term, what would happen? Attach a plot of the joint values during simulation. Comment on the system response. (Remember to add gravity compensation back in your function after this question.)

2. **Operational Space Control of a Non-redundant Manipulator**
This type of control projects joint space dynamics $M(q), V(q, \dot{q}), G(q)$ into task space dynamics $M_x(q), V_x(q, \dot{q}), G_x(q)$ at the end-effector. It is based on our favorite equation $\tau = J^T(q)F$ where:

$$\begin{aligned}
\tau &= J^T(q)F \\
F &= M_x(q)\ddot{x} + V_x(q, \dot{q}) + G_x(q) \\
\ddot{x} &= \ddot{x}_d - k_p(x - x_d) - k_v(\dot{x} - \dot{x}_d) \\
M_x(q) &= J^{-T}MJ^{-1} \\
V_x(q, \dot{q}) &= J^{-T}V - M_x\dot{J}\dot{q} \\
G_x(q) &= J^{-T}G
\end{aligned}$$

After some algebraic manipulation, this can be expressed as:

$$\begin{aligned}
\tau &= M(q)J^{-1}\ddot{x} + V(q, \dot{q}) - MJ^{-1}\dot{J}\dot{q} + G(q) \\
\ddot{x} &= \ddot{x}_d - k_p(x - x_d) - k_v(\dot{x} - \dot{x}_d)
\end{aligned}$$

Ignoring centrifugal and Coriolis forces, this becomes equivalent to a PD joint space controller with J^{-1} as the projection from task space accelerations $\ddot{x}$ to joint space accelerations $\ddot{q}$ (the true relationship is $\ddot{x} = J\ddot{q} + \dot{J}\dot{q}$):

$$\begin{aligned}
\tau &= M(q)\ddot{q} + G(q) \\
\ddot{q} &= J^{-1}\ddot{x} \\
\ddot{x} &= \ddot{x}_d - k_p(x - x_d) - k_v(\dot{x} - \dot{x}_d)
\end{aligned}$$

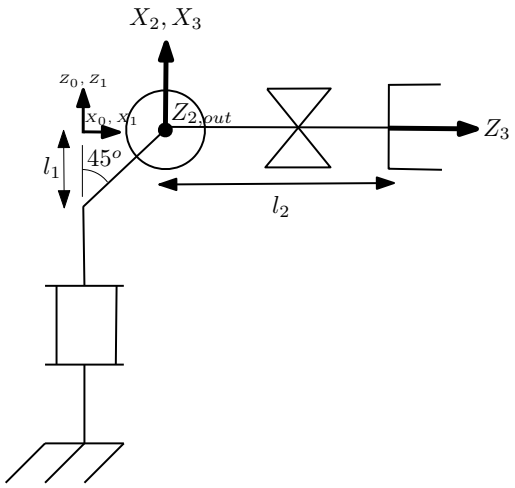
For the manipulator in Problem 1, we are going to assume that we have some errors in our model. Our inaccurate mass matrix is $\hat{M} = M + \delta I$

(a) Implement `nonredundant_operational_space()` in `controllers.py` that will allow the end effector to track a desired trajectory. The output of this function should be torques that we send to the robot joints. This question will be tested by the autograder.

(b) If you use the same k_p and k_v values that you found in problem 1 in task space, what happens for different values of δ? A simple end effector trajectory is given to you in `main.py`. You could use this trajectory to test your controller. You could also come up with other trajectories. Include plots of the end-effector position over time.

(c) How sensitive is this type of control law to errors in our robot model? In other words, what is the largest δ that the controller can tolerate while maintaining a reasonable performance?

3. **Inverse Kinematics as an Optimization Problem**
Consider the RRP manipulator shown below. Assume $l_1 = 0.2$ and $l_2 = 0.1$.



Suppose we want to use Inverse Kinematics to control this robot. For the simple 3 DOF robot above, velocity-based Inverse Kinematics is equivalent to taking the inverse of the Jacobian to get $\dot{q}_{des} = J^{-1} v_{\text{des}}$. However, if we want to enforce joint limit constraints, then we need to set up Inverse Kinematics as a quadratic program:

$$\begin{aligned} \min_{\dot{q}_{\text{des}}} \; & ||J\dot{q}_{\text{des}} - v_{\text{des}}||^2 \\ \text{s.t.} \quad & \dot{q}_{\text{des}} \geq K_{lim} \frac{\underline{q} - q}{\delta t} \\ & \dot{q}_{\text{des}} \leq K_{lim} \frac{\bar{q} - q}{\delta t}, \end{aligned}$$

where $\underline{q}, \bar{q}$ are the lower and upper joint limits. To avoid Jacobian conditioning issues near singular configurations, we usually also penalize large joint velocities with a regularization term, such that the objective function becomes:

$$\min_{\dot{q}_{\text{des}}} \; ||J\dot{q}_{\text{des}} - v_{\text{des}}||^2 + \alpha ||\dot{q}_{\text{des}}||^2,$$

where α is the weight to penalize large velocities. The solution to this optimization gives us a $\dot{q}_{des}$ that we could command to a velocity-controlled robot. However, for a torque-controlled robot, we need to convert the joint velocity to a torque. One way to do this is with inverse dynamics control (Problem 1) with $\dot{q}_{des}$ and the following q_{des}:

$$q_{des} = q + \dot{q}_{des} * dt$$

The inverse dynamics controller then gives us the final torques we send to the robot.

(a) In `controllers.py`, use the provided `solve_qp()` function to implement an I.K. controller that will allow the end effector to track the sinusoidal trajectory specified in the provided starting code file `main.py`. (You may need to install `quadprog` by calling `pip install -r requirements.txt` in the virtual environment).

(b) Discuss what happens when you try to track the sinusoidal trajectory for different values of α. Include plots of the end-effector position over time.