Experimental Robotics (CS225A) Homework #2

(Spring 2016/2017) Due: **Tuesday, April 25**

In this homework assignment, you will implement operational space control for the Kuka and experiment with feedback linearization in operational space.

To download the assignment, you'll have to pull the latest updates from `cs225a-dist.git`. If you want to keep your progress from previous homeworks, first call `git status` to see what files you've modified, and then call `git add <filename>` for all the files you want to save. Next call `git commit -m "Your commit message here"` to commit the changes to these files. For the rest of the files you don't care about, call `git stash` to revert them back to the original version (if you ever decide you want to bring back the modified files, you can call `git stash pop`). At this point, `git status` should show no modified files (untracked files are fine).

Now, you are ready to download the assignment. Call `git pull --recurse-submodules`. This will likely ask you to save a commit message for merging `cs225a.git` with your local repository - you can simply save and exit. If there were merging issues, you'll have to go into the problem files, manually fix the merging, and then commit those changes again. If you get issues about linking errors when building, the sai2-common updates might not have pulled correctly. Try `cd sai2-common` from the cs225 folder to go into the sai2-common folder and call `git pull origin master`. Go back into the cs225a folder and call the make script again with `./make.sh`.

1. Let the task positions and velocities of the robot be given by x and $\dot{x}$, respectively. Let x_d be the desired position of the robot. Implement the operational space control law with joint space gravity:

$$F = \Lambda(k_p(x_d - x) - k_v\dot{x})$$
$$\Gamma = J_v^T F + g$$

where k_p and k_v are your control gains, Λ is your operational space mass matrix, and g is your joint space gravity vector. Notice that in your code you will see two controllers, a joint space controller and a spot for you to fill in your operational space controller. We need the joint space controller to get our robot out of singularity configuration from the initial configuration; after the desired joint coordinates are reached, we switch to the other controller for the rest of the simulation.

   (a) Tune your gains to achieve critical damping with $k_p = 300$ and report your chosen k_v. Simply make the operational space goal a point in Cartesian space for the robot to reach (anywhere within the workspace but reasonably far from the initial location). Plot the operational point trajectory as well as the joint trajectories.

   (b) Notice that the robot reaches the goal position but is still moving weirdly, why is this happening? *Hint: The task is 3DOF but the Kuka is a 7DOF robot. Does this affect the motion?

   (c) We will add extra damping to remove this issue. Add to your commanded torques some joint space damping. Choose a good kv_joint value. Plot the operational point trajectory as well as the joint trajectories now. Compare with before.

(d) Notice your robot moves a lot slower now because all your joints are more damped (a physical intuition is all your joints have more friction now). We will implement the null space damping to only add damping in the null space of our task.

$$\mathcal{N}^T A(-K_v \dot{q})$$

You can find $\mathcal{N}$ (the projection matrix from joint space to task null space) by calling the function `nullspaceMatrix()` from `ModelInterface`. Plot the operational point trajectory as well as the joint trajectories now. Compare with before.

2. We've been using the joint space gravity vector, which simultaneously cancels gravity in all joints. Instead, we will use the operational space gravity vector:

$$\begin{aligned} P &= \bar{J}^T g \\ F &= \Lambda(k_p(x_d - x) - k_v \dot{x}) + p \\ \Gamma &= J_v^T F \end{aligned}$$

$\bar{J}$ is also a function in `ModelInterface`. This is the dynamically consistent jacobian inverse (generalized inverse of jacobian with mass matrix weighting).

(a) Control the end effector to $\begin{bmatrix} 0.5 \\ 0 \\ 0.8 \end{bmatrix}$ with null space damping or joint space damping turned off (only task space damping). What is happening to the motion of the robot once we reach the desired position? *Hint: Wait!

(b) Why do we not see this behavior with joint space gravity compensation?

3. We will run our controller to track a circular trajectory.

(a) Test your controller (with null space damping and joint space gravity) by drawing a circular trajectory with:

$$x_d = x_{init} + \begin{bmatrix} -0.1 \\ 0 \\ 0.05 \end{bmatrix} + 0.2 \begin{bmatrix} \sin(2\pi t) \\ \cos(2\pi t) \\ 0 \end{bmatrix}$$

x_{init} is the location we end up after the joint space controller is done (given in code). Choose gains that track the trajectory well while maintaining critical damping.

Plot the trajectory for a few cycles as well as the desired trajectory for these four scenarios:

i. Well damped in op space
ii. Underdamped in op space
iii. Well damped in op space without the Λ matrix (same gains as i.)
iv. Underdamped in op space without the Λ matrix (same gains as ii.)

4. Attach your SAI code here.