Advanced Robotic Manipulation (CS 327A) Homework #3

(Spring 2022/2023) Due: **Friday, April 05, 11:59pm**

In this homework you will further explore the control of redundant robots in operational space and understand the effects of the inertia-weighted generalized inverse of the Jacobian $\bar{J}$ (a.k.a. the dynamically consistent inverse). You will also investigate using your robot to apply forces using the Unified Motion and Force control scheme.

Code Submission Instructions

For Problems 2 and 3, include a screenshot of your FILL ME IN code section in your homework submission.

Additionally, you should record a video of the simulation running in each part, upload this video somewhere accessible (Google Drive or Youtube both work well), and then include a link to the video for each part.

Alternatively, you may come by Wesley's office hours or email him to schedule a Zoom session in order to show your working simulations.

Code Update

You can find the new homework code by navigating to the cs327a directory in your computer and following the steps below:

1. Stash your changes and pull the new updates (which will include the solutions) by using the following commands:

```
$ git stash
$ git pull
```

2. Retrieve your own changes (that you previously stashed) by typing:

```
$ git stash pop
```

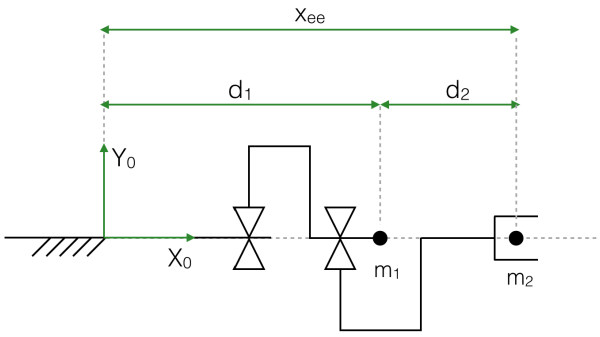
3. Build the updated repo by navigating to cs327a/build and running:

```
$ cmake .. && make
```

4. The new code is available in the hw3 folder

Problem 1 - A Simple Dynamically Consistent Inverse

Consider this simple 2 DoF Robot, which is redundant with respect to the task of moving along the X_0 axis.



The masses for link 1 and link 2 are $m_1 = 6\,\text{kg}$ and $m_2 = 3\,\text{kg}$ respectively.
The robot has the following simple mass matrix:

$$A = \begin{bmatrix} m_1 + m_2 & m_2 \\ m_2 & m_2 \end{bmatrix} = \begin{bmatrix} 9 & 3 \\ 3 & 3 \end{bmatrix}$$

(a) Let x_{ee} be the displacement of the end-effector along the X_0 axis.
Find the task Jacobian J such that

$$\dot{x}_{ee} = J\dot{q}, \;\; q = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$$

Hint: Note that this will be a 1 row jacobian!

(b) Find J^+ and $[I - J^+J]$. Then express the family of "instantaneous inverse kinematic" solutions in terms of δx_{ee} and δq_0 (as seen in pg. 86 of the course reader):

$$\delta q = J^+ \delta x_{ee} + [I - J^+J]\delta q_0$$

(c) Find $J^\#$ and $[I - J^\#J]$ where

$$J^\# = W^{-1}J^T(JW^{-1}J^T)^{-1}, \;\; W = \begin{bmatrix} w_1 & 0 \\ 0 & w_2 \end{bmatrix}.$$

Then express the family of weighted "instantaneous inverse kinematic" solutions in terms of δx_{ee} and δq_0. Note that w_1, w_2 are just two generic weights:

$$\delta q = J^\# \delta x_{ee} + [I - J^\#J]\delta q_0$$

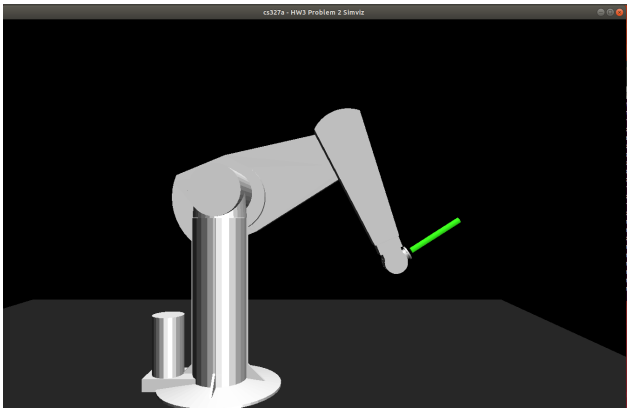
(d) Given an arbitrary nonzero δx_{ee} and a $\delta q_0 = 0$, compare the solutions obtained by using both the pseudo-inverse J^+ and the inertia-weighted inverse $\bar{J}$ (where $\bar{J} = J^\#$ when the latter is calculated with $W = A$). Explain how and why the solution obtained using the inertia-weighted inverse differs from the one obtained with the pseudo-inverse.

Hint: Pay particular attention to which joints are moved in each solution.

Problem 2 - Using the Dynamically Consistent Inverse for Null Space Motion

Now let us investigate the control of task-redundant robots through joint torques. The PUMA(560) arm is a 6-DOF manipulator you are probably familiar with through CS223A and the lecture slides. With respect to the 3-DOF positioning task at the end-effector, the robot is redundant.

The goal of the control is to hold the tip of the green cylinder stationary while also moving the second joint (using the Puma's redundancy). The end effector starts off in the following position:



We will be using a dynamically-decoupled operational space PD controller to hold the tip steady, and then a dynamically-decopuled joint space PD controller to perform the joint motion. This joint space controller will be projected into the null space of the operational task's Jacobian.

(a) Build your code following the Code Update instructions. You should see a new directory called `hw3` in your `bin` folder. Run the `hw3_p2_simviz` executable, and you should see the Puma robot falling under the effects of gravity.

(b) **Null space motion w/ pseudo-inverse**

As a first attempt, we will try to accomplish these two tasks using the psuedoinverse.

In the area marked as "FILL ME IN" of the `hw3_p2_controller.cpp` file, implement the following controller in **PART1** of the switch block:

$$\Gamma = J_v^T(\Lambda_v(-k_{px}(x - x_d) - k_{vx}\dot{x}) + p) \; + [I - J_v^T J_v^{+T}](A(-k_{pj}(q - q_d) - k_{vj}\dot{q}) + g)$$

The first half of the controller is the operational space end-effector positioning task. The second half is the joint space motion tracking task. The matrix $[I - J_v^T J_v^{+T}]$ is equivalent to $[I - J_v^+ J_v]^T$, and represents the transpose of the null space projection matrix associated with J_v.

Since the task is just positioning the end effector, the task Jacobian is equivalent to the linear velocity Jacobian J_v at the end-effector tip. x is the end-effector tip position and x_d is the end-effector tip desired position, both calculated with respect to the base frame. q_d is our desired joint motion. We will use the following parameters:

$$x_d = [-0.15 \;\; 0.81 \;\; 0.58]^T$$

$$k_{px} = 50 \;\; k_{vx} = 20 \;\; k_{pj} = 50 \;\; k_{vj} = 20$$

The above values are already set for you in the variables `ee_des_pos`, `op_task_kp`, `op_task_kv`, `joint_task_kp`, and `joint_task_kv` respectively in the starter code.

Additionally, set the desired joint motion to be:

$$q_d = \begin{bmatrix} q_1 \\ -\frac{\pi}{8} + \frac{\pi}{8}\sin\frac{2\pi t}{10} \\ q_3 \\ q_4 \\ q_5 \\ q_6 \end{bmatrix}$$

You will have to define this one yourself in your code.

Make sure that your final control torques are being stored in the `command_torques` variable.

To see this controller in action, run the `launch_p2.sh` script from your `bin/hw3` folder with a "1" argument following it like so:

```
$ ./launch\_p2.sh 1
```

Question: Describe the overall behavior of the controller. In particular, does the controller succeed at holding the end effector position stationary?

(c) Null space motion w/ dynamically consistent inverse

Now let us accomplish the same task using the dynamically consistent Jacobian inverse $\bar{J}$ instead.

Implement this controller below in the **PART2** of the switch block

$$\Gamma = J_v^T(\Lambda_v(-k_{px}(x - x_d) - k_{vx}\dot{x}) + p) + [I - J_v^T\bar{J}_v^T](A(-k_{pj}(q - q_d) - k_{vj}\dot{q}) + g)$$

where x_d, q_d, and gains are the same as in part (b). Note that the main difference here is that the null space projection matrix is calculated using $\bar{J}_v$ rather than J_v^+.

To see this controller, run the `launch_p2.sh` script with a "2" argument

Question: Now, does the end effector position remain stationary? Briefly explain why the dynamically consistent generalized inverse works but the pseudo-inverse does not.

Note: The end-effector tip is located at $[-0.2 \;\; 0 \;\; 0]$ in the last joint frame, and is available through the variable `ee_pos_local` in the starter code. So, you have to use the following commands from the model interface to evaluate the position and linear velocity Jacobian at the tip of the end-effector.

```
// calculate end-effector tip position
Eigen::Vector3d ee_x;
robot->position(ee_x, ee_link_name, ee_pos_local);

// calculate end-effector tip linear velocity Jacobian
Eigen::VectorXd Jv(3, robot->dof());
robot->Jv(Jv, ee_link_name, ee_pos_local);
```

Problem 3 - Unified motion and force control

In this problem, you will control the KUKA-IIWA arm to track a desired motion at its end effector just like in Homework 2. However, certain directions will also be controlled to apply open loop forces in order to maintain contact with a window while performing the windshield wiping motion.

The desired end-effector motion and constant applied forces are described by:

$$y_d = 0.5 + 0.1 \cos \frac{2\pi t}{5}$$
$$z_d = 0.65 - 0.05 \cos \frac{4\pi t}{5}$$

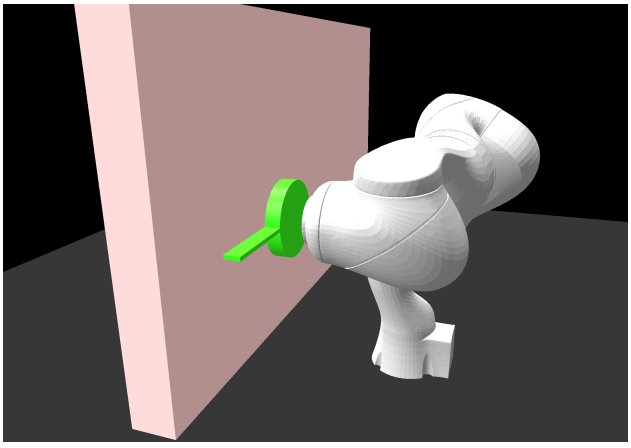
$$F_{x,d} = 10$$
$$M_{y,d} = 0$$
$$M_{z,d} = 0$$

where t is time in seconds.

The desired orientation trajectory (same as HW 2) is represented in Euler parameters as $\lambda_d = (\lambda_{0d}, \lambda_{1d}, \lambda_{2d}, \lambda_{3d})$ where

$$\lambda_{0d} = \lambda_{2d} = \frac{1}{\sqrt{2}} \sin\left(\frac{\pi}{4} \cos \frac{2\pi t}{5}\right)$$
$$\lambda_{1d} = \lambda_{3d} = \frac{1}{\sqrt{2}} \cos\left(\frac{\pi}{4} \cos \frac{2\pi t}{5}\right)$$

As shown below, the force will allow us to keep contact with the window surface located at $x = 0.05(m)$.



(a) Force, motion selection matrices

Find the selection matrices Ω and $\bar{\Omega}$ that separate the controlled force directions from the controlled motion directions at the end-effector.

(b) Unified motion and force controller

Modify the controller from Homework 2 Part (c) (including gravity compensation and joint damping in the null space of the task) to produce the desired response specified above. Additionally, this time you should use the dynamically consistent inverse $\bar{J}$ rather than the pseudo-inverse to compute your dynamics.

Question: Write out your modified controller.

You do not need to perform closed loop force control for maintaining the desired force. A feedforward term is enough.

**** Hint **** You will have to add an operational space damping term to dampen the motion along x and about the y and z axis. This should prevent the simulation from going unstable. At first the robot is in free space so there is nothing to counteract the 10N force. Therefore, unless you dampen that direction, it will move quite violently towards the wall and then not be able to maintain contact. The damping term should look something like: $\Lambda * selectForces * (-Kvx * v)$

(c) Simulation

Working now with the `hw3_p3_controller.cpp` file, implement your proposed controller in the area marked as "FILL ME IN". The resulting motion should be identical to the motion you observed in Homework 2, except the end-effector should move in the x direction until it reaches the window and then slide against it.

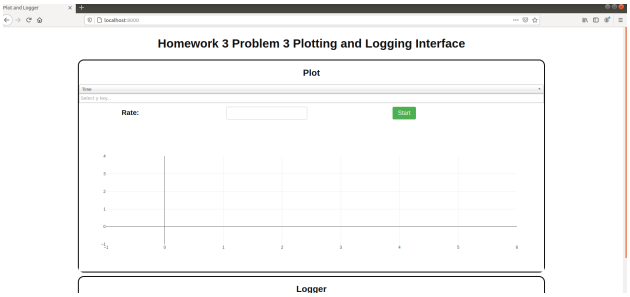
You can start this simulation by running the `launch_p3.sh` script (no additional arguments for this one).

Question: How could we change our controller to obtain the same behavior if the window was tilted 45 degrees about the y axis (relative to its current orientation)?

(d) Real-Time Data Observation with sai2-interfaces

The launch script for this problem also starts up a plotting and logging html template through sai2-interfaces. With this interface, we can directly observe key features of the simulation in real time.

Open up a web browser and enter the following into the address bar: `http://localhost:8000/`. You should see something like the following interface:



Using this interface, plot the sensed end-effector force in real time as the robot moves. The data is stored in the following redis key: `cs327a::hw3::robot::Kuka-IIWA::sensors::ee_force`. Check the sai2-interface tutorials first if you need assistance.

Question: Does the force always match our desired 10N? How could we improve the performance of the force control? Include an image of your plotted force with your response.