Make sure to provide justification for your answers. This includes labeling all of your plots (title, axes, legend, etc.) and explaining what is shown in the plots. Otherwise, you will lose points.

In this homework assignment, you will implement joint space control for the Franka Panda and experiment with control gains in simulation.

To download the assignment, you’ll have to pull the latest updates from cs225a.git. If you want to keep your progress from HW0, 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. 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. Now you’re ready to start Homework 1!

Note: You may use simviz_hw1_slow.cpp instead of simviz_hw1.cpp if your computer has trouble running the simulation smoothly. The normal simviz runs the physics simulation loop at 1,000 Hz, whereas the slow version runs the loop at 200 Hz.

Note: For all problems, the initial robot configuration is $q_0 = (-80^\circ, -45^\circ, 0^\circ, -125^\circ, 0^\circ, 80^\circ, 0^\circ)$. The desired robot configuration is $q_d = (90^\circ, -45^\circ, 0^\circ, -125^\circ, 0^\circ, 80^\circ, 0^\circ)$.
1. Let the joint positions and velocities of the robot be given by \( q \) and \( \dot{q} \), respectively. Let \( q_d \) be the desired joint positions of the robot. Implement the joint space control law:

\[
\Gamma = -k_p(q - q_d) - k_v\dot{q}
\]

where \( k_p \) and \( k_v \) are your control gains.

(a) Tune your gains to achieve critical damping on joint 1 with \( k_p = 400 \) and report your chosen \( k_v \).

(b) Move the robot from its initial configuration to its desired configuration. Plot the joint trajectory for joints 1, 3, and 4. Why do some joints converge closer to the desired position than others?

Solution:

All the plots for all the questions will have the same axis range (for corresponding joints) so it is easy to compare the plots. We tune the gain by trying values and observing the response of the first joint. With \( k_v = 51 \) we see a little overshoot in the response of the joint. With \( k_v = 52 \) we see no overshoot. We will chose \( k_v = 52 \). We plot the trajectory for joints 1, 3 and 4.

![PD controller, q vs qd](image)

Figure 1: System response for joints 1, 3 and 4 when implementing the controller from question 1

We see that joint 1 converges with no steady state error and no overshoot (because we tuned the controller for this joint). Joint 3 presents big disturbances that are due to the dynamic coupling in the manipulator. Joint 4 shows both a disturbance during the transient phase and a steady state error. The disturbance is due to the dynamic coupling. The steady state error is due to the gravity that we did not compensate for in this controller. The reason why gravity has no effect on the first and third joint is because of the way the joints are aligned at the final configuration. Gravity causes no moment around the joint axis for these two joints.
2. Now, implement the joint space control law:

\[ \Gamma = -k_p(q - q_d) - k_v \dot{q} + g(q) \]

where \( g(q) \) is the joint space gravity compensation vector.

(a) Tune your gains to achieve critical damping on joint 1 with \( k_p = 400 \) and report your chosen \( k_v \).

(b) Again, move the robot from its initial configuration to its desired configuration. Plot the joint trajectory for joints 1, 3, and 4. Compare these plots to the ones in Problem 1 and explain what you see.

Solution:

We do not change much the closed loop dynamic behavior of the system when we add the gravity term (gravity has more of a static effect). Therefore, the previous gains will work. We chose \( k_v = 52 \). We plot the trajectory for joints 1, 3 and 4.

The behavior of joint 1 and 3 did not change. This is expected since gravity is a static effect. We see that now, joint 4 has no steady state error. This is thanks to the gravity compensation the controller. However, joints 3 and 4 still shows disturbance effects due to dynamic coupling.
3. Now, implement the joint space control law that takes into account the dynamics of the robot:

\[
\Gamma = A(q)(-k_p(q - q_d) - k_v \dot{q}) + g(q)
\]

where \(A(q)\) is the joint space mass matrix.

(a) Tune your gains to achieve critical damping on joint 1 with \(k_p = 400\) and report your chosen \(k_v\).

(b) Again, move the robot from its initial configuration to its desired configuration. Plot the joint trajectory for joints 1, 3, and 4. Compare these plots to the ones in Problem 2 and explain what you see.

Solution:

The closed loop dynamics for this system correspond to a unit mass, second order linear system (with some disturbances due to the fact we neglect coriolis effects). Therefore, we will chose \(k_v = 2\sqrt{k_p} = 40\) in order to get critical damping.

![PD controller with gravity compensation and inertial compensation, q vs qd](image)

Figure 3: System response for joints 1, 3 and 4 when implementing the controller from question 3

We see that the disturbance effect is much lower on joint 3. This is expected since the coupling between joint 1 and 3 comes from the mass matrix only (in the configuration of the robot we are trying to maintain). Indeed, in this configuration, all the mass of the robot is in a plane that contains the axis of joint 1 and 3, so the coriolis effect is mostly handled by the structure of joint 3 and not the joint itself. However, joint 4 still shows important disturbances. This is because the coupling effect between joint 1 and 4 has an important component that is due to coriolis.
4. Now, implement the joint space control law:

\[ \Gamma = A(q)(-k_p(q - q_d) - k_v\dot{q}) + b(q, \dot{q}) + g(q) \]

where \( b(q, \dot{q}) \) is the joint space coriolis and centrifugal force compensation vector.

(a) Tune your gains to achieve critical damping on joint 1 with \( k_p = 400 \) and report your chosen \( k_v \).

(b) Again, move the robot from its initial configuration to its desired configuration. Plot the joint trajectory for joints 1, 3, and 4. Compare these plots to the ones in Problem 3 and explain what you see.

Solution:

The closed loop dynamics for this system correspond to a unit mass system second order linear system. Therefore, we will choose \( k_v = 2\sqrt{k_p} = 40 \) in order to get critical damping.

![PD controller with gravity compensation and dynamic decoupling, q vs qd](image)

Figure 4: System response for joints 1, 3 and 4 when implementing the controller from question 4

Now, the tracking is almost perfect. Joint 1 shows a nice step response and joint 3 and 4 stay at the desired configuration. This is expected since now the closed loop dynamics of the system are perfectly decoupled by the controller. The small errors that we still see are due to integration errors in the simulation and discretization of the controller.
5. Now, let’s assume the robot is carrying a payload of 2.5 kg at the end-effector. In order to simulate this, change the mass of “link 7” in `panda_arm_simulation.urdf` from 0.5 kg to 3.0 kg. Using the same controller from Problem 4, move the robot from its initial configuration to its desired configuration. Plot the joint trajectory for joints 1, 3, and 4. Compare these plots to the ones in Problem 4 and explain what you see. Hint: You will need to call the cmake command in order to copy the modifications to your urdf file to the runtime directory.

**Solution:**

Here is the plot we obtain when we add mass to the simulation only.

![Error in payload mass](image)

![Joint angles](image)

![Time (seconds)](image)

Figure 5: System response for joints 1, 3 and 4 when implementing the controller from question 5 and when the controller has model errors

We see that the disturbances during the transient phase and the steady state error for joint 4 appears again. Indeed, the fact that we added mass at the end effector and did not take it into account in the controller will cause our estimate of the mass matrix $M$, the coriolis force $b$ and even the gravity vector $g$ to be incorrect. For example, the gravity is under estimated so joint 4 converges to a lower value than the desired one. And because of the errors on $M$ and $b$, the dynamic decoupling is not perfect anymore so we see disturbances during the transient phase on both joints 3 and 4.

6. **Extra credit:** What do you need to do in the controller in order to take the payload into account (you can’t modify the urdf file for the controller). Implement this change and then move the robot from its initial configuration to its desired configuration. Plot the joint trajectory for joints 1, 3, and 4. Compare these plots to the ones in Problem 5 and explain what you see.
Solution:

If we know the added mass at the end effector, we can try and add it in our model in the controller. The easiest way would be to change the urdf file. However, in a lot of cases, when you program a real robot, you will not have access to a parametric dynamic model of the robot, you will just have a function that is a black box and that you can call to get the mass matrix, gravity and coriolis. In this case, you will have to manually compensate for any added load on the robot since you don’t have access to the model this function uses. For the gravity, it is fairly easy. You can compute the added gravity term by just compensating the joint torques caused by the gravity force on your added mass. Here, if we call $m = 2.5kg$ the mass of your payload and $J_{vp}$ the jacobian at the center of mass of the payload, the gravity term will just be

$$g_{add} = m \cdot J_{vp}^T \cdot [0, 0, 9.81]^T$$

To take the load into account in the mass matrix, we can use the explicit formula for the mass matrix. We see that the total mass matrix will be the sum of the previous one, plus the kinetic energy terms due to the added payload. Here, since there is no inertia, we get

$$M_{add} = m J_{vp}^T J_{vp}$$

Finally, for coriolis, in practice, there is no easy way to get it. We would need to have access to the derivative of the jacobian or the mass matrix, which are quantities that are not well estimated in practice. Therefore, we will neglect it. One way of reducing the effect of the coriolis term though will be to limit the velocity of the robot. This is what we do in the right plot below.

![System response for joints 1, 3 and 4 when compensating for an added mass at the end effector.](image)

We see on the left plot that the tracking is better but not perfect, as expected since we did not take the coriolis force into account. On the right plot, we limit the angular velocity for joint 1, and that reduces the coriolis disturbance effect to a point where we do not see it anymore. In practice, you will always limit the velocity of your robot for safety reasons (both for the human and the robot) and therefore, in practice it will be often recommended to completely neglect coriolis altogether since its effect will be small when your robot moves slowly, and it is difficult to estimate properly.

7. Submit your SAI code (`hw1.cpp`).