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 trajectory tracking, joint limit avoidance, orientation control and velocity saturation for the Panda.

To download the assignment, you’ll have to pull the latest updates from cs225a.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`. 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.
1. In the last homework, you implemented an operational space PD controller with null space posture control similar to the one below:

\[
\begin{align*}
F &= \Lambda (-k_p (x - x_d) - k_v \dot{x}), \\
\Gamma &= J_v^T F + N^T (-k_{pj} (q - q_d) - k_{vj} \dot{q}) + g.
\end{align*}
\]

where \(k_p\) and \(k_v\) are your operational space control gains, \(\Lambda\) is your operational space mass matrix, \(k_{pj}\) and \(k_{vj}\) are your posture control gains, and \(g\) is your joint space gravity vector. In this problem, use null space posture control to track a desired joint configuration of \(q_d = [0, 0, 0, 0, 0, 0, 0]^T\). In the following scenarios, the desired end-effector trajectory is given by the following:

\[
x_d = \begin{bmatrix} 0.3 \\ 0.1 \\ 0.5 \end{bmatrix} + 0.1 \begin{bmatrix} \sin(\pi t) \\ \cos(\pi t) \\ 0 \end{bmatrix}.
\]

Use the following gains for all scenarios:

\[
\begin{align*}
k_p &= 100.0 \\
k_v &= 20.0 \\
k_{pj} &= 50.0 \\
k_{vj} &= 14.0
\end{align*}
\]

(a) In this part, only use the controller given at the beginning of this problem. Plot the actual vs. desired end-effector trajectories (not the joint trajectories). Does your actual trajectory perfectly match the desired trajectory? Why or why not?

**Solution:**

The left plot shows the end effector trajectory with the controller implemented here. We see that there is a delay between the end effector actual trajectory and the desired one. This is due to the fact we apply a controller that is designed to perform regulation to a tracking problem, mainly, we do not consider the velocity and acceleration of the trajectory and try to regulate these to zero. We could decrease the error by increasing the gains.

(b) Now, we will add proper trajectory tracking to better track the desired end-effector trajectory. In order to do this, we need the appropriate values of \(\dot{x}_d\) and \(\ddot{x}_d\). Using the desired end-effector trajectory \(x_d\), derive the expressions for the following:

i. \(\dot{x}_d\)
ii. \(\ddot{x}_d\)

**Solution:**

We just take the time derivative of \(x_d\)

\[
\dot{x}_d = 0.1\pi \begin{bmatrix} \cos(\pi t) \\ -\sin(\pi t) \\ 0 \end{bmatrix}.
\]

\[
\ddot{x}_d = 0.1\pi^2 \begin{bmatrix} -\sin(\pi t) \\ -\cos(\pi t) \\ 0 \end{bmatrix}.
\]
(c) After adding proper trajectory tracking, our operational space controller is given by the following:

\[ F = \Lambda(\ddot{x}_d - k_p(x - x_d) - k_v(\dot{x} - \dot{x}_d)), \]
\[ \Gamma = J^T v F + N^T (-k_{pq}(q - q_d) - k_{qv}\dot{q}) + g. \]

Implement the new controller and track the desired trajectory. Plot the actual vs. desired end-effector trajectories (not the joint trajectories). Compare your results to 1 (a).

**Solution:**

The plot on the right shows the trajectory tracking. There is an initial error due to the initial robot configuration not matching the desired initial point in the tracked circle, however, once the robot catches up with the desired trajectory, the tracking seems perfect. On a real system, there might be disturbances during the trajectory (if someone touches the robot for example), and we could reduce the effect of these disturbances on the tracking error by increasing the gains. However, in simulation, there are no such effects.
2. Now, we will utilize null space posture control to avoid joint limits. Let \( q_i \) and \( \bar{q}_i \) be the upper and lower bounds on the \( i \)th joint position \( q_i \). We construct the following potential function:

\[
V_{mid}(q) = k_{mid} \sum_{i=1}^{n} \left( q_i - \frac{\bar{q}_i + q_i}{2} \right)^2,
\]

where \( k_{mid} \) a constant gain and \( n \) is the number of joints in the manipulator. The following gradient of this potential function provides the required attraction to the mid-range joint positions of the manipulator:

\[
\Gamma_{mid} = -\nabla V_{mid}.
\]

In order to avoid the interference of these additional torques with the end-effector dynamics, we project these additional torques into the dynamically consistent null space. The resulting torques are then given by \( N^T \Gamma_{mid} \). Next, we introduce a null space damping term \( N^T \Gamma_{damp} \), where

\[
\Gamma_{damp} = -k_{damp} \dot{q}.
\]

The operational space controller is given by the following:

\[
F = \Lambda(-k_p(x - x_d) - k_v \ddot{x}),
\]

\[
\Gamma = J^T_v F + N^T \Gamma_{mid} + N^T \Gamma_{damp} + g.
\]

For the intent of this problem, we will use 1-based indexing for the joints. This means that the first joint is joint 1 and the last joint is joint 7. Also, the lower and upper joint limits for the Panda are given by the following (these are not the real joint limits for the manipulator):

\[
q = [-165^\circ, -100^\circ, -165^\circ, -170^\circ, -165^\circ, 0^\circ, -165^\circ]^T,
\]

\[
\bar{q} = [165^\circ, 100^\circ, 165^\circ, -30^\circ, 165^\circ, 210^\circ, 165^\circ]^T.
\]

(a) Write out the analytical expression for \( \Gamma_{mid} \). Show your work.

Solution:

\[
\Gamma_{mid} = -\nabla V_{mid} = -k_{mid} \nabla \sum_{i=1}^{n} \left( q_i - \frac{\bar{q}_i + q_i}{2} \right)^2 = -k_{mid} \begin{bmatrix}
\frac{\partial}{\partial q_1} \left( q_1 - \frac{\bar{q}_1 + q_1}{2} \right)^2 \\
\vdots \\
\frac{\partial}{\partial q_n} \left( q_n - \frac{\bar{q}_n + q_n}{2} \right)^2
\end{bmatrix}
\]

\[
= -2k_{mid} \begin{bmatrix}
\left( q_1 - \frac{\bar{q}_1 + q_1}{2} \right) \\
\vdots \\
\left( q_n - \frac{\bar{q}_n + q_n}{2} \right)
\end{bmatrix} = -2k_{mid} \left( q - (\bar{q} + q)/2 \right)
\]

(b) With your expression for \( \Gamma_{mid} \), compare the command torques \( \Gamma \) in this problem with the command torques from problem 1 (\( \Gamma = J^T_v F + N^T(-k_{pj}(q - q_d) - k_{vj}\dot{q}) + g \)). Specifically, compare the terms \( N^T \Gamma_{mid} + N^T \Gamma_{damp} \) and \( N^T(-k_{pj}(q - q_d) - k_{vj}\dot{q}) \).
Solution:

If we compare $\Gamma_{\text{mid}} = -2k_{\text{mid}}(q - (\bar{q} + q)/2)$ and $-k_{pj}(q - q_d)$ we see that these terms have the same form and they will be equal if we choose $q_d = (\bar{q} + q)/2$ and $k_{\text{mid}} = k_{pj}/2$. Similarly, $\Gamma_{\text{damp}} = -k_{\text{damp}}\dot{q}$ and $-k_{vj}\dot{q}$ are equal if we chose $k_{\text{damp}} = k_{vj}$.

(c) Based on your analysis in part (ii), choose and report the values of $k_{\text{mid}}$ and $k_{\text{damp}}$ that achieve the same dynamic response as the one in problem 1.

Solution:

We choose $k_{\text{mid}} = 25.0$ and $k_{\text{damp}} = 14.0$.

(d) Here, we will move the Panda to the desired position $x_d = [-0.1, 0.15, 0.2]^T$. Use the following operational space controller (note that we do not use the potential field here):

$$F = \Lambda(-k_p(x - x_d) - k_v\dot{x}),$$
$$\Gamma = J_v^T F + N^T \Gamma_{\text{damp}} + g.$$ 

Plot your actual vs. desired end-effector trajectories. Are you able to track the desired end-effector position? Also plot the joint trajectories and joint limits for joints 4 and 6. Do these joints approach their joint limits?

Solution:

Experiment 2d

Figure 2: Robot end effector trajectory and joints 4 and 6 versus their limits

With this controller, we are able to track the desired position in theory. However, we see that the joints would go beyond their limits (joint 4). Indeed, we are not performing any kind of joint limit avoidance here so nothing guarantees that the joints will not violate their limits.

(e) Now, we will still move the Panda to the desired position $x_d = [-0.1, 0.15, 0.2]^T$. However, we will add the potential field to avoid joint limits in the null space of the task:
\[ F = \Lambda(-k_p(x - x_d) - k_v \dot{x}), \]
\[ \Gamma = J^T_v F + N^T \Gamma_{\text{mid}} + N^T \Gamma_{\text{damp}} + g. \]

Plot your actual vs. desired end-effector trajectories. Are you able to track the desired end-effector position? Also plot the joint trajectories and joint limits for joints 4 and 6. Do these joints approach their joint limits? Compare your results to 2 (d) and explain any differences that you see.

**Solution:**

**Figure 3:** Robot end effector trajectory and joints 4 and 6 versus their limits

with this new controller, we are still able to track the desired position, and at steady state, the joints stay inside their bounds. Adding the joint limit avoidance term improves joint limit avoidance as expected. However, this is not perfect as we see in the transient phase, joint 4 will violate its limit. In this case, increasing \( k_{\text{damp}} \) would solve that issue.

(f) Now, we will change the Panda’s desired position to \( x_d = [-0.65, -0.45, 0.7]^T \). Use the same operational space controller as the previous part:

\[ F = \Lambda(-k_p(x - x_d) - k_v \dot{x}), \]
\[ \Gamma = J^T_v F + N^T \Gamma_{\text{mid}} + N^T \Gamma_{\text{damp}} + g. \]

Plot your actual vs. desired end-effector trajectories. Are you able to track the desired end-effector position? Also plot the joint trajectories and joint limits for joints 4 and 6. Do these joints approach their joint limits? Compare your results to 2 (e) and explain any differences that you see.
Solution:

Now, with this new desired position, the joints stay always inside their limits even though they approach the limits dangerously (joint 4). The desired position is still perfectly tracked. We see that this controller helps avoid the joint limits but does not provide any guarantees.

(g) Again, we will move the Panda to the desired position \( x_d = [-0.65, -0.45, 0.7]^T \). However, we will not project the torques designed to avoid joint limits onto the null space. The new operational space controller is given by the following:

\[
F = \Lambda(-k_p(x - x_d) - k_v\dot{x}), \\
\Gamma = J^T_v F + \Gamma_{mid} + N^T \Gamma_{damp} + g.
\]

Plot your actual vs. desired end-effector trajectories. Are you able to track the desired end-effector position? Also plot the joint trajectories and joint limits for joints 4 and 6. Do these joints approach their joint limits? Compare your results to 2 (f) and explain any differences that you see. In your opinion, which controller is preferable between the one from part (f) and part (g)? Explain why.

Solution:

With this controller, the tracking is not perfect anymore. Since we do not put the joint limit avoidance in the nullspace of the task anymore, the two interfere with each other. As a result, the robot compromises on the task in order to better avoid the joint limits, and we see that in the right plot, the joints end up farther away to their joint limits when compared to the previous question.
Figure 5: Robot end effector trajectory and joints 4 and 6 versus their limits

(h) **Extra credit**: What would you do to implement a controller that guarantees joint limit avoidance, but does not affect the task when all the joints are far from their limits?

**Solution:**

We saw in this exercise that putting the joint avoidance term in the nullspace of the task will not guarantee joint limit avoidance. If the task violates joint limits, then in general we want to give up part of the task. However, in general we want the task to be fulfilled. Another thing to keep in mind is that the potential function we chose here does not guarantee joint limit avoidance since the repulsive force does not become infinity at the joint limit.

One idea to guarantee better joint limit avoidance is to create a potential function that becomes infinity when the joints reach their limits, and become zero when the joints are far enough from their limits. We can then use a gradient decent with this potential and apply the resulting torques to the robot in addition to the normal control torques. When the robot is far from its limits, the gradient torque is zero so the task is not affected, and when we get close to the limit, a force that will reach infinity will push the robot away (so no task force will be able to overcome that). An other idea is to remove the controllability of the joints that get close to their limits from the task. We can do that by projecting the task into the nullspace of the jacobian of the joints close to their limits.
3. We will implement orientation control with the full jacobian $[J_v\ J_\omega]$. 

Recall orientation error and full control is:

$$\delta \phi = -\frac{1}{2} \sum_{n=1}^{3} R_i \times (R_d)_i$$

$$F = \Lambda_0 \begin{bmatrix} k_p(x_d - x) - k_v \dot{x} \\ k_p(-\delta \phi) - k_v \omega \end{bmatrix}$$

$$\Gamma = \begin{bmatrix} J_v^T \\ J_\omega \end{bmatrix} F - N^T k_v \dot{q} + g$$

where $R_i$ and $(R_d)_i$ are the i-th columns of $R$ and $R_d$, respectively, and $\omega$ is the angular velocity. Note that you need to use the function `robot->J0` in order to get the Jacobian in the form $[J_v\ J_\omega]$.

(a) Control your robot to orientation:

$$R_d = \begin{bmatrix} \cos \frac{\pi}{3} & 0 & \sin \frac{\pi}{3} \\ 0 & 1 & 0 \\ -\sin \frac{\pi}{3} & 0 & \cos \frac{\pi}{3} \end{bmatrix}$$

and position:

$$x_d = \begin{bmatrix} 0.6 \\ 0.3 \\ 0.5 \end{bmatrix}$$

Plot the actual vs. desired end-effector trajectories. Also plot $\delta \phi$ over time. Explain what you see in your plots. Choose gains appropriately to achieve good tracking but critical damping. If your plot is not precise enough because the robot is moving too fast, reduce your gains.

Solution:

The robot moves to the desired position and then rotates to the desired orientation. During the rotation phase, we see some position error. This is due to the fact that we are not compensating for coriolis effects in this controller and the robot rotates quite fast. The orientation behavior seems weird. The orientation error is zero at first and then it increases and finally decreases. This is due to the specific representation we chose for the orientation. The term $\delta \phi$ can be zero when the frames are not aligned. However, the controller we implement for orientation is a stabilizing equilibrium only when the frames are aligned. We see it very clearly here since the orientation error in $x$ and $z$ starts increasing exponentially right away when the controller starts. Usually we want to perform interpolation to move the robot orientation in order to avoid these cases.
Figure 6: Robot end effector trajectory and orientation error
4. We will implement velocity saturation in the following operational space PD controller with null space posture control from the last homework:

\[
F = \Lambda(k_p(x_d - x) - k_v \dot{x})
\]

\[
\Gamma = J^T F + N^T M(-k_p_j(q - q_d) - k_v_j \dot{q}) + g
\]

Reach a desired position of \( x_d = [0.6, \ 0.3, \ 0.4]^T \).

(a) With operational space \( k_p = 200.0 \) and a critically damped \( k_v \), plot \( x, x_d \) (on the same plot) and \( \dot{x} \). Explain what you see.

**Solution:**

![Question 4a](image)

**Figure 7: Robot end effector trajectory and linear velocity**

The robot performs the usual step response, resulting in an impulse response for the velocity. As a result, it gets well above \( 0.1 \) m/s.

(b) Now, implement velocity saturation with:

\[
\dot{x}_d = \frac{k_p}{k_v} (x_d - x)
\]

\[
F = \Lambda(-k_v (\dot{x} - \nu \dot{x}_d))
\]

\[
\nu = \text{sat} \left( \frac{V_{max}}{|\dot{x}_d|} \right)
\]

where \( \text{sat}() \) is the saturation function:

\[
\text{sat}(x) = \begin{cases} 
  x & \text{if } |x| \leq 1.0 \\
  \text{sgn}(x) & \text{if } |x| > 1.0
\end{cases}
\]

and \( \text{sgn}(x) \) is the sign function. In this problem, use \( V_{max} = 0.1 \) m/s. Plot \( x, x_d \) (on the same plot) and \( \dot{x}, V_{max} \) (on the same plot) and compare your results to 4 (a).
Solution:

**Question 4b**

**x vs x_desired**

**end effector position (m)**

**y vs y_desired**

**end effector velocity**

**z vs z_desired**

Figure 8: Robot end effector trajectory and linear velocity

Now, the robot does not perform a step response anymore. The velocity gets limited as expected and the robot moves with constant velocity to its desired position. Once that position is reached, the error stays zero.

5. Submit your SAI code (`hw3.cpp`).