ME 446 Laboratory #4
Task Space PD control, Impedance Control

Report is due May 1st at 5pm. Lab sessions will be held the weeks of April 13th, April 20th and April 27th

Implement Task Space PD Control.

Task Space PD Control.

Implement task space control law of the form: (call the control flag, p.flag_ctrl = 0)

$$\begin{bmatrix}\tau_{M1}\\ \tau_{M2}\\ \tau_{M3}\end{bmatrix} = \begin{bmatrix} J^T * \begin{bmatrix}F_X\\ F_Y\\ F_Z\end{bmatrix}\end{bmatrix} = \begin{bmatrix} J^T * \begin{bmatrix} KP_X * \left(x^d - x\right) + KD_X * \left(\dot{x}^d - \dot{x}\right)\\ KP_Y * \left(y^d - y\right) + KD_Y * \left(\dot{y}^d - \dot{y}\right)\\ KP_Z * \left(z^d - z\right) + KD_Z * \left(\dot{z}^d - \dot{z}\right)\end{bmatrix}\end{bmatrix}$$

J^T is the velocity Jacobian of the end effector. $\begin{bmatrix}x^d\\ y^d\\ z^d\end{bmatrix}$ and $\begin{bmatrix}\dot{x}^d\\ \dot{y}^d\\ \dot{z}^d\end{bmatrix}$ are the desired position and desired velocity of the end effector in the task space. Note that if you are giving the system a step input or the desired velocity is not easily calculated, the desired velocity can be set to zero like done in previous labs. $\begin{bmatrix}x\\ y\\ z\end{bmatrix}$ and $\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix}$ are the measured position and measured velocity of the end effector in the task space.

Note: J^T is calculated by the function `fcn_J4.m`, $\begin{bmatrix}x\\ y\\ z\end{bmatrix}$ is calculated by forward kinematics function `fcn_p4.m` and $\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} = J * \begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\\ \dot{\theta}_3\end{bmatrix}$.

Initially just command the robot to hold position at one x,y,z point (0.381 m, 0 m, 0.254 m). Note: to make things simpler and less messy with the simulation of the robot we are switching to meters for the length measurements of the robot and the x,y,z coordinates of the end effector. 0.254 meters equals the 10 inch length of the robot arms. Note for this control law the Kp and Kd gains are much larger than the gains we used in the last two lab assignments. Part of the reason for this is we are using meters for the value of our x,y,z point of the end effector, which leads to a decreasing of the magnitude of the error terms. (Hint: Kp can be 1,000 to 100,000 and Kd can be 100 to 1,000.) For this lab we are also increasing the maximum torque the robot motors and produce from 10N to 100N. We are doing this so that torque limits are not a factor in this lab causing instability as you are trying to tune the gains of your PD controller. Obviously if we were wanting to try our controller on the actual robot, much more study would be needed to match the simulation to the actual robot where we know the maximum torque. After you implement your PD task space controller, tune your PD gains so that the robot stays close to its desired end effector position. Once tuned, pick another x,y,z point somewhat close to the initial point and have the robot step to that new x,y,z point say 2 seconds or so into the simulation. Does the robot move to the new point? This task space control can have problems if you command the robot to

an x,y,z point far away as the arm could go through singularities like switching from elbow up to elbow down configuration. You may have seen this when you were tuning your gains above, if they were too small.

Please make a video for this part stepping from one x,y,z point to another x,y,z point.

Impedance Control.

Impedance Control

First go back to holding the robot at one x,y,z point (0.381 m, 0 m, 0.254 m) and use p.flag_ctrl = 1. Then try weakening the X axis control gains, and while the controller is running, manually move the arm in the three axis directions and notice that it is weak (larger error) in the x axis and strong (smaller error) in the y and z axes. This "manually moving the arm is accomplished in simulation by adding an external force at the end effector of the robot. So in dyn_manip.m when p.flag_ctrl = 1,2,3 or 4, an external force (Fext) is applied at the end effector and stepped back and forth between [5N,5N,5N]' and [-5N,-5N,-5N]' every 1 second. This simulates you taking your hand and attempting to push the robot's end effector in the direction of those two force vectors. Try weakening Y axis. Z axis. X and Y axis. Weakening both the X and Y axis could help the robot insert a peg it is holding vertically into a vertical hole especially if the beginning of the hole had a bit of a counter sink.
Please make these two videos for this part:

1. Weaken world Y -axis
2. Weaken world X-axis and Y-axis

Now, what if you wanted to weaken the robot at its end effector along a different axis then a world coordinate axis? Here we need to use a rotation matrix. To do this let's call our new coordinate frame, frame N, and the world coordinate frame of the robot frame W. Frame N is found by rotating θ_z about the z axis and then rotating θ_x about the x axis and then θ_y about the y axis. This gives the rotation matrix

$$R_{zxy} = R_N^W = \begin{bmatrix} cos\theta_z & -sin\theta_z & 0 \\ sin\theta_z & cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta_x & -sin\theta_x \\ 0 & sin\theta_x & cos\theta_x \end{bmatrix} * \begin{bmatrix} cos\theta_y & 0 & sin\theta_y \\ 0 & 1 & 0 \\ -sin\theta_y & 0 & cos\theta_y \end{bmatrix},$$

$$R_{zxy} = R_N^W = \begin{bmatrix} cos\theta_z cos\theta_y - sin\theta_z sin\theta_x sin\theta_y & -sin\theta_z cos\theta_x & cos\theta_z sin\theta_y + sin\theta_z sin\theta_x cos\theta_y \\ sin\theta_z cos\theta_y + cos\theta_z sin\theta_x sin\theta_y & cos\theta_z cos\theta_x & sin\theta_z sin\theta_y - cos\theta_z sin\theta_x cos\theta_y \\ -cos\theta_x sin\theta_y & sin\theta_x & cos\theta_x cos\theta_y \end{bmatrix}$$

Given this rotation matrix, we can perform a coordinate transformation of Fx, Fy, Fz in the N frame to Fx, Fy, Fz in the world frame.

$$\begin{bmatrix} F_{x_W} \\ F_{y_W} \\ F_{z_W} \end{bmatrix} = R_N^W * \begin{bmatrix} F_{x_N} \\ F_{y_N} \\ F_{z_N} \end{bmatrix}$$

Controlling now x, y, z in the N frame, our control equations become

$$\begin{bmatrix} \tau_{M1} \\ \tau_{M2} \\ \tau_{M3} \end{bmatrix} = \begin{bmatrix} J^T * \begin{bmatrix} F_X \\ F_Y \\ F_Z \end{bmatrix} \end{bmatrix} = \begin{bmatrix} J^T * R_N^W \begin{bmatrix} KP_{X_N} * \left(x_N^d - x_N\right) + KD_{X_N} * \left(\dot{x}_N^d - \dot{x_N}\right) \\ KP_{Y_N} * \left(y_N^d - y_N\right) + KD_{Y_N} * \left(\dot{y}_N^d - \dot{y_N}\right) \\ KP_{Z_N} * \left(z_N^d - z_N\right) + KD_{Z_N} * \left(\dot{z}_N^d - \dot{z_N}\right) \end{bmatrix} \end{bmatrix}$$

For the robot arm to remain at a single x, y z point, it is much easier to think about commanding the arm to stay at a World x, y, z coordinate point and therefore the errors in World coordinates $(x_W^d - x_W), (y_W^d - y_W), (z_W^d - z_W)$. To rotate these errors in World coordinates to the N frame you will need another rotation matrix. This time a rotation from the World coordinate frame to the N frame, R_W^N. Properties of the rotation matrix prove that R_W^N is simply the transpose of R_N^W. $R_W^N = R_N^{W^T}$.

Looking in the above torque equation,$(x_N^d - x_N)$, $(y_N^d - y_N)$, $(z_N^d - z_N)$, are values in the N frame. So the World coordinate errors must be rotated into the N frame and then multiplied by the N frame KP and KD gains. The controller equations then become (make sure the note the parenthesizes)

$$\begin{bmatrix}\tau_{M1}\\ \tau_{M2}\\ \tau_{M3}\end{bmatrix} = \left[J^T * R_N^W * \left[\begin{bmatrix} KP_{X_N} & 0 & 0 \\ 0 & KP_{Y_N} & 0 \\ 0 & 0 & KP_{Z_N} \end{bmatrix} * R_W^N * \begin{bmatrix} (x_W^d - x_W) \\ (y_W^d - y_W) \\ (z_W^d - z_W) \end{bmatrix} + \begin{bmatrix} KD_{X_N} & 0 & 0 \\ 0 & KD_{Y_N} & 0 \\ 0 & 0 & KD_{Z_N} \end{bmatrix} * R_W^N * \begin{bmatrix} (\dot{x}_W^d - \dot{x}_W) \\ (\dot{y}_W^d - \dot{y}_W) \\ (\dot{z}_W^d - \dot{z}_W) \end{bmatrix} \right] \right]$$

Play around with some rotations of one axis first and the two axes. Remember that the external force we are added to the simulation to see the weak axes move more is [5,5,5]', so do not use a rotation in multiples of pi/4 so that the rotation creates the same direction as the external force.

Please make two videos for this part and discuss why the plots make sense remembering that the plots are in world coordinates. Make some x,y and/or x,y,z plots to show the correct perspectives that show the rotations. Play around here a bit try different angles positive and negative to make sure you understand how the rotations are moving the weak axis. Put a few 2D and 3D plots in your report that help show your understanding. There is some commented out code at the bottom of MAIN.m that shows you how to set the correct aspect ratio and use the view function.

For the videos:

1. Rotate about world Z-axis for pi/3 to get Frame N, and weaken N frame Y-axis.
2. Rotate the world X-axis by pi/6 and then the Y-axis by pi/3, and weaken N frame Z-axis.

Straight Line Following.

Equation of a straight line trajectory

To make the robot follow a straight line with a desired speed along that line's direction, three equations for x, y, z as a function of time can be derived given the start and end points along with the desired speed in inches/second.

$$Straight\ Line\ from\ (x_a, y_a, z_a)\ \ to\ \ (x_b, y_b, z_b)$$

$$\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta z\end{bmatrix} = \begin{bmatrix}x_b - x_a\\ y_b - y_a\\ z_b - z_a\end{bmatrix}, vector\ direction\ to\ travel$$

$$\mathrm{t_{total}} = \frac{distance\ between\ a\ and\ b}{desired\ speed\ along\ the\ line}$$

$$\begin{bmatrix}x^d\\ y^d\\ z^d\end{bmatrix} = \begin{bmatrix}\Delta x\\ \Delta y\\ \Delta z\end{bmatrix} * \frac{t - t_{start}}{t_{total}} + \begin{bmatrix}x_a\\ y_a\\ z_a\end{bmatrix}.$$

Repeating the same argument as above, for the robot arm to follow a straight line, it is much easier to think about commanding the arm to go from one World x, y, z coordinate point to another World x, y, z coordinate. To derive the torques required to make the robot follow this line defined in World coordinates you will need the same rotation matrix equations as above. Repeating what was stated above just to be clear, using the rotation from the World coordinate frame to the N frame, R_W^N. Properties of the rotation matrix prove that R_W^N is simply the transpose of R_N^W. $R_W^N = {R_N^W}^T$.

The desired line trajectory in the World frame must be rotated into the N frame and then multiplied by the KP and KD gains. The controller equations then become (which are the same equations as above)

$$\begin{bmatrix}\tau_{M1}\\ \tau_{M2}\\ \tau_{M3}\end{bmatrix} = \left[J^T * R_N^W * \left[\begin{bmatrix}KP_{X_N} & 0 & 0\\ 0 & KP_{Y_N} & 0\\ 0 & 0 & KP_{Z_N}\end{bmatrix} * R_W^N * \begin{bmatrix}(x_W^d - x_W)\\ (y_W^d - y_W)\\ (z_W^d - z_W)\end{bmatrix} + \begin{bmatrix}KD_{X_N} & 0 & 0\\ 0 & KD_{Y_N} & 0\\ 0 & 0 & KD_{Z_N}\end{bmatrix} * R_W^N * \begin{bmatrix}(\dot{x}_W^d - \dot{x}_W)\\ (\dot{y}_W^d - \dot{y}_W)\\ (\dot{z}_W^d - \dot{z}_W)\end{bmatrix}\right]\right]$$

Please make two videos for this part:

1. With all axes stiff and making the N frame the World Frame, so no rotation, command the robot's end effector to follow a straight line from one point to another point. Let's make the start point to be (0.35 m, -0.2 m, 0.25 m) and the end point to be (0.25 m, 0.3 m, 0.15 m) and time interval to be 2 sec. We are giving you the total time to go from one point to another instead of giving you the points and the traveling speed.
2. Again follow a straight line but this time make the direction perpendicular to the line weak and the direction along the line stiff. Let's make the start point to be (0.35 m, -0.2 m, 0.25 m) and the end point to be (0.25 m, 0.3 m, 0.25 m) in the World frame and time interval to be 2 sec. Notice that Z stays constant so all you need to find is the rotation angle that rotates the

x axis along the line and simply weaken the Y axis to make the robot weak perpendicular to the line.

Report:

The report for Lab 4 is just the Matlab code you developed, but well commented code. In addition the plots that you created to explain your understanding of the rotations. You can comment individual lines explaining what they are performing but in addition there should be paragraphs explaining in your own words what that section of code is accomplishing. Poorly commented code will receive a low grade.

Also, please use the `flag_movie` function provided in the code to make a video for each task you finished and name it correctly.