Objectives

- Add Friction Compensation for all three joints and investigate how well it works to cancel friction effects in the arm.
- Implement Task Space PD Control.
- Weaken Axis gains to achieve a Simple Impedance Control.
- Straight Line Following.


1.1 Identified Friction Curves.

Below are experimental friction plots that were identified on one CRS robot arm. Constant velocity and control effort (proportional to torque) was recorded and these plots show the best fit lines that were fit to the collected data. Use these straight line equations to implement friction compensation for each joint of your robot arm. Make sure to not hard code the friction coefficients because these values will need to be tuned for your robot arm. Once you have friction compensation implemented, adjust each joint's friction coefficients until you feel you have the correct coefficients for your robot arm.

Pseudocode:

```plaintext
if (joint_velocity > minimum_velocity) {
    u_fric = Viscous_positive*joint_velocity + Coulomb_positive;
} else if (joint_velocity < -minimum_velocity) {
    u_fric = Viscous_negative*joint_velocity + Coulomb_negative;
} else {
    u_fric = slope_between_minimums*joint_velocity;
}
```

Then after full control effort has been calculated, add u_fric to your calculated control effort. When testing only friction compensation simply set control effort only to u_fric.
Joint 1, +Viscous=0.2513, +Coulomb=0.3637, -Viscous=0.2477, -Coulomb=-0.2948

\[ u = 0.2513w + 0.3637 \]

slope = 3.6 from \( v = -0.1 \) to 0.1

\[ u = 0.2513w + 0.3637 \]

Joint 2, +Viscous=0.2500, +Coulomb=0.4759, -Viscous=0.2870, -Coulomb=-0.5031

\[ u = 0.25w + 0.4759 \]

slope = 3.6 from \( v = -0.05 \) to 0.05

\[ u = 0.25w + 0.4759 \]
Part 2: Implement Task Space PD Control.

2.1 Task Space PD Control.

Given that the transpose of the Jacobian for the CRS robot is equal to

\[
J^T = \begin{bmatrix}
-10 \sin \theta_{M1} * (\cos \theta_{M3} + \sin \theta_{M2}) & 10 \cos \theta_{M1} * (\cos \theta_{M3} + \sin \theta_{M2}) & 0 \\
10 \cos \theta_{M1} * (\cos \theta_{M2} - \sin \theta_{M3}) & 10 \sin \theta_{M1} * (\cos \theta_{M2} - \sin \theta_{M3}) & -10 \cos \theta_{M3} \\
-10 \cos \theta_{M1} * \sin \theta_{M3} & -10 \sin \theta_{M1} * \sin \theta_{M3} & -10 \cos \theta_{M3}
\end{bmatrix}
\]

and the forward kinematic equations for the CRS robot are

\[
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix} = \begin{bmatrix}
10 \cos \theta_{M1} * (\cos \theta_{M3} + \sin \theta_{M2}) \\
10 \sin \theta_{M1} * (\cos \theta_{M3} + \sin \theta_{M2}) \\
10 * (1 + \cos \theta_{M2} - \sin \theta_{M3})
\end{bmatrix}
\]

Implement task space control law with added friction compensation of the form

\[
\begin{bmatrix}
\tau_{M1} \\
\tau_{M2} \\
\tau_{M3}
\end{bmatrix} = \begin{bmatrix}
J^T \cdot \begin{bmatrix}
F_X \\
F_Y \\
F_Z
\end{bmatrix}
+ \begin{bmatrix}
\tau_{M1FricComp} \\
\tau_{M2FricComp} \\
\tau_{M3FricComp}
\end{bmatrix}
\end{bmatrix} = \begin{bmatrix}
K_P_X * (x^d - x) + K_D_X * (\dot{x}^d - \dot{x}) \\
K_P_Y * (y^d - y) + K_D_Y * (\dot{y}^d - \dot{y}) \\
K_P_Z * (z^d - z) + K_D_Z * (\dot{z}^d - \dot{z})
\end{bmatrix}
\]

Initially just command the robot to hold position at one \( x, y, z \) point. Note for this control law the \( K_p \) and \( K_d \) gains are much smaller than the gains we used in the last two lab assignments. Start out with a \( K_p \) gain around 0.5 and a \( K_d \) gain around 0.025. Also in your implementation remember to minimize the number of
repeated calls of \( \sin() \) and \( \cos() \) by using float variables to store the repeated trigonometry expressions and use them in your calculations.

**Part 3: Feedforward Force.**

3.1 Apply a Force in the Z Direction

Now that task space PD around one point is implemented, let’s look at a way to apply a force with the robot in the direction of its Z world direction. (You could also use this method to apply force in the X or Y direction, and after part 4 below, you could create a force in any direction.) What we do here will look very similar to your gravity compensation of Lab 2. First as a test, rerun your code from Part 2 and have the robot control itself at one point. Then while the robot is controlling at that point, use Code Composer’s “Watch Expressions” window to first zero your Kp for the Z direction. Your robot will probably fall in the Z direction. Now also zero the Kd for the Z direction. By hand, you should be able to move the robot easily in the Z direction but it resists you in the X and Y directions. If the robot kind of “bounces” in your hand as you are moving it up and down, your friction compensation may be a bit large. Create a “float” friction multiplication factor, in the below equation I call this “\( ff \)”, that you can adjust in Code Composer until the “bouncing” is not as noticeable. Your TA can help you find this correct factor but it also can be adjusted when you apply the force below. Now implement the following addition to your PD task space control to apply a negative Zcmd force (pressing down). Note the 0.0254 that is converting the inches of the Jacobian to meters. “\( Kt \)” is the robot’s torque constant which is approximately 6.0.

\[
\begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = J^T \begin{bmatrix} F_X \\ F_Y \\ F_Z \end{bmatrix} + \begin{bmatrix} \tau_{M1FricComp} \\ \tau_{M2FricComp} \\ \tau_{M3FricComp} \end{bmatrix} = J^T \begin{bmatrix} KP_x \times (x^d - x) + KD_x \times (\dot{x}^d - \dot{x}) \\ KP_y \times (y^d - y) + KD_y \times (\dot{y}^d - \dot{y}) \\ KP_z \times (z^d - z) + KD_z \times (\dot{z}^d - \dot{z}) \end{bmatrix} + ff \begin{bmatrix} \tau_{M1FricComp} \\ \tau_{M2FricComp} \\ \tau_{M3FricComp} \end{bmatrix} + 0.0254 \times J^T \begin{bmatrix} 0 \\ 0 \\ F_{Zcmd}/K_t \end{bmatrix}
\]

Try 5 different force values in the range of 0 to -20 and notice the difference as you hold the robot arm in the Z direction.

**Part 4: Play with Simple Impedance Control.**

4.1 Impedance Control

Now that task space PD around one point is implemented, try weakening one axis of control gains and while the controller is running manually move the arm in the three axis directions and notice that it is weak in the one axis and strong in the other two axes. Then play around with making two axes “weak” and one axis strong. Finally use a rotation matrix coordinate transformation to select another non world frame axis as the weak axis. 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 \( \theta_z \) about the z axis and then rotating \( \theta_x \) about the x axis and then \( \theta_y \) about the y axis. This gives the rotation matrix

\[
R_{zxy} = R_N^W = \begin{bmatrix} \cos \theta_z & -\sin \theta_z & 0 \\ \sin \theta_z & \cos \theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_x & -\sin \theta_x \\ 0 & \sin \theta_x & \cos \theta_x \end{bmatrix} \times \begin{bmatrix} \cos \theta_y & 0 & \sin \theta_y \\ 0 & 1 & 0 \\ -\sin \theta_y & 0 & \cos \theta_y \end{bmatrix},
\]
\[ R_{xyz} = R_N^W = \begin{bmatrix} \cos\theta_z\cos\theta_y - \sin\theta_z\sin\theta_x\sin\theta_y & -\sin\theta_z\cos\theta_y + \sin\theta_z\sin\theta_x\cos\theta_y & \cos\theta_z\sin\theta_x \\ \sin\theta_z\cos\theta_y + \cos\theta_z\sin\theta_x\sin\theta_y & \cos\theta_z\cos\theta_x & \sin\theta_z\sin\theta_y - \cos\theta_z\sin\theta_x\cos\theta_y \\ -\cos\theta_x\sin\theta_y & \sin\theta_x & \cos\theta_x\cos\theta_y \end{bmatrix} \]

Given this rotation matrix, we can perform a coordinate transformation of \( F_x, F_y, F_z \) in the N frame to \( F_x, F_y, F_z \) in the World frame.

\[
\begin{bmatrix} F_{xW} \\ F_{yW} \\ F_{zW} \end{bmatrix} = R_N^W \begin{bmatrix} F_{xN} \\ F_{yN} \\ F_{zN} \end{bmatrix}
\]

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

\[
\begin{bmatrix} \tau_{M1} \\ \tau_{M2} \\ \tau_{M3} \end{bmatrix} = J^T \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} + R_N^W \begin{bmatrix} KP_{xN} * (x_N^d - x_N) + KD_{xN} * (\dot{x}_N^d - \dot{x}_N) \\ KP_{yN} * (y_N^d - y_N) + KD_{yN} * (\dot{y}_N^d - \dot{y}_N) \\ KP_{zN} * (z_N^d - z_N) + KD_{zN} * (\dot{z}_N^d - \dot{z}_N) \end{bmatrix} + \begin{bmatrix} \tau_{M1FricComp} \\ \tau_{M2FricComp} \\ \tau_{M3FricComp} \end{bmatrix}
\]

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, \( R_N^W \). Properties of the rotation matrix prove that \( R_N^W \) is simply the transpose of \( R_N^W \). \( R_N^W = R_N^{WT} \).

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 KP and KD gains. The controller equations then become

\[
\begin{bmatrix} \tau_{M1} \\ \tau_{M2} \\ \tau_{M3} \end{bmatrix} = J^T R_N^W \begin{bmatrix} KP_{xN} & 0 & 0 \\ 0 & KP_{yN} & 0 \\ 0 & 0 & KP_{zN} \end{bmatrix} \begin{bmatrix} x_W^d - x_W \\ y_W^d - y_W \\ z_W^d - z_W \end{bmatrix} + R_N^W \begin{bmatrix} 0 & KP_{xN} & 0 \\ KP_{yN} & 0 & 0 \\ 0 & 0 & KP_{zN} \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_N^d - x_N \\ y_N^d - y_N \\ z_N^d - z_N \end{bmatrix} \]

Part 5: Straight Line Following.

5.1 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.

\[
\text{Straight Line from } (x_a, y_a, z_a) \text{ to } (x_b, y_b, z_b)
\]

\[
\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} x_b - x_a \\ y_b - y_a \\ z_b - z_a \end{bmatrix}, \text{ vector direction to travel}
\]
\[ t_{\text{total}} = \frac{\text{distance between } a \text{ and } b}{\text{desired speed along the line}} \]

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

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 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 \((x^d_N - x_N), (y^d_N - y_N), (z^d_N - z_N)\), are values in the N frame. So 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

\[
\begin{bmatrix}
  \tau_{M1} \\
  \tau_{M2} \\
  \tau_{M3}
\end{bmatrix} = J^T \cdot R^W_N \cdot \begin{bmatrix}
  KP_{x_N} & 0 & 0 \\
  0 & KP_{y_N} & 0 \\
  0 & 0 & KP_{z_N}
\end{bmatrix} \cdot R^N_W \cdot \begin{bmatrix}
  (x^d_W - x_W) \\
  (y^d_W - y_W) \\
  (z^d_W - z_W)
\end{bmatrix} + \begin{bmatrix}
  KD_{x_N} & 0 & 0 \\
  0 & KD_{y_N} & 0 \\
  0 & 0 & KD_{z_N}
\end{bmatrix} \cdot R^N_W \cdot \begin{bmatrix}
  (\dot{x}^d_W - \dot{x}_W) \\
  (\dot{y}^d_W - \dot{y}_W) \\
  (\dot{z}^d_W - \dot{z}_W)
\end{bmatrix} + \begin{bmatrix}
  \tau_{M1FricComp} \\
  \tau_{M2FricComp} \\
  \tau_{M3FricComp}
\end{bmatrix}.
\]

**Assignment:**

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.

2. Again follow a straight line but also make the direction perpendicular to the line weak and the direction along the line stiff. To make things a bit easier here, keep the line in a plane parallel to the World XY plane, so one rotation.

**Report:**

The report for Lab 3 is just the C code you developed, but well commented code. 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.

Controllers to demonstrate to your TA:

1. Task space PD controller controlling at one point in space.

2. Impedance Control
   a. Weak in one World coordinate axis
b. Weak in two World coordinate axis

c. Weak in one non-World coordinate axis. (At least 2 rotations)

3. Following a straight line from one point to a second point.

a. First with all axes stiff and making the N frame the World Frame, so no rotation.

b. Second make the direction perpendicular to the line weak and the direction along the line stiff. To make things a bit easier here, keep the line in a plane parallel to the World XY plane.

S. T. E. R. R. A. C. C. E

```c
float cosq1 = 0;
float sinq1 = 0;
float cosq2 = 0;
float sinq2 = 0;
float cosq3 = 0;
float sinq3 = 0;

float JT_11 = 0;
float JT_12 = 0;
float JT_13 = 0;
float JT_21 = 0;
float JT_22 = 0;
float JT_23 = 0;
float JT_31 = 0;
float JT_32 = 0;
float JT_33 = 0;

float cosz = 0;
float sinz = 0;
float cosx = 0;
float sinx = 0;
float cosy = 0;
float siny = 0;

float thetaz = 0;
float thetax = 0;
float thetay = 0;

float R11 = 0;
float R12 = 0;
float R13 = 0;
float R21 = 0;
float R22 = 0;
float R23 = 0;
float R31 = 0;
float R32 = 0;
float R33 = 0;
```
float RT11 = 0;
float RT12 = 0;
float RT13 = 0;
float RT21 = 0;
float RT22 = 0;
float RT23 = 0;
float RT31 = 0;
float RT32 = 0;
float RT33 = 0;

// Rotation zxy and its Transpose
  cosz = cos(thetaz);
sinz = sin(thetaz);
cosx = cos(thetax);
sinx = sin(thetax);
cosy = cos(thetay);
siny = sin(thetay);

  RT11 = R11 = cosz*cosy-sinz*sinx*siny;
  RT21 = R12 = -sinz*cosx;
  RT31 = R13 = cosz*siny+sinz*sinx*cosy;
  RT12 = R21 = sinz*cosy+cosz*sinx*siny;
  RT22 = R22 = cosz*cosx;
  RT32 = R23 = sinz*siny-cosz*sinx*cosy;
  RT13 = R31 = -cosx*siny;
  RT23 = R32 = sinx;
  RT33 = R33 = cosx*cosy;

// Jacobian Transpose
  cosq1 = cos(theta1motor);
sinq1 = sin(theta1motor);
cosq2 = cos(theta2motor);
sinq2 = sin(theta2motor);
cosq3 = cos(theta3motor);
sinq3 = sin(theta3motor);

  JT_11 = -10*sinq1*(cosq3 + sinq2);
  JT_12 = 10*cosq1*(cosq3 + sinq2);
  JT_13 = 0;
  JT_21 = 10*cosq1*(cosq2 - sinq3);
  JT_22 = 10*sinq1*(cosq2 - sinq3);
  JT_23 = -10*(cosq3 + sinq2);
JT_{31} = -10\cos q_1 \sin q_3;
JT_{32} = -10\sin q_1 \sin q_3;
JT_{33} = -10\cos q_3;