

ME 446 Laboratory #4

Task Space PD control, Impedance Control

Report is due the week of April 24th. One report per group. Lab sessions will be held the weeks of April 10th and April 17th.

Objectives

- Implement Task Space PD Control.
- Weaken Axis gains to achieve a Simple Impedance Control.
- Straight Line Following.

Part 1: Implement Task Space PD Control.

1.1 Task Space PD Control.

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

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

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

$$\begin{bmatrix} \tau_{M1} \\ \tau_{M2} \\ \tau_{M3} \end{bmatrix} = \begin{bmatrix} F_X \\ F_Y \\ F_Z \end{bmatrix} + \begin{bmatrix} \tau_{M1FricComp} \\ \tau_{M2FricComp} \\ \tau_{M3FricComp} \end{bmatrix} = \begin{bmatrix} KP_X * (x^d - x) + KD_X * (\dot{x}^d - \dot{x}) \\ KP_Y * (y^d - y) + KD_Y * (\dot{y}^d - \dot{y}) \\ KP_Z * (z^d - z) + KD_Z * (\dot{z}^d - \dot{z}) \end{bmatrix} + \begin{bmatrix} \tau_{M1FricComp} \\ \tau_{M2FricComp} \\ \tau_{M3FricComp} \end{bmatrix}$$

Initially just command the robot to hold position at one x,y,z point. Note for this control law the Kp and Kd gains are much smaller than the gains we used in the last two lab assignments. Start out with a Kp gain around 0.5 and a Kd 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 2: Play with Simple Impedance Control.

2.1 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 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 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

$$\begin{bmatrix} \tau_{M1} \\ \tau_{M2} \\ \tau_{M3} \end{bmatrix} = \begin{bmatrix} J^T * \begin{bmatrix} F_X \\ F_Y \\ F_Z \end{bmatrix} \end{bmatrix} + \begin{bmatrix} \tau_{M1FricComp} \\ \tau_{M2FricComp} \\ \tau_{M3FricComp} \end{bmatrix} = \begin{bmatrix} J^T * R_N^W \begin{bmatrix} KP_{X_N} * (x_N^d - x_N) + KD_{X_N} * (\dot{x}_N^d - \dot{x}_N) \\ KP_{Y_N} * (y_N^d - y_N) + KD_{Y_N} * (\dot{y}_N^d - \dot{y}_N) \\ KP_{Z_N} * (z_N^d - z_N) + KD_{Z_N} * (\dot{z}_N^d - \dot{z}_N) \end{bmatrix} \end{bmatrix} + \begin{bmatrix} \tau_{M1FricComp} \\ \tau_{M2FricComp} \\ \tau_{M3FricComp} \end{bmatrix}$$

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_W^N .

Properties of the rotation matrix prove that R_W^N is simply the transpose of R_N^W . $R_W^N = R_N^{WT}$.

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

$$\begin{bmatrix} \tau_{M1} \\ \tau_{M2} \\ \tau_{M3} \end{bmatrix} = \begin{bmatrix} J^T * R_N^W * \begin{bmatrix} KP_{X_N} & 0 & 0 \\ 0 & KP_{Y_N} & 0 \\ 0 & 0 & KP_{Z_N} \end{bmatrix} * R_W^N * \begin{bmatrix} (x_W^d - x_W) \\ (y_W^d - y_W) \\ (z_W^d - z_W) \end{bmatrix} + \begin{bmatrix} KD_{X_N} & 0 & 0 \\ 0 & KD_{Y_N} & 0 \\ 0 & 0 & KD_{Z_N} \end{bmatrix} * R_W^N * \begin{bmatrix} (\dot{x}_W^d - \dot{x}_W) \\ (\dot{y}_W^d - \dot{y}_W) \\ (\dot{z}_W^d - \dot{z}_W) \end{bmatrix} \end{bmatrix} + \begin{bmatrix} \tau_{M1FricComp} \\ \tau_{M2FricComp} \\ \tau_{M3FricComp} \end{bmatrix}$$

Part 3: Straight Line Following.

3.1 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.

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

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

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

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

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 . $R_W^N = R_N^{WT}$.

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 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

$$\begin{bmatrix} \tau_{M1} \\ \tau_{M2} \\ \tau_{M3} \end{bmatrix} = \left[J^T * R_N^W * \begin{bmatrix} KP_{X_N} & 0 & 0 \\ 0 & KP_{Y_N} & 0 \\ 0 & 0 & KP_{Z_N} \end{bmatrix} * R_W^N * \begin{bmatrix} (x_W^d - x_W) \\ (y_W^d - y_W) \\ (z_W^d - z_W) \end{bmatrix} + \begin{bmatrix} KD_{X_N} & 0 & 0 \\ 0 & KD_{Y_N} & 0 \\ 0 & 0 & KD_{Z_N} \end{bmatrix} * R_W^N * \begin{bmatrix} (\dot{x}_W^d - \dot{x}_W) \\ (\dot{y}_W^d - \dot{y}_W) \\ (\dot{z}_W^d - \dot{z}_W) \end{bmatrix} \right] + \begin{bmatrix} \tau_{M1FricComp} \\ \tau_{M2FricComp} \\ \tau_{M3FricComp} \end{bmatrix}$$

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 4 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.

Starter C Code

```
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(thetamotor1);
sinq1 = sin(thetamotor1);
cosq2 = cos(thetamotor2);
sinq2 = sin(thetamotor2);
cosq3 = cos(thetamotor3);
sinq3 = sin(thetamotor3);
```

```
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*cosq1*sinq3;
JT_32 = -10*sinq1*sinq3;
JT_33 = -10*cosq3;
```