Chapter 5 HW Hints

5.1. This problem refers to Example 5.4, not 5.3. Basically it’s asking whether the singularities of $^0J$ are the same as those of $^3J$. Think about whether the existence of singularities is dependent on the coordinate frames you’re using.

5.3. This problem requires a complete kinematic analysis: placing frames, getting $T$ matrices, etc. To help you out a little, the necessary $T$ matrices are listed below:

$$
^0T = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad ^1T = \begin{bmatrix} c_2 & -s_2 & 0 & L_1 \\ s_2 & c_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad ^3T = \begin{bmatrix} c_3 & -s_3 & 0 & L_2 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The final transform, $^3T$, should be very easy to obtain. You should consider this robot as a “tip positioner.” The position of the tip relative to base frame $\{0\}$ is simply $[x \ y \ z]$. We will not need to consider the orientation of frame $\{4\}$.

A partial result for the Jacobian in $\{4\}$ (velocity & force propagation) is

$$^4J = \begin{bmatrix} 0 & L_2s_3 & 0 \\ * & L_2c_3 + L_3 & * \\ * & 0 & 0 \end{bmatrix}$$

A partial result for the Jacobian in $\{0\}$ (differentiation method) is

$$^0J = \begin{bmatrix} -(L_1 + L_2c_2 + L_3c_2) s_1 & * & -L_3c_1s_23 \\ * & -(L_2s_2 + L_3s_23)s_1 & * \\ 0 & * & L_3c_23 \end{bmatrix}$$

One can get this Jacobian expressed in frame $\{4\}$ by doing

$$^4J = ^3R^0J$$

and the result should agree with the previous $^4J$, but it’s more work than I felt like doing.

NOTE 1: When you are doing force propagation you should set the Cartesian force and moment in frame $\{4\}$ to be

$$^4F = [f_x \ f_y \ f_z]^T, \quad ^4N = [0 \ 0 \ 0]^T$$

This Cartesian force and moment is consistent with this mechanism’s use as a “tip positioner.”

NOTE 2: You need only find the Z-component of $^1n_1$, since that is where you obtain the joint torque $\tau_1$.

NOTE 3: I think with the “differentiation” method you should leave the Jacobian in frame $\{0\}$; it is too much work to transform it into frame $\{4\}$.

5.4. Consider whether a matrix has the same singularities as its transpose...

5.8. Isotropic points are almost the opposite of singularities—they are points where the Jacobian matrix is “best-conditioned.” Requiring the columns of the Jacobian to be orthogonal and of equal magnitude causes each joint to cause motion of equal amount and orthogonal to that caused by the other joints. We are making “best use” of the available degrees of freedom.

There is a little bit of a trick you can use here. The Jacobian for this 2-link arm given in (5.66) can be expressed

$$^3J = \begin{bmatrix} l_1s_2 & 0 \\ l_1c_2 + l_2 & l_2 \end{bmatrix} = [V_1 \ V_2]$$
and since column vector $V_2$ is constant, we need column vector $V_1$ to have the same magnitude as $V_2$, but arranged “oppositely”, so we want

$$V_1 = \begin{bmatrix} l_1 s_2 \\ l_1 c_2 + l_2 \end{bmatrix} = \begin{bmatrix} l_2 \\ 0 \end{bmatrix}$$

Be careful when solving these equations; you need to get the required relationship between $l_1$ and $l_2$ and the correct value(s) for $\theta_2$. First solve for the relationship between $l_1$ and $l_2$, which is $l_1/l_2 = \pm \sqrt{2}$; only the “+” sign gives a feasible relationship, but the “±” is necessary to find the angle(s). Then solve for $\theta_2$. It’s a little tricky; good luck.

Partial answer: $l_1 = \sqrt{2} l_2$, $\theta_2 =$?

Problem 5.13. Recall that $\tau = J^T F$. Since both $^0J$ and $^0F$ are expressed in frame $\{0\}$, and since $^0F = 10X_0$, or in matrix form, $^0F = \begin{bmatrix} 10 & 0 & 0 \end{bmatrix}$, this problem should be pretty straightforward.

MATLAB Exercise 5. Note that this is not “Programming Exercise 5”—this time we are doing MATLAB Exercise 5. You should do this complete exercise and make all the plots listed in part (a) of the problem. You can ignore part (b). In this exercise you will be using the inverse Jacobian to achieve Cartesian velocity control of a 3-link RRR manipulator. This is the same manipulator used in the Programming Exercises but it has different link lengths (4, 3, and 2 meters, respectively). This Jacobian inverse control method was first proposed by Dan Whitney in 1972, and is called “Resolved Motion Rate Control” (RMRC). I used it in my Ph.D. work in the mid-70s. Unfortunately MATLAB didn’t exist then (computers hardly did—we did everything on an old IBM 1800). If you Google “IBM 1800” you get the idea. It did have lots of pretty front-panel lights. Anyway...

You should write two M-files (one function and one script) in the completion of this exercise; to find the Cartesian tip pose you will also need the forward kinematics, but this is the same as for the Programming Exercises (with different link lengths). So you can use your kin.m from before. The new function and script are:

- Function jacob.m which computes the Jacobian matrix in frame $\{0\}$ (this Jacobian is given in the problem statement in the text).
- Script rmrc.m which contains the loop to calculate the motion of the manipulator over the required trajectory. I changed the time step from 0.1 second to 0.05 seconds so my simulation would be easier to see; you can use a smaller time step yet if you wish (it won’t change the plots).

1. My description of function jacob.m is as follows:

```matlab
% function J = jacob(theta) is a function which computes the 3x3 Jacobian
% matrix in frame $\{0\}$ for Craig MATLAB Exercise 5. This is based on the
% 3R planar manipulator but with link lengths of 4, 3, and 2 meters,
% respectively. 3x1 vector theta contains the three joint angles in
% radians.

function J = jacob(theta)

... % The function code goes here

J = [ ... ];  % Expression to compute Jacobian
```

The expression for the Jacobian in frame $\{0\}$ is given within the MATLAB Exercise 5 problem statement on text page 163.
2. My description of script \texttt{rmrc.m} is as follows:

% Script \texttt{rmrc.m} computes the required joint rates (thetadots) to execute
% the desired Cartesian path specified in MATLAB Exercise 5. It uses
% function \texttt{jacob.m} to find the Jacobian in frame \{0\}. The main portion of
% this script is a loop that runs over the 5 seconds of the trajectory
% using a stepsize of 0.05 second.
%
% In accordance with part (a) of the exercise, the main loop calculates and
% saves (1) [thetadots], (2) [thetas], (3) [x y phi], (4) |J|, and (5)
% [torques] at each time step for subsequent plotting. A very simple
% integration method (newtheta = oldtheta + thetadot*dt) is adequate.
%
% The saved [thetas} can be used with a line-drawing simulation provided by
% Starr to check your motion path.

l1 = 4;  \ % Link 1 length (m)
l2 = 3;  \ % Link 2 length (m)
l3 = 2;  \ % Link 3 length (m)


... \ % Remainder of the code for the script file goes here

Pseudo-code description of \texttt{rmrc.m}:

* Initialize all constants: link lengths l, motion stepsize \( dt \) (use 0.05 or smaller)
and duration \( tf \), Cartesian velocity \( V \) and Cartesian wrench \( W \).

* Given the motion duration and stepsize, compute the required number of samples \( N \).

* Given \( N \), allocate data arrays for all quantities that will be plotted after the
\texttt{rmrc.m} has been executed: \texttt{theta}, \texttt{thetad}, \texttt{cart ([x y phi])}, \texttt{Jdet}, and \texttt{Torque}.
The method I used was: \texttt{theta = zeros(N,3)}; similar for other variables.

* Set joint angles \texttt{theta(1,:)} to their initial values (RAD).

* begin main loop (for \( i = 1:N \))
  * Get Cartesian position \([x y phi]\) using forward kinematics at current joint angles
  * Compute Jacobian \( J \) at current joint angles
  * Evaluate determinant of \( J \) and store in array \texttt{Jdet}
  * Find joint rates "thetad" using inverse Jacobian \texttt{inv(J)}: \texttt{thetad = inv(J)*V}
  * Find static joint torques: \texttt{Torque = J'*W}, where \( J' \) is \( J \) transpose and \( W \) is wrench
  * Find new thetas using: \texttt{theta(i+1,:) = theta(i,:)} + \texttt{thetad(i,:)*dt}
* end main loop

* I created a time vector \( t = [0:dt:tf]' \) for subsequent offline plotting of variables.

\textit{NOTE}: The explicit inversion of Jacobian \( J \) (fourth item in the loop above) is now discouraged by The Mathworks,
since it’s inefficient. The “efficient” way to find the joint rates from the Cartesian rates is:

\( \texttt{thetad = J\backslash V}; \ % \) The backslash operator is preferable to explicit inversion using \texttt{inv()}
Results. Shown below is the plot for the joint angles.

Animated Simulation. There will be a function titled \texttt{simr.m} on my website which will allow you do perform an animated simulation of the motion. The format of the function is: \texttt{simr(theta)}, where \texttt{theta} is an $N \times 3$ array of joint angles in radians.

\begin{verbatim}

function simr(theta)
    % This function simulates the motion of the 3R manipulator from the Chapter 5 MATLAB Exercises.
    % Parameter "theta" should be an Nx3 array of joint angles in radians. Number "N" is the total number of samples in the run: (5/dt)+1.
end
\end{verbatim}