ME 449 Homework 3

1 Problem 1

The distance from the origin to joint 2 is defined by the equations

$$x = L_1 \cos\theta_1$$
$$y = L_1 \sin\theta_1.$$

The distance from joint 2 to the end-effector is given by

$$x = L_2 \cos(\theta_1 + \theta_2)$$
$$y = L_2 \sin(\theta_1 + \theta_2).$$

So the equations to get from the origin to the (x, y) position of the end-effector are given by

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) \\ L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2) \end{bmatrix}.$$

The Jacobian is defined as

$$J(\theta) = \frac{\partial f}{\partial \theta} = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_1}{\partial \theta_2} \\ \frac{\partial f_2}{\partial \theta_1} & \frac{\partial f_2}{\partial \theta_2} \end{bmatrix}$$

where $f(\theta)$ is

$$f(\theta) = \begin{bmatrix} L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2) \\ L_1 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2) \end{bmatrix}.$$

and the elements in the matrix evaluate to

$$\frac{\partial f_1}{\partial \theta_1} = -L_1\dot{\theta_1}\sin\theta_1 - L_2\dot{\theta_1}\sin(\theta_1+\theta_2)$$
$$\frac{\partial f_1}{\partial \theta_2} = -L_2\dot{\theta_2}\sin(\theta_1+\theta_2)$$
$$\frac{\partial f_2}{\partial \theta_1} = L_1\dot{\theta_1}\cos\theta_1 + L_2\dot{\theta_1}\cos(\theta_1+\theta_2)$$
$$\frac{\partial f_2}{\partial \theta_2} = L_2\dot{\theta_2}\cos(\theta_1+\theta_2).$$

The $\dot{\theta_1}$ and $\dot{\theta_2}$ terms get extracted into a column vector that gets multiplied against the Jacobian. This yields a Jacobian matrix of the following form

$$\boxed{J(\theta) = \begin{bmatrix} -L_1\sin\theta_1 - L_2\sin(\theta_1+\theta_2) & -L_2\sin(\theta_1+\theta_2) \\ L_1\cos\theta_1 + L_2\cos(\theta_1+\theta_2) & L_2\cos(\theta_1+\theta_2) \end{bmatrix}}$$

Another way to calculate the Jacobian is to take the original equations in matrix form and take the time derivative of the x and y parts. The result is

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -L_1\dot{\theta_1}\sin\theta_1 - L_2\dot{\theta_1}\sin(\theta_1+\theta_2) - L_2\dot{\theta_2}\sin(\theta_1+\theta_2) \\ L_1\dot{\theta_1}\cos\theta_1 + L_2\dot{\theta_1}\cos(\theta_1+\theta_2) + L_2\dot{\theta_2}\cos(\theta_1+\theta_2) \end{bmatrix}$$
$$= \begin{bmatrix} -L_1\sin\theta_1 - L_2\sin(\theta_1+\theta_2) & -L_2\sin(\theta_1+\theta_2) \\ L_1\cos\theta_1 + L_2\cos(\theta_1+\theta_2) & L_2\cos(\theta_1+\theta_2) \end{bmatrix} \begin{bmatrix} \dot{\theta_1} \\ \dot{\theta_2} \end{bmatrix}$$

This yields the same Jacobian matrix

$$\boxed{J(\theta) = \begin{bmatrix} -L_1\sin\theta_1 - L_2\sin(\theta_1+\theta_2) & -L_2\sin(\theta_1+\theta_2) \\ L_1\cos\theta_1 + L_2\cos(\theta_1+\theta_2) & L_2\cos(\theta_1+\theta_2) \end{bmatrix}}$$

2 Problem 2

In my Mathematica code I created a function called `CalculateSpaceFrameJacobian` that takes in a list of screw axes and a list of θ values associated with each screw axis. These screw axes and θ values are used to construct the Jacobian using the following form

$$J_s(\theta) = \left[\; \mathcal{S}_1 \;\middle|\; Ad_{e^{[\mathcal{S}_1]\theta_1}}(\mathcal{S}_2) \;\middle|\; Ad_{e^{[\mathcal{S}_1]\theta_1}e^{[\mathcal{S}_2]\theta_2}}(\mathcal{S}_3) \;\middle|\; ... \;\middle|\; Ad_{e^{[\mathcal{S}_1]\theta_1}e^{[\mathcal{S}_2]\theta_2}...e^{[\mathcal{S}_{n-1}]\theta_{n-1}}}(\mathcal{S}_n) \; \right]$$

The code operates in a loop and appends new columns to the Jacobian matrix as it goes. It begins by adding the $\mathcal{S}_1$ screw axis 6-vector as the first column of the Jacobian. It then calculates the transformation matrix $T_1 = e^{[\mathcal{S}_1]\theta_1}$ and

generates its adjoint matrix. The code now enters a loop where it first calculates the new column of the Jacobian by multiplying the adjoint matrix by the second screw axis

$$\mathcal{V}_{s2}\left(\theta\right) = Ad_{e^{[\mathcal{S}_1]\theta_1}}\left(\mathcal{S}_2\right)$$

and adds this new column to the Jacobian matrix. Then it calculates the transformation matrix for the second screw axis using $T_2 = e^{[\mathcal{S}_2]\theta_2}$ and multiplies it against T_1 so that

$$T_1 T_2 = e^{[\mathcal{S}_1]\theta_1} e^{[\mathcal{S}_2]\theta_2}$$

The loop starts over at the beginning where it calculates the adjoint matrix and generates the next column. The algorithm finishes when it has iterated over all of the screw axes and generated a full Jacobian matrix.

The body Jacobian is generated in a similar way to the space Jacobian, except the algorithm starts from the body frame and works its way towards the space frame. The makeup of the Jacobian columns ends up being

$$J_b(\theta) = \left[\begin{array}{c|c|c|c|c} Ad_{e^{-[\mathcal{B}_n]\theta_n}\ldots e^{-[\mathcal{B}_2]\theta_2}}\left(\mathcal{B}_1\right) & \ldots & Ad_{e^{-[\mathcal{B}_n]\theta_n} e^{-[\mathcal{B}_{n-1}]\theta_{n-1}}}\left(\mathcal{B}_{n-2}\right) & Ad_{e^{-[\mathcal{B}_n]\theta_n}}\left(\mathcal{B}_{n-1}\right) & \mathcal{B}_n \end{array}\right]$$

The body Jacobian is calculated in the function `CalculateBodyFrameJacobian` which takes in a list of body-frame screw axes and a list of θ values associated with each screw axis. The code is similar to the calculation of the space-frame Jacobian, except that it operates in reverse order when adding columns to the matrix. It begins by initializing a matrix with a single column that corresponds to the $\mathcal{B}_n$ screw axis. It then calculates a transformation matrix from the screw axis $\mathcal{B}_n$ by using the equation

$$T_n = e^{-[\mathcal{B}_n]\theta_n}.$$

This transformation matrix is used to build up the successive adjoint matrices as the algorithm iterates over the screw axes. Once this matrix has been initialized, the main loop begins by calculating the adjoint matrix of the transformation matrix. It then multiplies this adjoint matrix with the next body frame screw axis, $\mathcal{B}_{n-1}$, to get the next column of the Jacobian

$$\mathcal{V}_{b,n-1}\left(\theta\right) = Ad_{e^{[\mathcal{B}_n]\theta_n}}\left(\mathcal{B}_{n-1}\right).$$

This column is then prepended to the growing matrix. The final step of the loop is to prepare for the next iteration by calculating the new transformation matrix to be used for the adjoint in the next iteration using the formula

$$T_n T_{n-1} = e^{-[\mathcal{B}_n]\theta_n} e^{-[\mathcal{B}_{n-1}]\theta_{n-1}}$$

The loop starts over at the beginning where it calculates the adjoint matrix and continues looping until it has iterated over all of the screw axes and generated a full Jacobian matrix.

The world-frame screw axes are

$$\mathcal{S}_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathcal{S}_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ -L_1 \\ 0 \end{bmatrix}.$$

When the robot is in the home position and is rotated about joint 1, the space frame simply rotates about the z-axis with no translational movement, yielding the values for $\mathcal{S}_1$. And when the robot is rotated about joint 2 the world frame rotates about the z-axis and moves $-L_1$in the $-y$-axis direction, giving the values for $\mathcal{S}_2$.

2.1 Part a

Using the space-frame and body-frame screw axes, $(\theta_1, \theta_2) = (0, \pi/2)$ in the previously-described functions produces a space-frame Jacobian with with members

$$J_s(\theta) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 0 & -2 \\ 0 & 0 \end{bmatrix},$$

and the body-frame screw axes produce a Jacobian with members

$$J_b(\theta) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \\ 2 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}.$$

2.2 Part b

For θ values, $(\theta_1, \theta_2) = (0, \pi/2)$, the space-frame Jacobian is

$$J_s(\theta) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \\ 0 & \sqrt{2} \\ 0 & -\sqrt{2} \\ 0 & 0 \end{bmatrix},$$

and the body-frame Jacobian is

$$J_b(\theta) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \\ \sqrt{2} & 0 \\ 1+\sqrt{2} & 1 \\ 0 & 0 \end{bmatrix}.$$

3 Problem 3

The plots for the three joint positions are based on the end-effector velocities computed using the Jacobian from problem 1

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -L_1 \sin\theta_1 - L_2 \sin(\theta_1+\theta_2) & -L_2 \sin(\theta_1+\theta_2) \\ L_1 \cos\theta_1 + L_2 \cos(\theta_1+\theta_2) & L_2 \cos(\theta_1+\theta_2) \end{bmatrix} \begin{bmatrix} \dot{\theta_1} \\ \dot{\theta_2} \end{bmatrix}$$

where $L_1 = 2$ and $L_2 = 1$. The first set of joint angles $(-\pi/8, \pi/4)$ produces a Jacobian with the elements

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \sin(\pi/8) & -\sin(\pi/8) \\ 3\cos(\pi/8) & \cos(\pi/8) \end{bmatrix} = \begin{bmatrix} 0.382683 & -0.382683 \\ 2.77164 & 0.92388 \end{bmatrix}.$$

This Jacobian is multiplied against the joint angle velocities to generate a polygon that represents all of the possible end-effector velocities from the robot configuration described by the given joint angles. The extents of the polygon are determined by the maximum joint velocities, which in this case are $\dot{\theta}_i = -1$ and $\dot{\theta}_i = 1$. The equations and results are

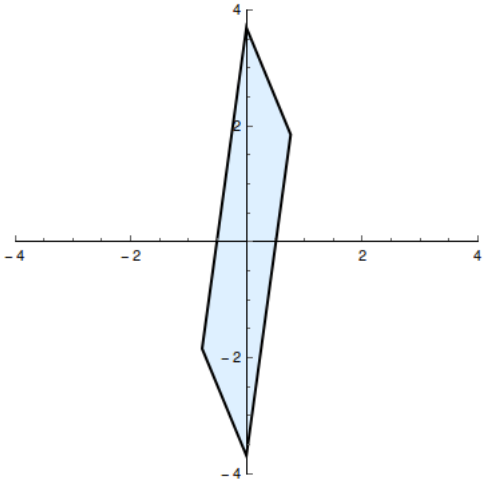
$$\begin{bmatrix} 0 \\ -3.69552 \end{bmatrix} = \begin{bmatrix} 0.382683 & -0.382683 \\ 2.77164 & 0.92388 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} -0.765367 \\ -1.84776 \end{bmatrix} = \begin{bmatrix} 0.382683 & -0.382683 \\ 2.77164 & 0.92388 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 3.69552 \end{bmatrix} = \begin{bmatrix} 0.382683 & -0.382683 \\ 2.77164 & 0.92388 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 0.765367 \\ 1.84776 \end{bmatrix} = \begin{bmatrix} 0.382683 & -0.382683 \\ 2.77164 & 0.92388 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

When these results are used as coordinates for the corners of the polygon plotted in end-effector velocity space, the inside and boundaries of the polygon represent all possible velocities for the current joint configuration. The following figure shows the plot.



The second set of joint angles $(-\pi/4, \pi/2)$ yields the following Jacobian

$$J(\theta) = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{3}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} 0.707107 & -0.707107 \\ 2.12132 & 0.707107 \end{bmatrix}.$$

Using the same joint velocities yields the following values

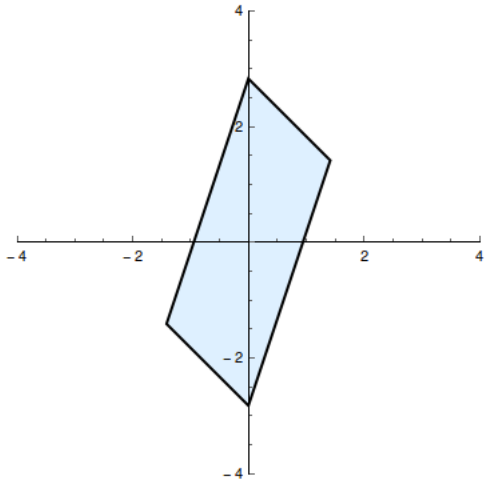
$$\begin{bmatrix} 0 \\ -2.82843 \end{bmatrix} = \begin{bmatrix} 0.707107 & -0.707107 \\ 2.12132 & 0.707107 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} -1.41421 \\ -1.41421 \end{bmatrix} = \begin{bmatrix} 0.707107 & -0.707107 \\ 2.12132 & 0.707107 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 2.82843 \end{bmatrix} = \begin{bmatrix} 0.707107 & -0.707107 \\ 2.12132 & 0.707107 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1.41421 \\ 1.41421 \end{bmatrix} = \begin{bmatrix} 0.707107 & -0.707107 \\ 2.12132 & 0.707107 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

The resulting plot is



The third set of joint angles $(-\pi/3, 2\pi/3)$ yields the following Jacobian

$$J(\theta) = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{3}{2} & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} 0.866025 & -0.866025 \\ 1.5 & 0.5 \end{bmatrix}.$$

Multiplying the Jacobian with the same joint velocities yields

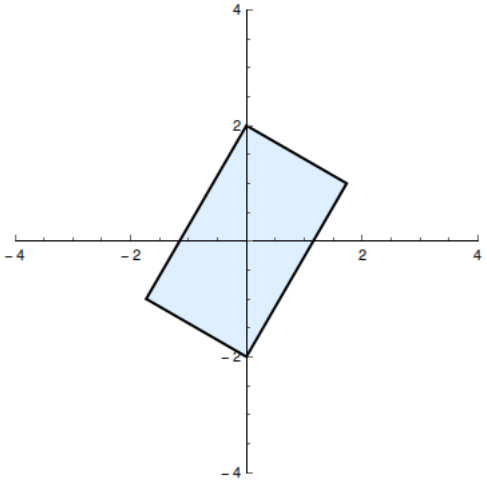
$$\begin{bmatrix} 0 \\ -2 \end{bmatrix} = \begin{bmatrix} 0.866025 & -0.866025 \\ 1.5 & 0.5 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} -1.73205 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.866025 & -0.866025 \\ 1.5 & 0.5 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 0.866025 & -0.866025 \\ 1.5 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1.73205 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.866025 & -0.866025 \\ 1.5 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

The resulting polygon is plotted below.



As can be seen from the plots the polygon gets closer and closer to looking like a square as it goes from the first set to the third set. This indicates that the robot gets farther and farther away from a singularity. If the polygon continued becoming thinner and thinner it would eventually become a line along the $\dot{y}$-axis, indicating that no linear velocities can be generated in the x direction. At this point the Jacobian loses rank and a singularity point has been reached.

4 Problem 4

As explained in section 6.3 of *Introduction to Robotics*, numerical inverse kinematics ultimately comes down to an iterative root-finding algorithm. The method used in this code is based on the Newton-Raphson method. The basic form of this method works by starting with an initial guess and using the slope of the curve it is following to find a new guess that inches closer to the true solution. It seeks to solve the equation

$$x_d - f(\theta_d) = 0$$

by iteratively solving the equation

$$x_d = f(\theta_d) \approx f(\theta_0) + \frac{\partial f}{\partial x}\Big|_{\theta_0} (\theta_d - \theta_0),$$

where θ_0 corresponds to an initial joint angle guess. In the second equation the slope term corresponds with the Jacobian and then difference term in parentheses corresponds with $\Delta\theta$. The $\Delta\theta$ corresponds to how much to change the latest θ_i guess for the next iteration. This value can be found by inverting the Jacobian to get the equation

$$\Delta\theta = \left(J\left(\theta_0\right)\right)^{-1}\left(x_d - f\left(\theta_0\right)\right).$$

If the Jacobian is not a square matrix, as is the case for the robot in problem 4, the Moore-Penrose pseudo-inverse must be used,

$$\Delta\theta = \left(J\left(\theta_0\right)\right)^{\dagger}\left(x_d - f\left(\theta_0\right)\right).$$

With these equations the algorithm proceeds by checking to see if the quantity $\|x_d - f\left(\theta_i\right)\|$ is less than some error threshold, ϵ. If it is not the algorithm then computes

$$\theta_{i+1} = \theta_i + \left(J\left(\theta_0\right)\right)^{\dagger}\left(x_d - f\left(\theta_0\right)\right)$$

and uses the new θ_{i+1} value as the new guess for the next iteration. The loop goes back up to the error check and determines whether the end-effector (x, y) pose is acceptably close to the desired position. If not, the loop continues to find new θ values using the algorithm. If it is close enough, the algorithm terminates and returns the most recently computed θ_{i+1} value.

The code for calculating the inverse kinematics in the body-frame is implemented in `CalculateBodyFrameInverseKinematics`. This function takes in the desired end-effector pose relative to the space frame, the list of body frame screw axes that define the movement of the end-effector from its home position, an initial guess for joint positions that could produce the desired end-effector pose, the robot's home matrix, and an error threshold value that is used to evaluate when the joint angle solution is "close enough." It begins by calculating the body-frame forward kinemetics using the initial joint position guess to produce a transformation matrix representing the end-effector relative to the space frame. It then calculates the error between the desired transformation matrix and the current transformation matrix. It then enters the main loop.

The main loop begins by calculating the inverse of the current transformation matrix and then multiplying it by the desired transformation matrix. This yields a transformation matrix that represents the desired transformation relative to the body frame. The log of this matrix is taken to produce a twist in the body frame. The pseudo-inverse of the current Jacobian is then multiplied against the 6-vector twist and the result is added to the current joint angle guess to yield the next joint angle guess. This new guess is used to calculate the new transformation matrix, and the error between this matrix and the desired matrix is calculated to determine if it falls below the threshold. If it does, the loop exits and the function returns the final guess. If it is not, the loop continues, using the new guess as the current guess for the next iteration, and it does not exit until the error betwee the two matrices has dropped below the threshold.

In this problem the desired transformation matrix is

$$T_{sd} = \begin{bmatrix} 0 & -1 & 0 & 2 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

and the initial joint angle guess is $(-10°, 80°)$. The first iteration gave new θ guesses of $(0.0759523°, 89.0493°)$, and the (x, y) position of the end-effector at those angles was $(2.01527, 1.00253)$. Running again, the new joint angle guess was $(-.00000334602°, 90.0006°)$ with an end-effector position of $(1.99999, 1.0)$. This final iteration yielded a transformation matrix with the following form

$$T_{sd} = \begin{bmatrix} -0.0000109983 & -1 & 0 & 1.9999 \\ 1 & -0.0000109983 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

This final transformation matrix was close enough to T_{sd} to have an error less than the parameter threshold, so after two iterations this was the final transformation matrix the algorithm settled on.

As can be seen from this final transformation matrix, the axes are not completely orthogonal to each other. There is very slight floating point error where the values should be 0. Unfortunately this will be propagated and integrated through further calculations if additional code is not put in place to account for this error and make the axes completely orthogonal. The error can be lessened further if the error threshold is reduced in order to force a final transformation matrix to be even closer to the desired matrix. If the error threshold is set to $\epsilon = 10^{-6}$, the algorithm runs through three iterations and yields a transformation matrix of

$$T_{sd} = \begin{bmatrix} -3.22791 \times 10^{-13} & -1 & 0 & 2 \\ 1 & -3.22791 \times 10^{-13} & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

While this does not completely eliminate integration errors, it does decrease them significantly while only adding one more iteration to the process.