Laboratory Assistant Arm

MSE 4401 – Robotic Manipulators

Dylan Meikle – 251237698

Eric Montag - 251233562

Forward Kinematics

Refer to Appendix for Forward Kinematics script, tables for joint sets, tables for link length sets, and drawings of manipulators

$$\begin{pmatrix} 0 & -1 & 0 & \frac{685\sqrt{2}}{2} + 1245 \\ 1 & 0 & 0 & \frac{685\sqrt{2}}{2} + 685 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 1.1: Transformation Matrix using Link Set 1 and Joint Set 1

$$\begin{pmatrix} 0 & -1 & 0 & 125\sqrt{2} + 1000 \\ 1 & 0 & 0 & 125\sqrt{2} + 250 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 1.2: Transformation Matrix using Link Set 2 and Joint Set 1

The Figure 1.1 and 1.2 transformation matrices show the alignment of the x-axis and y-axis of the end effector frame with the y-axis and negative x-axis of the base frame, respectively. The z-axes of the end effector frame and the base frame align, which makes sense as all the z-axes are parallel, and our manipulator doesn't have the joints to change that. Additionally, the displacements relative to the base frame indicate that x is equal to the first link length fully vertical plus one of the following links at a 45-degree angle and that y is equal to one of the last two links at a 45-degree angle plus one of the last two links fully horizontal. It can also be mentioned that the z displacement simply equals the d_1 value of the prismatic joint.

$$\begin{pmatrix} 0 & -1 & 0 & 1930 \\ 1 & 0 & 0 & 685 \\ 0 & 0 & 1 & 25 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 1.3: Transformation Matrix using Link Set 1 and Joint Set 2

$$\begin{pmatrix} 0 & -1 & 0 & 1250 \\ 1 & 0 & 0 & 250 \\ 0 & 0 & 1 & 25 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 1.4: Transformation Matrix using Link Set 2 and Joint Set 2

The Figure 1.3 and 1.4 transformation matrices show that the x, y, and z axes of the end effector frame align with the positive y, negative x, and positive z axes of the base frame, respectively. For displacements, the z-axis displacement is still simply equal to the displacement of the prismatic joint no matter the configuration of the joints as all the z-axes are parallel. The y-axis displacement is equal to the length of either link 2 or 3 if it were fully horizontal and the x-axis displacement is equal to the sum of the length of link 1 and either link 2 or 3 fully vertical

which follows our drawing of position 2 as well as the alternate configuration where l2 is fully horizontal and l3 is fully vertical.

$$\begin{pmatrix} -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 & 1245 \\ -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 & -685\sqrt{2} \\ 0 & 0 & 1 & 50 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 1.5: Transformation Matrix using Link Set 1 and Joint Set 3

$$\begin{pmatrix} -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 & 1000 \\ -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 & -250\sqrt{2} \\ 0 & 0 & 1 & 50 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 1.6: Transformation Matrix using Link Set 2 and Joint Set 3

The Figure 1.5 and 1.6 transformation matrices show that, while the z-axes are still fully aligned, the x and y axes are at a 45-degree angle to each other. The first column indicates that the x-axis of the end effector frame is pointed opposite the x and y axes of the base frame at an angle of -45 degrees from each which tells us that it is in the lower left quadrant relative to the base frame. The second column indicates that the y-axis of the end effector frame is pointed opposite the y axis of the base frame but pointed up with the x-axis as only 45 degrees away. This means that the y-axis of the end effector is in the upper left quadrant relative to the base frame. All these conclusions align with our drawing of the manipulator.

The original link lengths were obtained experimentally using SolidWorks to roughly position all critical points in our workspace and test various link lengths. The first link length was determined based on the expected height of the worksurface and the second and third lengths using the max length of the table as well as positions of critical points. The second and third lengths were made the same for manufacturability purposes as it simplified the process if only 2 different links needed to be manufactured. Using our SolidWorks model, we can see that the links at sufficiently long to reach all potential locations in the workspace and not so long as to collide with the ceiling or other environmental obstacles.

Inverse Kinematics

Refer to Appendix for Inverse Kinematics script and tables for arm positions used for validation.

The following table shows the end effector positions for each combination of link lengths and joint variables as calculated using the forward kinematics function, and these values are used in the inverse kinematics function to determine the necessary joint angles to achieve these end effector positions.

	Link Lengths 1			Link Lengths 2		
	Position 1	Position 2	Position 3	Position 1	Position 2	Position 3
x	1729.37	1930	1245	1176.78	1250	1000
y	1169.37	685	-968.74	426.78	250	-353.55
z	0	25	50	0	25	50

The following table shows, for each combination of link lengths and joint variables, the forward transformation matrix calculated using the defined joint variables, and the forward transformation matrices corresponding to the positive and negative solutions of the inverse kinematics problem. It can be seen that for a given set of link lengths and joint variables the forward kinematics function gives the coordinates of the end effector. The inverse kinematics function takes these coordinates and calculates the necessary joint variables. These joint variables are shown to give the same forward transformation matrix as calculated using the user-defined joint variables in the forward kinematics function.

Joint Variables Test 1

```
T1_1 = 4×4
10^3 ×
         0   -0.0010         0    1.7294
    0.0010         0         0    1.1694
         0         0    0.0010         0
         0         0         0    0.0010

Warning: The function sym/eval is deprecat
Depending on the usage, use subs or double
ans = 4×4
10^3 ×
    0.0000   -0.0010         0    1.7294
    0.0010    0.0000         0    1.1694
         0         0    0.0010         0
         0         0         0    0.0010

Warning: The function sym/eval is deprecat
Depending on the usage, use subs or double
ans = 4×4
10^3 ×
    0.0007   -0.0007         0    1.7294
    0.0007    0.0007         0    1.1694
         0         0    0.0010         0
         0         0         0    0.0010
```

Joint Variables Test 2

```
T2_1 = 4×4
         0         -1         0       1930
         1          0         0        685
         0          0         1         25
         0          0         0          1

Warning: The function sym/eval is deprecated and wi
Depending on the usage, use subs or double instead.
ans = 4×4
         0         -1         0       1930
         1          0         0        685
         0          0         1         25
         0          0         0          1

Warning: The function sym/eval is deprecated and wi
Depending on the usage, use subs or double instead.
ans = 4×4
         1          0         0       1930
         0          1         0        685
         0          0         1         25
         0          0         0          1
```

Joint Variables Test 3

```
T3_1 = 4×4
10^3 ×
   -0.0007    0.0007         0    1.2450
   -0.0007   -0.0007         0   -0.9687
         0         0    0.0010    0.0500
         0         0         0    0.0010

Warning: The function sym/eval is deprecate
Depending on the usage, use subs or double
ans = 4×4
10^3 ×
    0.0007    0.0007         0    1.2450
   -0.0007    0.0007         0   -0.9687
         0         0    0.0010    0.0500
         0         0         0    0.0010

Warning: The function sym/eval is deprecate
Depending on the usage, use subs or double
ans = 4×4
10^3 ×
   -0.0007    0.0007         0    1.2450
   -0.0007   -0.0007         0   -0.9687
         0         0    0.0010    0.0500
         0         0         0    0.0010
```

```
T1_2 = 4×4
10^3 ×
         0   -0.0010         0    1.1768
    0.0010         0         0    0.4268
         0         0    0.0010         0
         0         0         0    0.0010

Warning: The function sym/eval is deprecate
Depending on the usage, use subs or double
ans = 4×4
10^3 ×
    0.0000   -0.0010         0    1.1768
    0.0010    0.0000         0    0.4268
         0         0    0.0010         0
         0         0         0    0.0010

Warning: The function sym/eval is deprecate
Depending on the usage, use subs or double
ans = 4×4
10^3 ×
    0.0007   -0.0007         0    1.1768
    0.0007    0.0007         0    0.4268
         0         0    0.0010         0
         0         0         0    0.0010
```

```
T2_2 = 4×4
         0         -1         0       1250
         1          0         0        250
         0          0         1         25
         0          0         0          1

Warning: The function sym/eval is deprecated and wi
Depending on the usage, use subs or double instead.
ans = 4×4
         0         -1         0       1250
         1          0         0        250
         0          0         1         25
         0          0         0          1

Warning: The function sym/eval is deprecated and wi
Depending on the usage, use subs or double instead.
ans = 4×4
         1          0         0       1250
         0          1         0        250
         0          0         1         25
         0          0         0          1
```

```
T3_2 = 4×4
10^3 ×
   -0.0007    0.0007         0    1.0000
   -0.0007   -0.0007         0   -0.3536
         0         0    0.0010    0.0500
         0         0         0    0.0010

Warning: The function sym/eval is deprecate
Depending on the usage, use subs or double
ans = 4×4
10^3 ×
    0.0007    0.0007         0    1.0000
   -0.0007    0.0007         0   -0.3536
         0         0    0.0010    0.0500
         0         0         0    0.0010

Warning: The function sym/eval is deprecate
Depending on the usage, use subs or double
ans = 4×4
10^3 ×
   -0.0007    0.0007         0    1.0000
   -0.0007   -0.0007         0   -0.3536
         0         0    0.0010    0.0500
         0         0         0    0.0010
```

We will impose a restriction on θ_2 that it must lie between $-\pi/2$ and $+\pi/2$ for the purposes of avoiding the elbow-down solution of our manipulator and thus avoiding hitting our workbench. We will also impose a restriction on d_1 that it must lie between 0mm and 2500mm to allow access to the full width of the laboratory bench.

Differential Kinematics

Looking at column 1, it makes sense that the prismatic joint effects only the linear velocity in the z-direction as it can only move linearly in that direction. Looking in the third row, we can see that none of the other joints can impact the linear velocity in the z-direction due to fact that all their z-axes are parallel. Since all the z-axes are parallel, it also makes sense that all the rotational velocities are zero except for those of the z-axes of joints 1 and 2 since they are rotary joints.

$$\begin{pmatrix} 0 & -\frac{685\sqrt{2}}{2} - 685 & -685 \\ 0 & \frac{685\sqrt{2}}{2} & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

Figure 3.1: Jacobian Matrix using Link Set 1 and Joint Set 1

$$\begin{pmatrix} 0 & -125\sqrt{2} - 250 & -250 \\ 0 & 125\sqrt{2} & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

Figure 3.2: Jacobian Matrix using Link Set 2 and Joint Set 1

$$\begin{pmatrix} 0 & -685 & -685 \\ 0 & 685 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

Figure 3.3: Jacobian Matrix using Link Set 1 and Joint Set 2

$$\begin{pmatrix} 0 & -250 & -250 \\ 0 & 250 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

Figure 3.4: Jacobian Matrix using Link Set 2 and Joint Set 2

$$\begin{pmatrix} 0 & 685\sqrt{2} & \frac{685\sqrt{2}}{2} \\ 0 & 0 & -\frac{685\sqrt{2}}{2} \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

Figure 3.5: Jacobian Matrix using Link Set 1 and Joint Set 3

$$\begin{pmatrix} 0 & 250\sqrt{2} & 125\sqrt{2} \\ 0 & 0 & -125\sqrt{2} \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

Figure 3.6: Jacobian Matrix using Link Set 2 and Joint Set 3

If we look specifically at Figure 3.1 to analyze the linear x-component, we can see that the linear velocity in the x-direction will is controlled by rotary joints 2 and 3 by factors representing their respective distances from the end-effector. Looking at the y-component in Figure 3.1 with reference to Figure A.1: Diagram 1, it makes sense that only rotary joint 2 impact the velocity as, at this specific pose, link 3 is fully horizontal and so there is no horizontal distance to apply to

the y-components from rotary joint 3. Conversely, Figure 3.5 shows that the y-component is only impacted by rotary joint 3 but this makes sense when referencing Figure A.1: Diagram 2 as the end effector and joint 3 are aligned vertically so there is no distance between them in the y-direction where there is distance between the end-effector and joint 3 vertically.

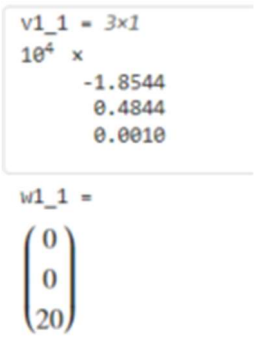

Figure 3.7: Velocities of Link Lengths Set 1

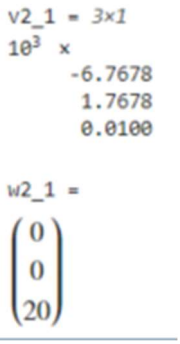

Figure 3.8: Velocities of Link Lengths Set 2

Looking at the velocities for the two different sets of link lengths which are based on Table A.1, Table A.2: Position 1, and a set of joint velocities, $\dot{q} = [\,10\ \frac{mm}{s}\ 10\ \frac{rad}{s}\ 10\ \frac{rad}{s}\,]$, one can see that the angular velocities about the z-axis are simply the sum of the two angular velocities (2*10 rad/s) and the linear velocity along the z-axis is the same for both and equal to the speed of the prismatic joint as it is the only joint that can change the position in the z-direction. The signs of the linear velocities make sense for pose 1 as both angular velocities will be acting clockwise, so the velocity vector will be pointing down and toward the right which translates to a negative sign on our x-components and positive sign on our y-component of our velocity acting on the end effector relative to the base frame. Comparing the two velocity vectors for the different link lengths, we can see that the velocity of link lengths set 1 (Figure 3.7) surpasses link lengths set 2 by a magnitude in the x-components of the linear velocity but they are otherwise similar. This difference in magnitude simply arises from the difference in the link lengths of the two different concepts.

Asses Your Design

Our final design does meet our requirements as listed. It sits very close to the laboratory table with just two links hanging over the table which move in a plane perpendicular to the width of the table. This means the robot takes up very little space over the top of the table while still reaching all necessary locations in the workspace. Thus, our configuration minimizes size while maintaining manipulability. The orientation of our rotary joints allows the robot to move out of the workspace over the table and avoid larger obstacles.

Additionally, our first set of joint values places the end effector at a vertical position of x = 1729.37, meaning that our manipulator can reach all heights necessary as described in our configuration selection matrix. In terms of manipulating pipettes according to our project application, we chose relatively short links, each of the same length. This helps minimize positions where one long link causes the end effector velocity to increase very quickly, meaning pipettes can be moved accurately according to our desired positions and paths.

Our finalized configuration is PRR, with link lengths of 1245mm, 685mm and 685mm. These allow us to meet our requirements as stated above. We will impose a range of allowable joint values such that $-\pi/2 < \theta_2 < \pi/2$. This prevents arm-down type solutions for the inverse kinematics problem which stops the second link from hitting our lab benchtop. Additionally, d_1 should be limited from 0mm to 2500mm. This allows the arm to access the full width of the workbench including any objects that may be on the left and right edges.

Finally, our end effector would be a simple rubber gripper with two fingers, to hold the pipettes securely and actuate the bulb.

Appendix

	Concept 1 Link Lengths	Concept 2 Link Lengths
Link 1	1245mm	1000mm
Link 2	685mm	250mm
Link 3	685mm	250mm

Table A.1: Link Length Concepts

	Position 1	Position 2	Position 3
d_1^*	0	20	50
θ_2^*	pi/4	0	-pi/4
θ_3^*	pi/4	pi/2	-pi/2

Table A.2: Joint Variables for Testing Positions

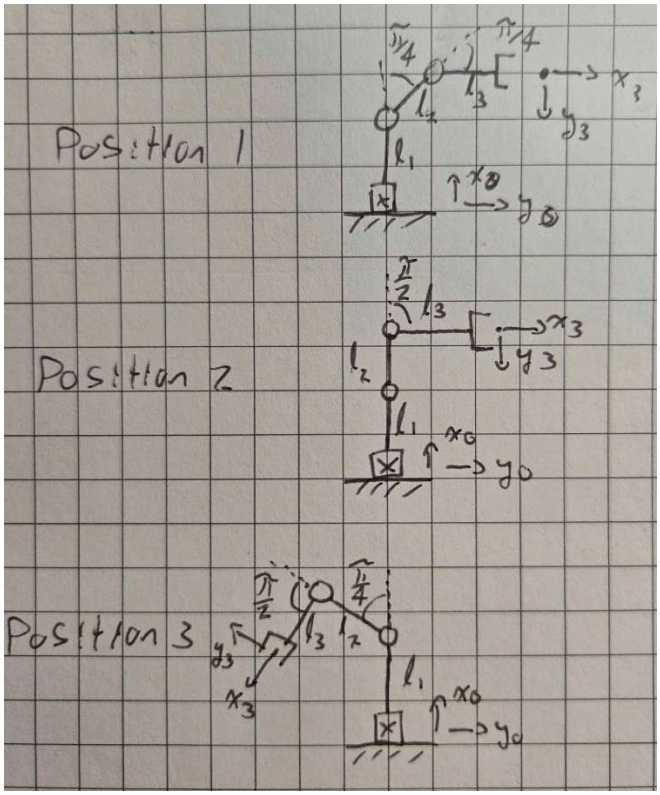

Figure A.1: Drawings of Manipulator in Various Positions

Code:

Forward Kinamatics

```
clc; clear;
syms l1 l2 l3 d1 d2 d3 theta1 theta2 theta3
%DH table specific to our robot
DH =    [l1 d1 0 0;
         l2 0 0 theta2;
         l3 0 0 theta3];

%Each row is a different position to test
q = [0 pi/4 pi/4    %Pose 1
    25 0 pi/2       %Pose 2
    50 -pi/4 -pi/2];%Pose 3

%Each row is a set of link lengths for a robot
l = [1245 685 685 %Link Lengths set 1
    1000 250 250];%Link Lengths set 2

%Creating the robot object (they only vary by link lengths we supply as we
%give them both the same DH parameters
bot1 = manipulator(DH,l(1,:));
bot2 = manipulator(DH,l(2,:));

%Calculating Transformation matrices for first set of link lengths
T1_1 = bot1.fkine(q(1,:))
T2_1 = bot1.fkine(q(2,:))
T3_1 = bot1.fkine(q(3,:))



%Calculating Transformation matrices for second set of link lengths
T1_2 = bot2.fkine(q(1,:))
T2_2 = bot2.fkine(q(2,:))
T3_2 = bot2.fkine(q(3,:))

%Same as above but evaluated to decimal numbers for conveniance of reading
%results
T1_1 = eval(T1_1)
T2_1 = eval(T2_1)
T3_1 = eval(T3_1)

T1_2 = eval(T1_2)
T2_2 = eval(T2_2)
T3_3 = eval(T3_2)
```

Inverse Kinamatics

```
%Calculating the inverse kinematics for pose 1
[d1 theta2 theta3] = bot1.ikine([1729.4 1169.4 0]);
%Transformation matrix calculated previouly to check results
T1_1
%Validating inverse kinematics results by running joint angles back through
%the forward kinematics to compare transformation matrices
eval(bot1.fkine([d1 theta2(1,1) theta3(1,1)]))
eval(bot1.fkine([d1 theta2(1,2) theta3(1,2)]))

%Calculating the inverse kinematics for pose 1
[d1 theta2 theta3] = bot1.ikine([1930 685 20]);
T2_1
eval(bot1.fkine([d1 theta2(1,1) theta3(1,1)]))
eval(bot1.fkine([d1 theta2(1,2) theta3(1,2)]))

%Calculating the inverse kinematics for pose 1
[d1 theta2 theta3] = bot1.ikine([1245 -968.7 50]);
T3_1
eval(bot1.fkine([d1 theta2(1,1) theta3(1,1)]))
eval(bot1.fkine([d1 theta2(1,2) theta3(1,2)]))
```

Jacobian

```
q = [0 pi/4 pi/4
    25 0 pi/2
    50 -pi/4 -pi/2];

l = [1245 685 685
    1000 250 250];
q_dot = [10 10 10] %mm/s rad/s rad/s

%Calculating Jacobians for link sets 1 and 2 without the poses assigned
J1 = bot1.Jacobian()
J2 = bot2.Jacobian();

%Assigning poses to the Jacobians for link length sets 1 and 2 in 3
%different poses
J1_1 = bot1.Jacobian(q(1,:))
J2_1 = bot2.Jacobian(q(1,:))

J1_2 = bot1.Jacobian(q(2,:))
J2_2 = bot2.Jacobian(q(2,:))

J1_3 = bot1.Jacobian(q(3,:))
J2_3 = bot2.Jacobian(q(3,:))


% J1_vel = eval(bot1.Jacobian(q(1,:))) *
%Here we're calculating the angular and linear velocities for Jacobians
%J1_1 and J2_1 (Using link set 1 and link set 2, respectively) to compare
%how the different in link lengths impacts the velocities
Jv1_1 = J1_1(1:3,:);
Jw1_1 = J1_1(4:6,:);
v1_1 = eval(Jv1_1 * q_dot.') %.' just transposes them to colum instead of row vectors
w1_1 = Jw1_1 * q_dot.'

Jv2_1 = J2_1(1:3,:);
Jw2_1 = J2_1(4:6,:);
v2_1 = eval(Jv2_1 * q_dot.')
w2_1 = Jw2_1 * q_dot.'
```

Manipulator Class Definition and Methods

```
classdef manipulator < handle

    properties
        DH          %DH parameters that the user inputs
        A_Mats      %3D array to hold all the A matrices
        Trans       %General transformation matrix for manipulator
        numJoints   %Number of joints in the system
        config      %Type of joints and their order from base to EE
        sing        %singularities
        links       %link lengths

    end

    methods

        %Constructor
        function self = manipulator(DH,l)

            self.DH = DH;
            self.numJoints = size(self.DH,1);
            self.links = l;
            self.Trans = self.fkine();

        end

        %Prints all the current manipulator parameters
        function currentProperties(self) ...

        %This function calculates the forward kinematics of the robotic
        %manipulator and return the general tranformation matrix or the
        %transformation matrix for a given set of joint positions and link
        %lengths
        function Trans = fkine(self,q)
            %q is an optional arguments
            arguments
                self
                q = []
            end
            %Needs to be here so we can use subs function later
            syms l1 l2 l3


            if isempty(self.Trans)


                numJoints = self.numJoints;
                %Have to setup array with syms otherwise MATLAB will complain about using
                %numeric vs variables
                A_Mats = sym(zeros(4,4,numJoints));



                for index = 1:numJoints
                    %User must define the DH table for a given robot
                    a = self.DH(index,1);
                    d = self.DH(index,2);
                    alpha = self.DH(index,3);
                    theta = self.DH(index,4);
```

```
            % Build the A_i matrix
            A_Mats(:,:,index) = createMatrix(a,d,alpha,theta);
        end



        self.A_Mats = A_Mats(:,:,:);


        % Multiply all transformation matrices to get the end effector
        % frame with respect to the base frame
        Trans = eye(4);
        for index = 1:numJoints
            Trans = Trans * A_Mats(:,:,index);
        end

        sympref('AbbreviateOutput', false);
        %Simplifies the transformation matrix
        Trans = simplify(Trans,"Steps",400);
        %This removes the pi/180 that appears when you just use the simplify
        %function to display
        %Trans = subs(Trans,pi/180,1);
        Trans = subs(Trans, {l1, l2, l3}, {self.links(1,1), self.links(1,2), self.links(1,3)});
        %Saving the general transformation matrix as a property of the
        %manipulator
        self.Trans = Trans;


    else %If Trans has already been calculated
        Trans = self.Trans;
    end

    %If the variable for q isn't empty, then sub them into the
    %transformation matrix
    if ~isempty(q)
        vars = symvar(Trans);
        q_index = 1;
        l_index = 1;
        %Loop through all the unknowns in the transformation matrix
        for index = 1:length(vars)
            %If the varible name doesn't start with l
            if ~startsWith(string(vars(index)), "l")
                Trans = subs(Trans,vars(index),q(q_index));
                q_index = q_index + 1;
            %If the variable name does start with l
            else
                Trans = subs(Trans,vars(index),self.links(l_index));
                l_index = l_index + 1;
            end
        end
    end

end
```

```
%Given a position q, return necessary joint values to achieve
%that position
function [d1, theta2, theta3] = ikine(self, q)
    % q = [x y z] (desired EE position in base frame)
    x = q(1); y = q(2); z = q(3);
    l1 = self.links(1);
    l2 = self.links(2);
    l3 = self.links(3);

    d1 = z;               % prismatic first joint


    D = (y^2 + (x-l1)^2 - l2^2 - l3^2)/(2*l2*l3);
    theta3_minus = atan2(-sqrt(1-D^2),D);
    theta3_plus = atan2(sqrt(1-D^2),D);

    theta3 = [theta3_plus theta3_minus];

    %Changed the plus theta2 to addition and it makes more sense now
    theta2_minus = atan2(y,x-l1) - atan2(l3*sin(theta3_minus),l2+l3*cos(theta3_minus));
    %theta2_plus_test = atan((l2+l3*cos(theta3_plus))/(l3*sin(theta3_plus))) - atan((x-l1)/y)
    theta2_plus = atan2(y,(x-l1)) - atan2(l3*sin(theta3_plus),l2+l3*cos(theta3_plus));

    theta2 = [theta2_plus theta2_minus];
end
```

```
function J = Jacobian(self,q)
    %q is an optional argument
    arguments
        self
        q = []
    end
    %Needs to be here so we can use subs function later
    syms l1 l2 l3 d1 theta2 theta3

    %This assumes that the user has already run fkine
    numJoints = self.numJoints;
    A_Mats = self.A_Mats;
    links = self.links;

    %This identifies the configuration based on the unknowns in the
    %transformation matrix
    vars = symvar(self.Trans);
    var_index = 1;
    self.config = strings(1,numJoints); %preallocate as string array

    %Loop through all the unknowns in the transformation matrix
    for index = 1:length(vars)
        %If the varible name doesn't start with l
        if startsWith(string(vars(index)), "theta")
            self.config(var_index) = "R";
            var_index = var_index + 1;
        elseif startsWith(string(vars(index)),"d")
            self.config(var_index) = "P";
            var_index = var_index + 1;
        end
    end

    % Preallocate
    z = sym(zeros(3,numJoints+1));
    O = sym(zeros(3,numJoints+1));
    T = sym(eye(4));

    % Base frame
    z(:,1) = [0;0;1];
    O(:,1) = [0;0;0];
    self.A_Mats(:,:,1);
    % Compute cumulative transforms
    for i = 1:numJoints
        T = T * A_Mats(:,:,i);
        z(:,i+1) = T(1:3,3);
        O(:,i+1) = T(1:3,4);
    end
```

```
            % Build Jacobian
            J = sym(zeros(6,numJoints));
            for i = 1:numJoints
                if self.config(i) == "R"

                    Jv = cross(z(:,i), O(:,end) - O(:,i));
                    Jw = z(:,i);
                    J(:,i) = [Jv; Jw];

                elseif self.config(i) == "P"

                    Jv = z(:,i);
                    Jw = sym([0;0;0]); %Using syn incase Jv is symbolic
                    J(:,i) = [Jv; Jw];

                else

                    disp(["Issue with configuration:", self.config]);

                end

            end

            J = simplify(J);
            %Substituting values for links into the jacobian
            J = subs(J, {l1 l2 l3}, {links(1) links(2) links(3)});
            %If values for q were passed, sub them into the jacobian
            if ~isempty(q)

                J = subs(J, {d1, theta2, theta3}, {q(1) q(2) q(3)});

            end
            %Don't need these atm
            %{
            Singularity = simplify(det(J));
            subs(Singularity,pi/180,1);
            fprintf("Determinant of J: %s\n", Singularity);
            %}
        end
    end
end

%General form of the homgeneous transformation matrix
function A = createMatrix(a,d,alpha,theta)
    A = [cos(theta) -sin(theta)*cos(alpha)  sin(theta)*sin(alpha)  a*cos(theta);
         sin(theta)  cos(theta)*cos(alpha) -cos(theta)*sin(alpha)  a*sin(theta);
         0           sin(alpha)             cos(alpha)             d;
         0           0                      0                      1];
end
```