Forward Kinematics

```
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,:))
```

T1_1 =

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

```
T2_1 = bot1.fkine(q(2,:))
```

T2_1 =

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

```
T3_1 = bot1.fkine(q(3,:))
```

T3_1 =

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

```
%Calculating Transformation matrices for second set of link lengths
```

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

T1_2 =

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

```
T2_2 = bot2.fkine(q(2,:))
```

T2_2 =

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

```
T3_2 = bot2.fkine(q(3,:))
```

T3_2 =

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

```
%Same as above but evaluated to decimal numbers for convenaince of reading
%results
T1_1 = eval(T1_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
```

```
T2_1 = eval(T2_1)
```

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

```
T3_1 = eval(T3_1)
```

```
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
```

```
T1_2 = eval(T1_2)
```

```
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
```

```
T2_2 = eval(T2_2)
```

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

```
T3_3 = eval(T3_2)
```

```
T3_3 = 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
```

Inverse Kinematics

```
%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
```

```
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
```

```
%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)]))
```

```
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
```

```
eval(bot1.fkine([d1 theta2(1,2) theta3(1,2)]))
```

```
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
```

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

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

```
eval(bot1.fkine([d1 theta2(1,1) theta3(1,1)]))
```

```
ans = 4×4
           0          -1           0        1930
           1           0           0         685
           0           0           1          20
           0           0           0           1
```

```
eval(bot1.fkine([d1 theta2(1,2) theta3(1,2)]))
```

```
ans = 4×4
           1           0           0        1930
           0           1           0         685
```

```
        0            0            1           20
        0            0            0            1
```

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

```
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
```

```
eval(bot1.fkine([d1 theta2(1,1) theta3(1,1)]))
```

```
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
```

```
eval(bot1.fkine([d1 theta2(1,2) theta3(1,2)]))
```

```
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
```

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
```

```
q_dot = 1×3

    10    10    10
```

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

J1 =

$$\begin{pmatrix} 0 & -685\sin(\theta_2+\theta_3)-685\sin(\theta_2) & -685\sin(\theta_2+\theta_3) \\ 0 & 685\cos(\theta_2+\theta_3)+685\cos(\theta_2) & 685\cos(\theta_2+\theta_3) \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

```
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,:))
```

J1_1 =

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

```
J2_1 = bot2.Jacobian(q(1,:))
```

J2_1 =

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

```
J1_2 = bot1.Jacobian(q(2,:))
```

```
J1_2 =
```

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

```
J2_2 = bot2.Jacobian(q(2,:))
```

```
J2_2 =
```

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

```
J1_3 = bot1.Jacobian(q(3,:))
```

```
J1_3 =
```

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

```
J2_3 = bot2.Jacobian(q(3,:))
```

```
J2_3 =
```

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

```
% 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
```

```
v1_1 = 3×1
```

```
10^4 ×
   -1.8544
    0.4844
    0.0010
```

```
w1_1 = Jw1_1 * q_dot.'
```

w1_1 =

$$\begin{pmatrix} 0 \\ 0 \\ 20 \end{pmatrix}$$

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

v2_1 = *3×1*

```
10^3 ×
   -6.7678
    1.7678
    0.0100
```

```
w2_1 = Jw2_1 * q_dot.'
```

w2_1 =

$$\begin{pmatrix} 0 \\ 0 \\ 20 \end{pmatrix}$$