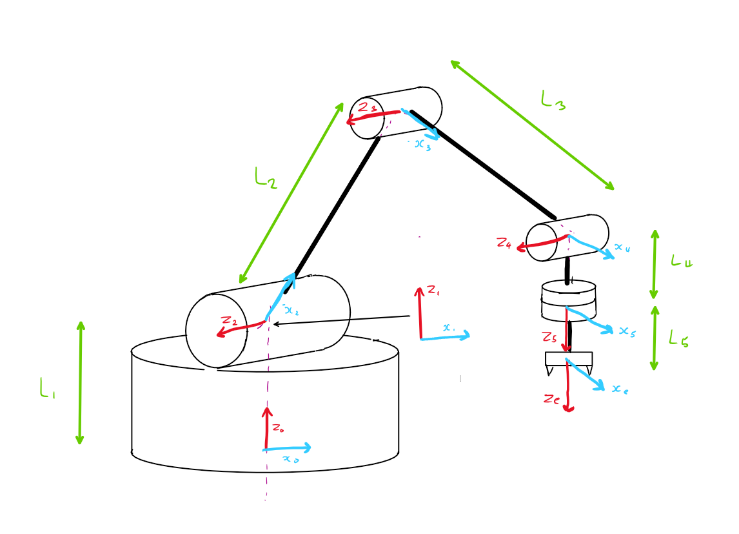
**Part I**

Robotics Fundamentals 2017

*Coursework Assignment - Sebastian Oakes*

Figure 1 - Diagram representing positions of assigned frames



***D-H representation of LynxMotion arm.***

In assigning coordinate frames to the links, the proximal (modified) Denavit-Hartenberg convention was employed. A few modifications were made to the normal frame assignment process to ensure that the joint positions are accurately represented. Frame 1 was attached to the distal end of link 1 to ensure the distance from the base to the first joint was accounted for, while still allowing for the 90˚ rotation required of the 2nd frame. Frame 4 is placed with the x axis perpendicular to both Z4 and the line joining frames 4 and 5, to allow the move between frames 4 and 5 to adhere to the ordering of the D-H convention. This is best displayed in figure 1, and it is noted that a theta value of zero between frames 3 and 4 leads to an automatic 90˚angle between links 3 and 4.

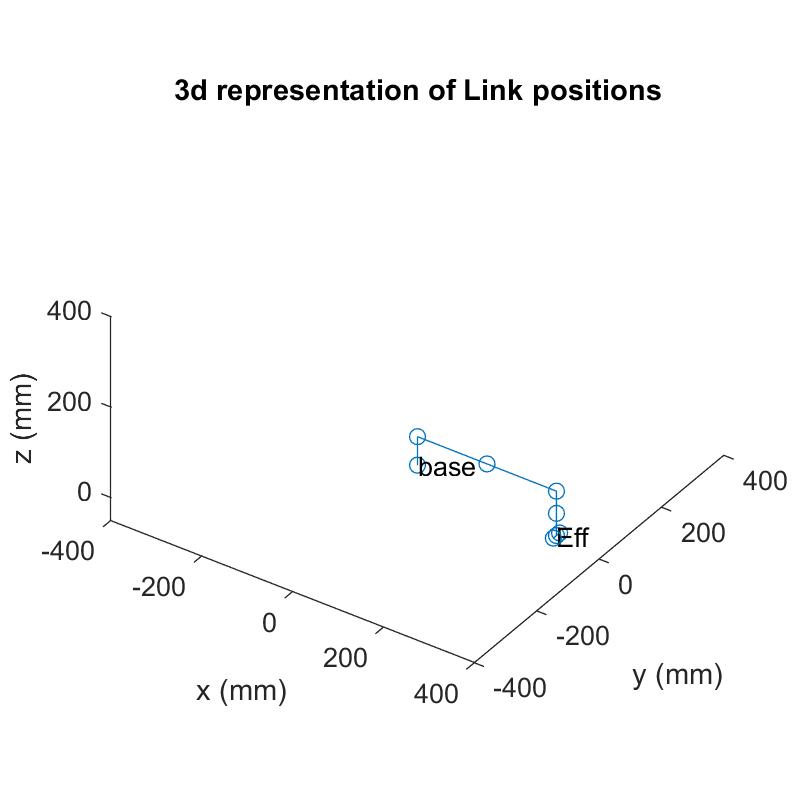
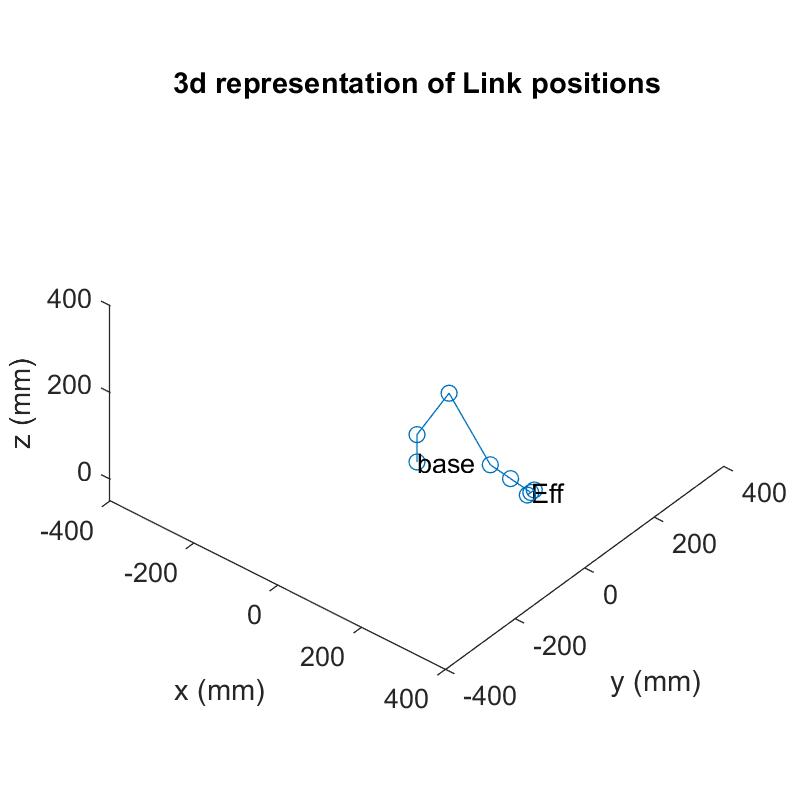


Figure 2 - 3D representations of joint positions after homogeneous transforms between frames. Top image displays arm with all θ values zero.

Some degree of consideration was required for the placement of the last two frames, since the 5th and 6th frames could arguably be combined. The method in figure 1 was chosen instead due to the more accurate representation of the physical arm, and the flexibility of being able to determine all link positions as they relate to the real system.

The D-H parameters are displayed in table 1, and follows the notation convention utilised by John J. Craig. The link lengths were determined through direct measurement of the test arm.

Table 1 - D-H table for LynxMotion arm.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i | αi-1 | ai-1 | di | θi |
| 1 | 0 | 0 | L1 | θ1 |
| 2 | +90˚ | 0 | 0 | θ2 |
| 3 | 0 | L2 | 0 | θ3 |
| 4 | 0 | L3 | 0 | θ4 |
| 5 | +90˚ | 0 | L4 | θ5 |
| 6 | 0 | 0 | L5 | 0 |

From this set of parameters, and through combining translation and rotation matrices, homogeneous transformation matrices can be built that describe the full transforms between each coordinate frame. It is here that MATLAB becomes invaluable, requiring only the individual 4x4 rotation and translation matrices to be input; the program can then easily combine all transformations into the final base to effector transform. The joint positions are simply the combined transforms required to reach that point in the kinematic chain, and are also easily determined. Figure 2 provides two example outputs from the program, showing both the default position, and a random variation of joint angles.

**Analysing the Workspace**

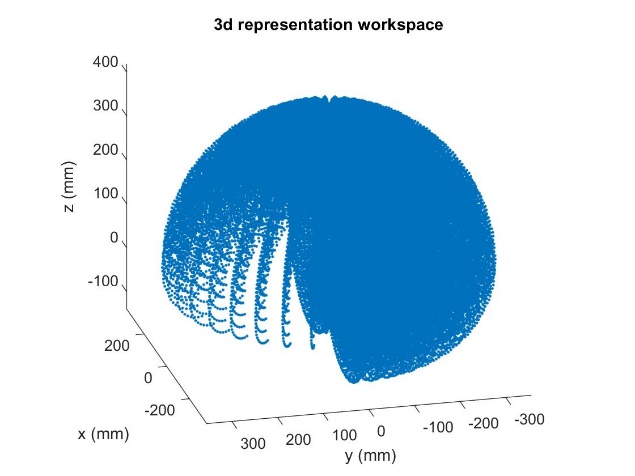
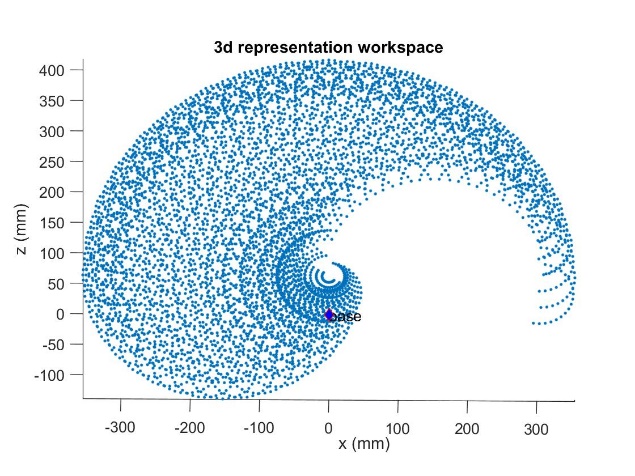


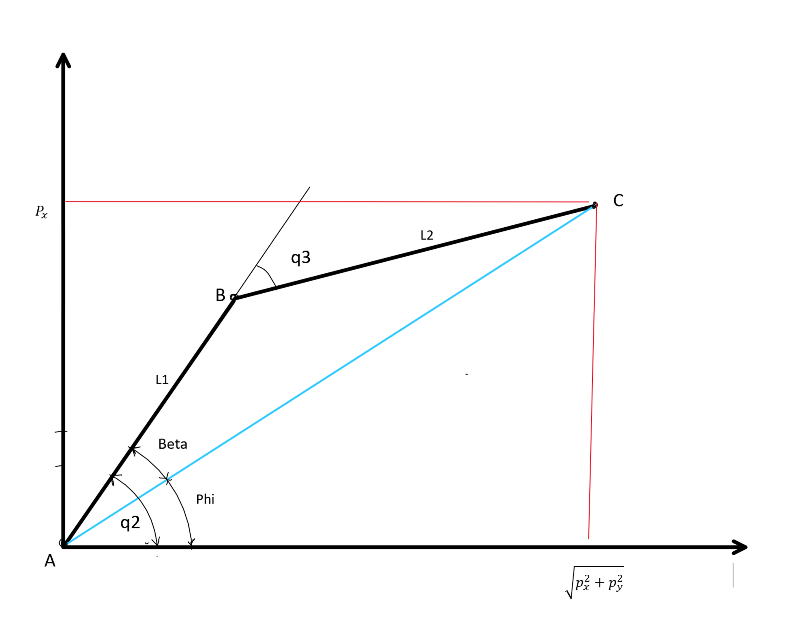
Figure 3 - 2D and 3D representations of workspace (of the 5th joint)

To analyse the workspace at the wrist (5th joint), for loops were utilised alongside the previously determined inverse kinematics model. To mark all possible positions of the wrist, each joint would need to be rotated about its full range of movement, for all other joint angles, achieved through the use of nested for loops. Each determined 5th frame position could then be added to a matrix, and later graphically displayed. The full transform to end effector was shortened to only the 5th joint.

It is important to note that limits must be set for the accurate reprentation of the workspace of the physical arm; servos have a limited angular range, and the morphology of the arm will restrict its possible movement. Information from the manufacturer’s data sheetgives angular ranges of 180˚, and by observation of the arm in operation, assumptions could be made about the rpossible angular relations between links (for example, links 2 and 3 can almost fold back on one another).

Figure 3 displays the results of this workspace analysis, and gives an accurate representation of possible joint positions, in particular highlighting the restriction caused by the servo in joint 1; the arm has severely restricted fidelity of m It is worth noting a limitation of the program as presented here; though the full angular range of each joint is displayed (limited by the actuators), the program does not account for possible collisions sections of the arm. It is obvious that many of the points close to the line joining the base and joint 1 would be unreachable due to collisions. A further extension of the program aim to alleviate this issue by carrying out collsion check, and generate a more accurate representation of the workspace in close proximity to the main robot body. A failsafe such as this would be particularly important if the code were to be implemented for control of the arm, since any collisions would likely cause damage to servo components.

Figure 4 - Geometric representation of links 2 and 3.



**Inverse Kinematics**

A derivation of an inverse kinematics model was formed largely through geometric considerations. Due to the matching orientation of joints 2, 3 and 4, the problem could largely be treated as a planar system, with rotation of the 1st joint affecting direction of any movement perpendicular to the z axis.

Once a target position and orientation was chosen, joint angles could be determined.

Firstly, θ1 (hereby q1) could be determined through consideration of the (x,y) coordinates of the target point, and therefore utilising:

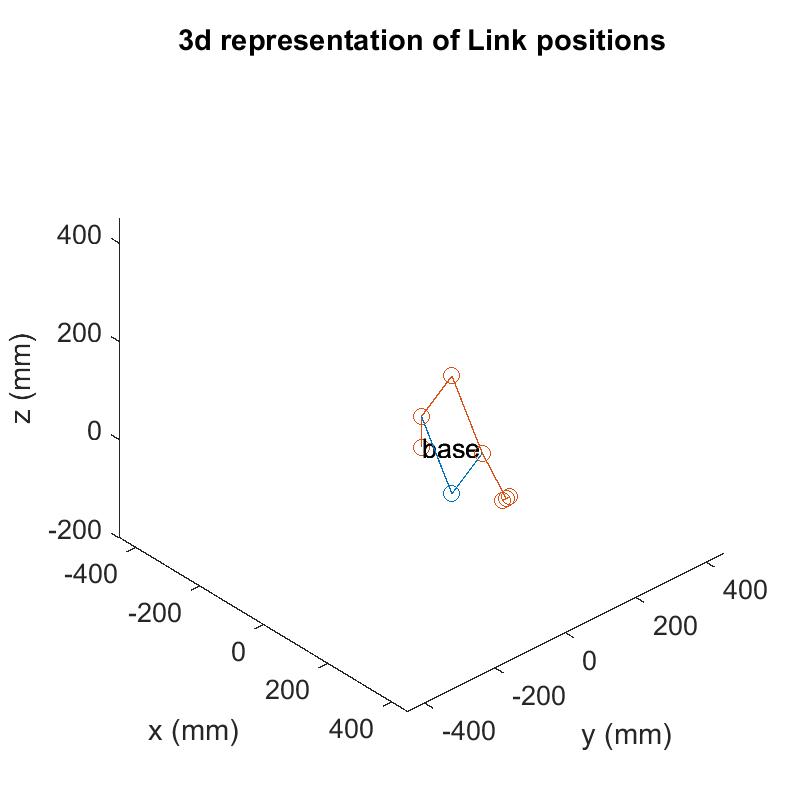
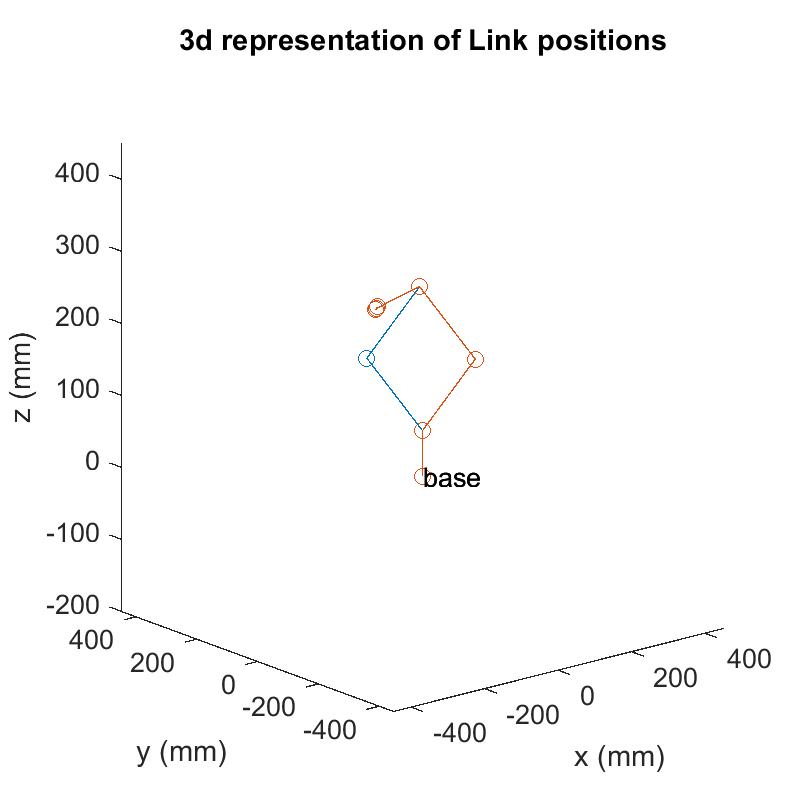


Figure 5 - Graphically displayed positions of arm, showing 4 solutions overlaid.

θ3  (hereby q3) is then determined through consideration of the geometry between joints 2, 3 and 4, as in figure 4. Therefore, q3 is given by:

Where, utilising the law of cosines for triangle ABC:

It then follows that:

Where:

(4)

To determine the orientation of the end effector, it is noted that:

θ234  denotes the total angular displacement of the effector, and can also be represented using ZYZ Euler angles. In particular, β (not the same as used in figure 4) represents the angle from z axis to a point, and is given by:

From this, any desired orientation can be chosen for the end effector, and θ4 values can be determined through combining equations (4) and (5).

Figure 5 displays two example results from an implementation of the above method within MATLAB. The written code initially uses the forward kinematics model to determine the homogeneous transformation matrix of the end effector, based on input joint angles. The inverse kinematics model is then applied to determine possible joint angles, thus allowing comparison between the two methods to ensure that a matching position and orientation is attained. Four solutions are overlaid, relating to the different possible combinations of joint angles.

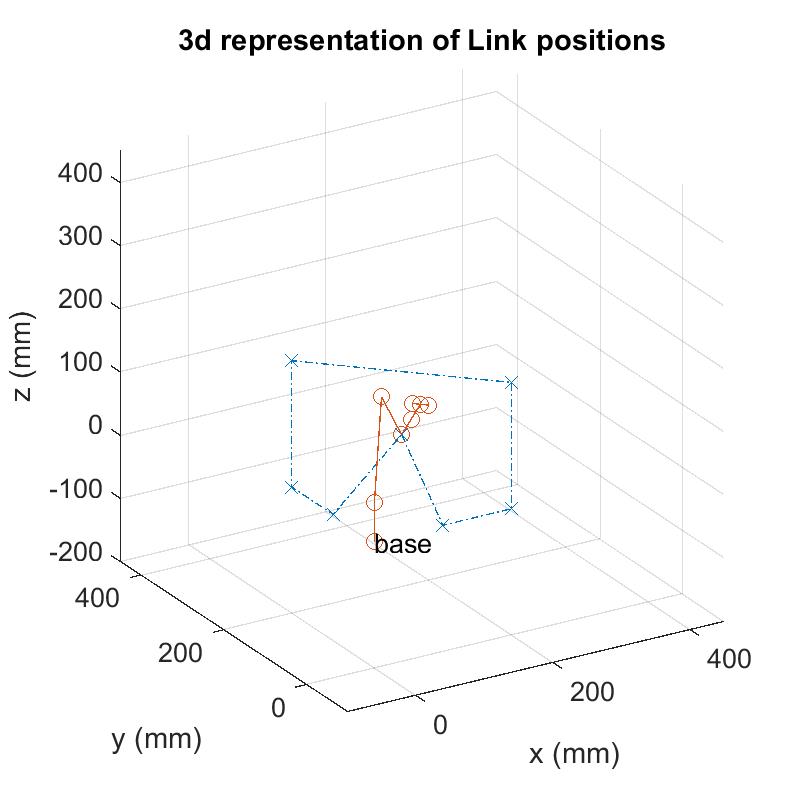
**Path Planning**

Figure 6 - 3D representation of end effector path for simulated object movement.

With the inverse kinematics model functioning correctly, a path could be planned for the effector to follow, in which the positions of joints are determined computationally for incremental steps along the path.

The path chosen aims to replicate the picking up and placement of a small object at an alternative location, with the end effector held level. This is perhaps representative of the movement of containers of liquid in a laboratory setting, for example. Figure 6 represents the start and end position of the arm, representing a stowed position before instruction is sent to the system. The program effectively determines the correct joint positions without error at every point along the path.

The program outputs four solutions, as with the previous inverse kinematics program, but does not contain a method of choosing the most suitable solution. This could perhaps be implemented by inclusion of cross checking of the initial stowed joint positions with the joint position results from the first path point; one solution will be much closer to the stowed position, and would therefore logically be considered the better choice for the any intial movements from the stow point.

This method would however also limit the possible link orientations; a better solution would perhaps involve cross checking joint values from the current effector position, and compare against the next projected position, so as to allow the program to make a decision on the more suitable solution based on the current orientation of links. This would also provide a method of switching between sets of solutions where needed (such as at the far reach of the arm, as θ3 tends to zero). This could be particularly useful in obstacle avoidance; both the end effector and the connecting links must be able to avoid the barrier. Though this may not be considered particularly useful with an arm of this type, 6DOF arms with many links would be capable of a huge range of solutions for a single effector position/orientation, and the capability to choose solutions to fit the confined or obstructed working environments may be essential.

**Code for function TransfMat.m (Required for other programs)**

%Seb Oakes

%function to determine homogeneous transformation matrix

function [T0e] = TransfMat(q1,q2,q3, q4, q5)

%link lengths

L1 = 63;

L2 = 153;

L3 = 153;

L4 = 49;

L5 = 49;

%trig abbrev.

c1 = cos(q1);

c2 = cos(q2);

c3 = cos(q3);

c4 = cos(q4);

c5 = cos(q5);

s1 = sin(q1);

s2 = sin(q2);

s3 = sin(q3);

s4 = sin(q4);

s5 = sin(q5);

baseF = [1 0 0 0;0 1 0 0; 0 0 1 0; 0 0 0 1];

z90 = [0 -1 0 0; 1 0 0 0; 0 0 1 0; 0 0 0 1]; %for +90 degree rotation about z

x90 = [1 0 0 0;0 0 -1 0;0 1 0 0;0 0 0 1]; % '' '' '' about x

trans10 = [1 0 0 0;0 1 0 0; 0 0 1 L1; 0 0 0 1];

rot10 = [c1 -s1 0 0; s1 c1 0 0; 0 0 1 0; 0 0 0 1];

trans21 = x90;

rot21 = [c2 -s2 0 0; s2 c2 0 0; 0 0 1 0; 0 0 0 1];

trans32 = [1 0 0 L2; 0 1 0 0; 0 0 1 0; 0 0 0 1];

rot32 = [c3 -s3 0 0; s3 c3 0 0 ; 0 0 1 0; 0 0 0 1];

trans34 = [1 0 0 L3; 0 1 0 0; 0 0 1 0; 0 0 0 1];

rot34 = [c4 -s4 0 0 ; s4 c4 0 0 ; 0 0 1 0; 0 0 0 1];

trans45 = [1 0 0 0; 0 1 0 0; 0 0 1 L4; 0 0 0 1];

rot45 = [c5 -s5 0 0; s5 c5 0 0; 0 0 1 0; 0 0 0 1];

transe5 = [1 0 0 0; 0 1 0 0; 0 0 1 L5; 0 0 0 1];

T01 = trans10\*rot10;

T12 = x90\*rot21;

T23 = trans32\*rot32;

T34 = trans34\*rot34;

T45 = x90\*trans45\*rot45;

T5e = transe5;

%TeL = [1 0 0 0; 0 1 0 10; 0 0 1 0; 0 0 0 1];

%TeR = [1 0 0 0; 0 1 0 -10; 0 0 1 0; 0 0 0 1];

T0e = T01\*T12\*T23\*T34\*T45\*T5e;

end

**CODE FOR FORWARD KINEMATICS**

clc

clear

close all

%deg to rad conversion

deg2rad = pi/180;

%Joint angles

q1 = 0\*deg2rad;

q2 = 90\*deg2rad;

q3 = -90\*deg2rad;

q4 = 10\*deg2rad;

q5 = 0\*deg2rad;

%link lengths

L1 = 63;

L2 = 153;

L3 = 153;

L4 = 49;

L5 = 49;

%trig abbrev.

c1 = cos(q1);

c2 = cos(q2);

c3 = cos(q3);

c4 = cos(q4);

c4\_90 = cos(q4+(pi/2));

c5 = cos(q5);

s1 = sin(q1);

s2 = sin(q2);

s3 = sin(q3);

s4 = sin(q4);

s5 = sin(q5);

s23 = sin(q2+q3);

s234 = sin(q2+q3+q4);

c23 = cos(q2+q3);

c234 = cos(q2+q3+q4);

baseF = [1 0 0 0;0 1 0 0; 0 0 1 0; 0 0 0 1];

%%

%Transformation Matrices

z90 = [0 -1 0 0; 1 0 0 0; 0 0 1 0; 0 0 0 1]; %for +90 degree rotation about z

x90 = [1 0 0 0;0 0 -1 0;0 1 0 0;0 0 0 1]; % '' '' '' about x

trans01 = [1 0 0 0;0 1 0 0; 0 0 1 L1; 0 0 0 1];

rot01 = [c1 -s1 0 0; s1 c1 0 0; 0 0 1 0; 0 0 0 1];

trans12 = x90;

rot12 = [c2 -s2 0 0; s2 c2 0 0; 0 0 1 0; 0 0 0 1];

trans23 = [1 0 0 L2; 0 1 0 0; 0 0 1 0; 0 0 0 1];

rot23 = [c3 -s3 0 0; s3 c3 0 0 ; 0 0 1 0; 0 0 0 1];

trans34 = [1 0 0 L3; 0 1 0 0; 0 0 1 0; 0 0 0 1];

rot34 = [c4 -s4 0 0 ; s4 c4 0 0 ; 0 0 1 0; 0 0 0 1];

trans45 = [1 0 0 0; 0 1 0 0; 0 0 1 L4; 0 0 0 1];

rot45 = [c5 -s5 0 0; s5 c5 0 0; 0 0 1 0; 0 0 0 1];

trans5e = [1 0 0 0; 0 1 0 0; 0 0 1 L5; 0 0 0 1];

T01 = trans01\*rot01;

T12 = x90\*rot12;

T23 = trans23\*rot23;

T34 = trans34\*rot34;

T45 = x90\*trans45\*rot45;

T5e = trans5e;

TeL = [1 0 0 0; 0 1 0 10; 0 0 1 0; 0 0 0 1];

TeR = [1 0 0 0; 0 1 0 -10; 0 0 1 0; 0 0 0 1]; %added translations to allow better orientationalrepresentation of end effector

% full homogeneous transform

T0e = T01\*T12\*T23\*T34\*T45\*T5e;

%% Joint positions

base = [1 0 0 0;0 1 0 0; 0 0 1 0; 0 0 0 1];

% joint 4x4 matrices after hom. transform

J1pos = T01\*base;

J2pos = T01\*T12\*base;

J3pos = T01\*T12\*T23\*base;

J4pos = T01\*T12\*T23\*T34\*base;

J5pos = T01\*T12\*T23\*T34\*T45\*base;

effpos = T01\*T12\*T23\*T34\*T45\*T5e\*base;

effL = T01\*T12\*T23\*T34\*T45\*T5e\*TeL\*base;

effR = T01\*T12\*T23\*T34\*T45\*T5e\*TeR\*base;

% joint position vectors

% create seperate x,y,z arrays for line plot

X = [base(1,4) J1pos(1,4) J2pos(1,4) J3pos(1,4) J4pos(1,4) J5pos(1,4) effpos(1,4) effL(1,4) effR(1,4)];

Y = [base(2,4) J1pos(2,4) J2pos(2,4) J3pos(2,4) J4pos(2,4) J5pos(2,4) effpos(2,4) effL(2,4) effR(2,4)];

Z = [base(3,4) J1pos(3,4) J2pos(3,4) J3pos(3,4) J4pos(3,4) J5pos(3,4) effpos(3,4) effL(3,4) effR(3,4)];

x5pos = L4\*(c4\*s1\*c23 - s4\*c1\*(c2\*c3 - s2\*c3)) + L3\*c1\*c23 + L2\*c1\*c2;

y5pos = L4\*(c4\*s1\*c23 - s4\*s1\*(c2\*s3 - s2\*c3)) + L3\*s1\*c23 - L2\*s1\*c2;

z5pos = L4\*(c4\*s23 - s4\*c23) + L3\*s23 + L2\*s2 +L1;

%% Plot lines connecting frames

figure (1)

set(1,'position',[200 300 400 400]) %Set figure position

plot3(X, Y, Z, '-o');

%plot3(x5pos,y5pos,z5pos, 'x');

axis equal

%text(Jmat1(1,4), Jmat1(2,4), Jmat1(3,4), 'J1')

%text(Jmat2(1,4), Jmat2(2,4), Jmat2(3,4), 'J2')

%text(Jmat3(1,4), Jmat3(2,4), Jmat3(3,4), 'J3')

%text(Jmat4(1,4), Jmat4(2,4), Jmat4(3,4), 'J4')

%text(Jmat5(1,4), Jmat5(2,4), Jmat5(3,4), 'J5')

title('3d representation of Link positions') ; xlabel('x (mm)') ; ylabel('y (mm)'), zlabel ('z (mm)') ;

axis([-400 400 -400 400 0 400])

text(0,0,0, 'base')

text(effpos(1,4), effpos(2,4), effpos(3,4), 'Eff')

hold on

%% Workspace

varq1 = (0:10:180)\*deg2rad; %180deg range - info found on technical sheet for arm

varq2 = (0:10:180)\*deg2rad; %180deg range

varq3 = (-10:10:170)\*deg2rad; %180deg range

varq4 = (0:10:180)\*deg2rad; %180 deg range

xwork = zeros(15000,1) ; % reserving space for the variables, because

ywork = zeros(15000,1) ; % otherwise they would be created later within a loop.

zwork = zeros(15000,1) ;

figure (2)

set(2,'position',[1243 190 560 420])

i=1; %set initial integer values for loops (to fill work arrays)

for q1 = varq1

for q2 = varq2

for q3 = varq3

for q4 = varq4

c1 = cos(q1);

c2 = cos(q2);

c3 = cos(q3);

c4 = cos(q4);

c5 = cos(q5);

s1 = sin(q1);

s2 = sin(q2);

s3 = sin(q3);

s4 = sin(q4);

s5 = sin(q5);

z90 = [0 -1 0 0; 1 0 0 0; 0 0 1 0; 0 0 0 1]; %for +90 degree rotation about z

x90 = [1 0 0 0;0 0 -1 0;0 1 0 0;0 0 0 1]; % '' '' '' about x

trans01 = [1 0 0 0;0 1 0 0; 0 0 1 L1; 0 0 0 1];

rot01 = [c1 -s1 0 0; s1 c1 0 0; 0 0 1 0; 0 0 0 1];

trans12 = x90;

rot12 = [c2 -s2 0 0; s2 c2 0 0; 0 0 1 0; 0 0 0 1];

trans23 = [1 0 0 L2; 0 1 0 0; 0 0 1 0; 0 0 0 1];

rot23 = [c3 -s3 0 0; s3 c3 0 0 ; 0 0 1 0; 0 0 0 1];

trans34 = [1 0 0 L3; 0 1 0 0; 0 0 1 0; 0 0 0 1];

rot34 = [c4 -s4 0 0 ; s4 c4 0 0 ; 0 0 1 0; 0 0 0 1];

trans45 = [1 0 0 0; 0 1 0 0; 0 0 1 L4; 0 0 0 1];

rot45 = [c5 -s5 0 0; s5 c5 0 0; 0 0 1 0; 0 0 0 1];

trans5e = [1 0 0 0; 0 1 0 0; 0 0 1 L5; 0 0 0 1];

T01 = trans01\*rot01;

T12 = x90\*rot12;

T23 = trans23\*rot23;

T34 = trans34\*rot34;

T45 = x90\*trans45\*rot45;

T5e = trans5e;

HomMat = T01\*T12\*T23\*T34\*T45\*base; % hom. transf. matrix for 5th joint.

xwork(i,1) = HomMat(1,4);

ywork(i,1) = HomMat(2,4);

zwork(i,1) = HomMat(3,4);

i=i+1; % add one to integer value

end

end

end

end

plot3(xwork,ywork,zwork, '.');

title('3d representation workspace') ; xlabel('x (mm)') ; ylabel('y (mm)'), zlabel ('z (mm)') ;

axis equal

hold on;

text(0,0,0, 'base');

scatter3(base(1,4),base(2,4),base(3,4), 'dr','MarkerFaceColor', 'b');

**CODE FOR INVERSE KINEMATICS**

%Seb Oakes

%Carry out inverse kinematic calculations for 5DOF LynxMotion arm

%requires TransfMat.m (transformation matrix function) to be open alongside this program.

clear all

deg2rad = pi/180;

rad2deg = 180/pi;

%% transformation matrix

%Joint angles

q1 = 0\*deg2rad;

q2 = 30\*deg2rad;

q3 = -80\*deg2rad;

q4 = 0\*deg2rad;

q5 = 0\*deg2rad;

%link lengths

L1 = 63;

L2 = 153;

L3 = 153;

L4 = 49;

L5 = 49;

T0e = TransfMat(q1,q2,q3,q4,q5);

T4e = [cos(q5) -sin(q5) 0 0; 0 0 -1 -(L5+L4); sin(q5) cos(q5) 0 0; 0 0 0 1]; %transform necessary for later calcs

T04 = T0e\*inv(T4e);

%%

%full transform in algebraic form

%r11 = c1\*c234\*c5 - s1\*s5;

%r12 = -c1\*c234\*s5 - s1\*c5;

%r13 = -c1\*s234;

%r21 = s1\*c234 + c1\*s5;

%r22 = s1\*c234 +c1\*c5;

%r23 = -s1\*s234;

%r31 = c5\*s234;

%r32 = -s5\*s234;

%r33 = c234;

%px = c1\*(c234\*c5 + L2\*c2 + L3\*c23 + (L4+L5)\*s234) - s1\*s5;

%py = s1\*(c234\*c5 + L2\*c2 + L3\*c23 + (L4+L5)\*s234) + c1\*s5;

%pz = c5\*s234 + L1 + L2\*s2 + L3\*s23 - L4\*c234 - L5\*c234;

%%

%desired Transformation matrix

Pdes = T0e; %desired position of wrist

Peff = T0e;

%Pdes =

T01 = [cos(q1) -sin(q1) 0 0; sin(q1) cos(q1) 0 0; 0 0 1 L1; 0 0 0 1];

T14 = inv(T01)\*Pdes\*inv(T4e);

T04 = Pdes\*inv(T4e);

T1e = T01\* Pdes

%variables required for q2,q3 calclation.

x = T1e(1,4);

y = T1e(2,4);

z = T1e(3,4);

%determine theta1 using x,y position of Eff

Q1a = atan2(T1e(2,4),T1e(1,4));

Q1b = Q1a + pi;

% determine q3

rw2 = (x^2 + y^2);

D = -(rw2+ (z-L1)^2 -L2^2 - L3^2)/(2\*L2\*L3);

Q3a = atan2(sqrt(1-D^2),-D);

Q3b = atan2(-sqrt(1-D^2),-D);

%determine q2

Q2a = (atan2(z-L1,sqrt(x^2+y^2))) - (atan2(L3\*sin(Q3a),L2+L3\*cos(Q3a)));

Q2b = (atan2(z-L1,-sqrt(x^2+y^2))) - (atan2(L3\*sin(Q3a),L2+L3\*cos(Q3a)));

Q2c = (atan2(z-L1,sqrt(x^2+y^2))) - (atan2(L3\*sin(Q3b),L2+L3\*cos(Q3b)));

Q2d = (atan2(z-L1,-sqrt(x^2+y^2))) - (atan2(L3\*sin(Q3b),L2+L3\*cos(Q3b)));

%theta1 = atan((zwrist - L1)/sqrt(xwrist^2 + ywrist^2));

%theta2 = atan((zwrist - L1)/-sqrt(xwrist^2 + ywrist^2));

%alpha1 = atan((L3\*sin(Q3a))/(L2+L3\*cos(Q3a)));

%alpha2 = atan((L3\*sin(Q3b))/(L2+L3\*cos(Q3b)));

%Q2a = (theta1 + alpha1);

%Q2b = (theta1+ alpha2);

%Q2c = (theta2 + alpha1);

%Q2d = (theta2 + alpha2);

%determine q4

beta = (atan2(sqrt(Pdes(3,1)^2+Pdes(3,2)^2),Pdes(3,3))); %for orientation of effector

BetaDeg = beta\*rad2deg % (just to allow me to check the angle)

phi = beta - pi/2;

Q4a = -phi+(pi/2)-Q2a-Q3a; % four solutions, using all previous Q2,Q3 angles

Q4b =-phi+(pi/2)-Q2b-Q3a;

Q4c = -phi+(pi/2)-Q2c-Q3b;

Q4d = -phi+(pi/2)-Q2d-Q3b;

Q5=0;

%matrix holding solutions

Qsol = [Q1a Q2a Q3a Q4a;

Q1a Q2c Q3b Q4c; ];

%Qsol = [Q1a Q2a Q3a Q4a;

%Q1b Q2b Q3a Q4b;

% Q1a Q2c Q3b Q4c;

% Q1b Q2d Q3b Q4d];

display '2 solutions:';

QsolDeg = Qsol\*rad2deg

%% Plot arm position

for i=1:2 %loop to step through sets of solutions

Q1 = Qsol(i,1);

Q2 = Qsol(i,2);

Q3 = Qsol(i,3);

Q4 = Qsol(i,4);

Q1deg = Q1\*rad2deg; %angles in deg for reference

Q2deg = Q2\*rad2deg;

Q3deg = Q3\*rad2deg;

c1 = cos(Q1);

c2 = cos(Q2);

c3 = cos(Q3);

c4 = cos(Q4);

c5 = cos(Q5);

s1 = sin(Q1);

s2 = sin(Q2);

s3 = sin(Q3);

s4 = sin(Q4);

s5 = sin(Q5);

c23 = cos(Q2+Q3);

s23 = sin(Q2+Q3);

baseF = [1 0 0 0;0 1 0 0; 0 0 1 0; 0 0 0 1];

%%

%Transformation Matrices

z90 = [0 -1 0 0; 1 0 0 0; 0 0 1 0; 0 0 0 1]; %for +90 degree rotation about z

x90 = [1 0 0 0;0 0 -1 0;0 1 0 0;0 0 0 1]; % '' '' '' about x

trans01 = [1 0 0 0;0 1 0 0; 0 0 1 L1; 0 0 0 1];

rot01 = [c1 -s1 0 0; s1 c1 0 0; 0 0 1 0; 0 0 0 1];

trans12 = x90;

rot12 = [c2 -s2 0 0; s2 c2 0 0; 0 0 1 0; 0 0 0 1];

trans23 = [1 0 0 L2; 0 1 0 0; 0 0 1 0; 0 0 0 1];

rot23 = [c3 -s3 0 0; s3 c3 0 0 ; 0 0 1 0; 0 0 0 1];

trans34 = [1 0 0 L3; 0 1 0 0; 0 0 1 0; 0 0 0 1];

rot34 = [c4 -s4 0 0 ; s4 c4 0 0 ; 0 0 1 0; 0 0 0 1];

%trans45 = [1 0 0 0; 0 1 0 0; 0 0 1 L4; 0 0 0 1];

rot45 = [c5 -s5 0 0; s5 c5 0 0; 0 0 1 0; 0 0 0 1];

trans5e = [1 0 0 0; 0 1 0 0; 0 0 1 L4+L5; 0 0 0 1];

T01 = trans01\*rot01;

T12 = x90\*rot12;

T23 = trans23\*rot23;

T34 = trans34\*rot34;

T45 = x90\*rot45;

T5e = trans5e;

TeL = [1 0 0 0; 0 1 0 10; 0 0 1 0; 0 0 0 1];

TeR = [1 0 0 0; 0 1 0 -10; 0 0 1 0; 0 0 0 1]; %added translations to allow better orientationalrepresentation of end effector

% full homogeneous transform

T0e = T01\*T12\*T23\*T34\*T45\*T5e;

%% Joint positions

base = [1 0 0 0;0 1 0 0; 0 0 1 0; 0 0 0 1];

% joint 4x4 matrices after hom. transform

J1pos = T01\*base;

J2pos = T01\*T12\*base;

J3pos = T01\*T12\*T23\*base;

J4pos = T01\*T12\*T23\*T34\*base;

J5pos = T01\*T12\*T23\*T34\*T45\*base;

effpos = T01\*T12\*T23\*T34\*T45\*T5e\*base;

effL = T01\*T12\*T23\*T34\*T45\*T5e\*TeL\*base;

effR = T01\*T12\*T23\*T34\*T45\*T5e\*TeR\*base;

% joint position vectors

% create seperate x,y,z arrays for line plot

X = [base(1,4) J1pos(1,4) J2pos(1,4) J3pos(1,4) J4pos(1,4) J5pos(1,4) effpos(1,4) effL(1,4) effR(1,4)];

Y = [base(2,4) J1pos(2,4) J2pos(2,4) J3pos(2,4) J4pos(2,4) J5pos(2,4) effpos(2,4) effL(2,4) effR(2,4)];

Z = [base(3,4) J1pos(3,4) J2pos(3,4) J3pos(3,4) J4pos(3,4) J5pos(3,4) effpos(3,4) effL(3,4) effR(3,4)];

x5pos = L4\*(c4\*s1\*c23 - s4\*c1\*(c2\*c3 - s2\*c3)) + L3\*c1\*c23 + L2\*c1\*c2;

y5pos = L4\*(c4\*s1\*c23 - s4\*s1\*(c2\*s3 - s2\*c3)) + L3\*s1\*c23 - L2\*s1\*c2;

z5pos = L4\*(c4\*s23 - s4\*c23) + L3\*s23 + L2\*s2 +L1;

%% Plot lines connecting frames

figure (1)

set(1,'position',[200 300 400 400]) %Set figure position

plot3(X, Y, Z, '-o');

title('3d representation of Link positions') ; xlabel('x (mm)') ; ylabel('y (mm)'), zlabel ('z (mm)') ;

axis([-450 450 -450 450 -200 450])

%axis equal

text(0,0,0, 'base')

hold on

end

**CODE FOR STRAIGHT PATH PLANNING -STRAIGHT TRAJECTORY**

%Sebastian Oakes

%Program simulating the picking up and placing down of an object at another location, while

%keeping end effector level. Arm starts at, and returns to default position, coordinates (100,100,100)

clc

clear

close all

deg2rad = pi/180;

rad2deg = 180/pi;

%Define Link lengths

L1 = 63;

L2 = 153;

L3 = 153;

L4 = 49;

L5 = 49;

stepNum = 30; %steps between nodes

%list of x,y,z node coordinates for path planning

PNX = [100 100 200 200 0 0 0 100]';

PNY = [100 0 0 0 200 200 100 100]';

PNZ = [100 0 0 200 200 0 0 100]';

orient1 = [0 0 1; 0 -1 0; 1 0 0]; %orientation of end effector

%orient2 = [];

for k = 1:7 %loop for moving between nodes

%Line vectors to supply x,y,z positions for inverse Kin. PPx = pathpoint x

%plot3(PPx, PPy, PPz, 'o-');

%title('Visualisation of path nodes');

for j = 1:stepNum %loop to add points on path between nodes

for i = 2 % NOTE - i can be set as 1 or 2 to choose between solutions

PPx = linspace(PNX(k), PNX(k+1), stepNum);

PPy = linspace(PNY(k), PNY(k+1), stepNum);

PPz = linspace(PNZ(k), PNZ(k+1), stepNum);

%determine Q1

Q1a = atan2(PPy(j),PPx(j));

Q1b = Q1a + pi;

%determine Q3

x = PPx(j);

y = PPy(j);

z = PPz(j);

rw2 = (x^2 + y^2);

D = -(rw2+ (z-L1)^2 -L2^2 - L3^2)/(2\*L2\*L3);

Q3a = atan2(sqrt(1-D^2),-D);

Q3b = atan2(-sqrt(1-D^2),-D);

%determine Q2

Q2a = (atan2(z-L1,sqrt(x^2+y^2))) - (atan2(L3\*sin(Q3a),L2+L3\*cos(Q3a)));

Q2b = (atan2(z-L1,-sqrt(x^2+y^2))) - (atan2(L3\*sin(Q3a),L2+L3\*cos(Q3a)));

Q2c = (atan2(z-L1,sqrt(x^2+y^2))) - (atan2(L3\*sin(Q3b),L2+L3\*cos(Q3b)));

Q2d = (atan2(z-L1,-sqrt(x^2+y^2))) - (atan2(L3\*sin(Q3b),L2+L3\*cos(Q3b)));

%determine Q4

beta = pi/2; %(atan2(sqrt(orient1(3,1)^2+ orient1(3,2)^2),orient1(3,3))); %for orientation of effector

BetaDeg = beta\*rad2deg; % (just to allow me to check the angle)

phi = beta - (pi/2); %ie angle w.r.t horizontal

Q4a = -phi+(pi/2)-Q2a-Q3a; % four solutions, using all previous Q2,Q3 angles

Q4b =-phi+(3\*pi/2)-Q2b-Q3a;

Q4c =-phi+(pi/2)-Q2c-Q3b;

Q4d =-phi+(3\*pi/2)-Q2d-Q3b;

Q5=0;

%Qsol = [Q1a Q2a Q3a Q4a;

% Q1a Q2c Q3b Q4c];

%matrix holding solutions

Qsol = [Q1a Q2a Q3a Q4a;

Q1b Q2b Q3a Q4b;

Q1a Q2c Q3b Q4c;

Q1b Q2d Q3b Q4d];

Q1 = Qsol(i,1);

Q2 = Qsol(i,2);

Q3 = Qsol(i,3);

Q4 = Qsol(i,4);

Q1deg = Q1\*rad2deg; %angles in deg for reference

Q2deg = Q2\*rad2deg;

Q3deg = Q3\*rad2deg;

c1 = cos(Q1);

c2 = cos(Q2);

c3 = cos(Q3);

c4 = cos(Q4);

c5 = cos(Q5);

s1 = sin(Q1);

s2 = sin(Q2);

s3 = sin(Q3);

s4 = sin(Q4);

s5 = sin(Q5);

c23 = cos(Q2+Q3);

s23 = sin(Q2+Q3);

baseF = [1 0 0 0;0 1 0 0; 0 0 1 0; 0 0 0 1];

%%

%Transformation Matrices

z90 = [0 -1 0 0; 1 0 0 0; 0 0 1 0; 0 0 0 1]; %for +90 degree rotation about z

x90 = [1 0 0 0;0 0 -1 0;0 1 0 0;0 0 0 1]; % '' '' '' about x

trans01 = [1 0 0 0;0 1 0 0; 0 0 1 L1; 0 0 0 1];

rot01 = [c1 -s1 0 0; s1 c1 0 0; 0 0 1 0; 0 0 0 1];

trans12 = x90;

rot12 = [c2 -s2 0 0; s2 c2 0 0; 0 0 1 0; 0 0 0 1];

trans23 = [1 0 0 L2; 0 1 0 0; 0 0 1 0; 0 0 0 1];

rot23 = [c3 -s3 0 0; s3 c3 0 0 ; 0 0 1 0; 0 0 0 1];

trans34 = [1 0 0 L3; 0 1 0 0; 0 0 1 0; 0 0 0 1];

rot34 = [c4 -s4 0 0 ; s4 c4 0 0 ; 0 0 1 0; 0 0 0 1];

trans45 = [1 0 0 0; 0 1 0 0; 0 0 1 L4; 0 0 0 1];

rot45 = [c5 -s5 0 0; s5 c5 0 0; 0 0 1 0; 0 0 0 1];

trans5e = [1 0 0 0; 0 1 0 0; 0 0 1 L5; 0 0 0 1];

T01 = trans01\*rot01;

T12 = x90\*rot12;

T23 = trans23\*rot23;

T34 = trans34\*rot34;

T45 = x90\*trans45\*rot45;

T5e = trans5e;

TeL = [1 0 0 0; 0 1 0 10; 0 0 1 0; 0 0 0 1];

TeR = [1 0 0 0; 0 1 0 -10; 0 0 1 0; 0 0 0 1]; %added translations to allow better orientationalrepresentation of end effector

% full homogeneous transform

T0e = T01\*T12\*T23\*T34\*T45\*T5e;

%% Joint positions

base = [1 0 0 0;0 1 0 0; 0 0 1 0; 0 0 0 1];

% joint 4x4 matrices after hom. transform

J1pos = T01\*base;

J2pos = T01\*T12\*base;

J3pos = T01\*T12\*T23\*base;

J4pos = T01\*T12\*T23\*T34\*base;

J5pos = T01\*T12\*T23\*T34\*T45\*base;

effpos = T01\*T12\*T23\*T34\*T45\*T5e\*base;

effL = T01\*T12\*T23\*T34\*T45\*T5e\*TeL\*base;

effR = T01\*T12\*T23\*T34\*T45\*T5e\*TeR\*base;

% joint position vectors

% create seperate x,y,z arrays for line plot

X = [base(1,4) J1pos(1,4) J2pos(1,4) J3pos(1,4) J4pos(1,4) J5pos(1,4) effpos(1,4) effL(1,4) effR(1,4)];

Y = [base(2,4) J1pos(2,4) J2pos(2,4) J3pos(2,4) J4pos(2,4) J5pos(2,4) effpos(2,4) effL(2,4) effR(2,4)];

Z = [base(3,4) J1pos(3,4) J2pos(3,4) J3pos(3,4) J4pos(3,4) J5pos(3,4) effpos(3,4) effL(3,4) effR(3,4)];

x5pos = L4\*(c4\*s1\*c23 - s4\*c1\*(c2\*c3 - s2\*c3)) + L3\*c1\*c23 + L2\*c1\*c2;

y5pos = L4\*(c4\*s1\*c23 - s4\*s1\*(c2\*s3 - s2\*c3)) + L3\*s1\*c23 - L2\*s1\*c2;

z5pos = L4\*(c4\*s23 - s4\*c23) + L3\*s23 + L2\*s2 +L1;

%% Plot lines connecting frames

figure (1)

set(1,'position',[200 300 400 400]) %Set figure position

plot3(PNX, PNY, PNZ, 'x-.');

hold on

plot3(X, Y, Z, '-o');

title('3d representation of Link positions') ; xlabel('x (mm)') ; ylabel('y (mm)'), zlabel ('z (mm)') ;

axis([-100 450 -100 450 -200 450])

%axis equal

text(0,0,0, 'base')

grid on;

pause(0.01);

hold off;

end

end

end