%SHV Problem 2.14
H0=eye(4);

%Perform Rotations
trans=[0;0;0];
H0_1 = hmc3_transrotX(trans,pi/2); %pi/2 rotation about X
H0_2 = hmc3_transrotY(trans,pi/2); %pi/2 rotation about Y

H = H0_2*H0_1; %Rotation realitive to fixed, note order
R = H(1:3,1:3) %Keep only rotation portion

subplot(1,3,1)
hmc3_showframe(H0, 1)
axis square
set(gca,'xlim',[-2 2],'ylim',[-2 2],'zlim',[-2 2])
title('x_0 y_0 z_0')
view(30,15)

subplot(1,3,2)
hmc3_showframe(H0_1*H0, 1)
axis square
set(gca,'xlim',[-2 2],'ylim',[-2 2],'zlim',[-2 2])
title('Rot \pi/2 About x_0')
view(30,15)

subplot(1,3,3)
hmc3_showframe(H*H0, 1)
axis square
set(gca,'xlim',[-2 2],'ylim',[-2 2],'zlim',[-2 2])
title('Rot \pi/2 About y_0')
view(30,15)

---

### Table 3. hmc3_transrotX Function for Performing Rotation about X Axis

```matlab
function T = hmc3_transrotX(translation, theta)
% Generates a homogeneous transformation matrix for a 3D translation followed
% by a rotation about the X-axis.
% %
% % Input:
% % translation     (3 x 1 matrix) translation vector
% % theta           (scalar) amount of rotation in radians (positive is screw towards +X)
% %
% % Output:
% % T             (4x4 matrix) homogeneous transformation matrix

T = [ 1 0 0 translation(1); ...
      0 cos(theta) -sin(theta) translation(2); ...
      0 sin(theta) cos(theta) translation(3); ...
      0 0 0 1 ];
```

End
TABLE 5. MATLAB SCRIPT FOR PROBLEM 2-22

```matlab
trans=[0;0;0];

syms theta phi

Rx_theta=hmc3_transrotX(trans,theta);
Ry_phi=hmc3_transrotY(trans,phi);
Rz_pi=hmc3_transrotZ(trans,sym(pi));
Ry_nphi=hmc3_transrotY(trans,-phi);
Rx_nttheta=hmc3_transrotX(trans,-theta);

R = Rx_theta * Ry_phi * Rz_pi * Ry_nphi * Rx_nttheta;

R_simp = simplify(R);
pretty(R_simp)
```

PROBLEM 3

Determine the slight difference between Matlab's atan2 and the function shown in Appendix A of SHV.

DISCUSSION

The Matlab code shown in Table 6 was used to compare the atan functions. As can be seen in Figure 2, the two algorithms return very similar results. There are two important exceptions to be aware of:

1) The SHV algorithm is undefined if x or y are 0 (depends on which version of the algorithm is implemented). MATLAB returns a value of 0.

2) The arguments for MATLAB are (y,x) whereas SHV is (x,y)

FIGURE 2. COMPARISON OF SHV AND MATLAB'S ATAN2 ALGORITHMS
PROBLEM 6

Solve SHV problem 3-13.

3-13 Solve the inverse position kinematics for the cylindrical manipulator of Figure 3.34.

![Figure 3.34: Cylindrical configuration.](image)

DISCUSSION

The results of inverse kinematics for the robot are shown in Table 7. The Matlab script for producing the results are shown in Table 8 and Table 9 a DH parameters function created for use by the script.

**Table 7. SHV Problem 3-13 Inverse Kinematic Results**

A =

\[
\begin{bmatrix}
1 & 0 & 0 & d_2 + d_3 + 2 \\
0, \cos(\theta_1), -\sin(\theta_1), (\pi \sin(\theta_1))/2 \\
0, \sin(\theta_1), \cos(\theta_1), -(\pi \cos(\theta_1))/2 \\
0, 0, 0, 1
\end{bmatrix}
\]

x =

d_2 + d_3 + 2

y =

(\pi \sin(\theta_1))/2

z =

-(\pi \cos(\theta_1))/2

**Table 8. Matlab Script for Solving SHV Problem 3-13**

```matlab
syms theta1 d1 d2 d3

A1=dhParameters(0,0,1,theta1);
A2=dhParameters(0,sym(-pi/2),d2,0);
A3=dhParameters(0,0,1+d3,0);

A=A1*A2*A3

x=A(1,4)
y=A(2,4)
z=A(3,4)
```

2/10/2015  
Homework Assignment #2
MAIN PROBLEM

Analyze the robot below for forward and inverse kinematics.

DISCUSSION

The attached MATLAB code was used to solve the problem shown above. Two parameters needed to be chosen:

\[
d_w = 1.0 \\
d_e = 0.1
\]

The attached code includes:

- Hw2_main_prob – this is the “main” function for the problem and works through each of the questions below. It includes a function for creating the Lemniscate and a function for smoothing the euler angle output (when calculating euler angles some cases have 2 possible euler angle sets; this function chooses the angle set closest to the current position of the robot).
- dhParameters – Function for creating a homogenous transformation matrix from DH parameters
- rot2eulerZYZ – function to determine euler ZYZ sequence from transformation matrix

I do not have the optimization toolbox (needed to use the ikunc method in the robotics toolbox). I therefore adapted the freely available IPOPT optimization solver to be used in place of fminunc; this function is called fminuncWithIPOPT. I then modified the Robotics toolbox to call my function. This produced good results but is computationally slow (~10 minutes to solve the inverse dynamics problem). The Robotics toolbox is using a numeric gradient though and could be speed up by symbolically solving which is possible with IPOPT.
4. Plot all joint variables as a function of figure parameter \( q \), for both halves of the figure.

Figure 4. Generalized Coordinates Determined by Inverse Kinematics

5. Verify that the inverse solution works by calculating the forward position kinematics and plotting in 3D.

The results of the inverse kinematics were used to drive forward kinematics and are shown below.
function hw2_main_prob

% Values not provided in the problem statement, so chosen to be:
w=1.0; % Distance from robot base to wall
d6=0.1; % End effector length

% Question 1 - Create Lemniscate and determine the world coordinates
thetaLem=[-pi/4:0.01:pi/4]';
[lemCoord,thetaOut]=lemn(thetaLem,dw);

% Question 2 - Determine the Required World Orientation of the end
% effector frame.

% Rotate 90deg (negative) about the x to align
TendEffector=hmc3_transrotX([0;0;0],sym(-pi/2));

% Get the rotation sub matrix
R0_6sym=TendEffector(1:3,1:3);
R0_6=eval(R0_6sym);

% Question 3 - Perform Inverse Kinematics

% Get the joint center of the spherical (SHV eq. 3.35)
endEffOffset=R0_6*(d6*[0;0;1]);
Pc_0(1,:) = lemCoord(1,:) - endEffOffset(1);
Pc_0(2,:) = lemCoord(2,:) - endEffOffset(2);
Pc_0(3,:) = lemCoord(3,:) - endEffOffset(3);

% Create DH Parameters for the robot (without wrist)

\[ n_1 = [q_1, 1, 0, 0] \]
\[ dh_2 = [0, q_2, 0, sym(-pi/2)] \]
\[ dh_3 = [0, q_3, 0, 0] \]
\[ A_1 = dhParameters(dh_1) \]
\[ A_2 = dhParameters(dh_2) \]
\[ A_3 = dhParameters(dh_3) \]
\[ T_0_3sym = A_1 \cdot A_2 \cdot A_3 \]

% Get the equations for the end effector center
display('The equation for xc = ');
pretty(T0_3sym(1,4))
display('The equation for yc = ');
pretty(T0_3sym(2,4))
display('The equation for zc = ');
pretty(T0_3sym(3,4))

% Create DH parameters for the wrist (Using the coordinate system from SHV
% Table 3.3
syms q4 q5 q6
dh_4 = [q_4, 0, 0, sym(-pi/2)];
dh_5 = [q_5, 0, 0, sym(pi/2)];
dh_6 = [q_6, d6, 0, 0];
A_4 = dhParameters(dh_4);
A_5 = dhParameters(dh_5);
A_6 = dhParameters(dh_6);
R_6sym = A_4 \cdot A_5 \cdot A_6;
% The entire robot DH

T0_6sym=T0_3sym*T1_6sym;

% Calculate the needed orientation for the end effector
R0_3sym = T0_3sym(1:3,1:3);
R3_6sym = R0_3sym.'*R0_6; %SHV eq 3.37

nSteps=size(Pc_0,2); %number of motion steps

% Performing inverse position kinematics
prevAngles=[1];
for i=1:nSteps %Looping through the end effector coordinates

%Get the position coordinates
xc = Pc_0(1,i);
yc = Pc_0(2,i);
zc = Pc_0(3,i);

%Perform the position inverse kinematics
%Calculate the genralized coordinates using the equations
q(3,i) = sqrt( xc^2 + yc^2 ); % only because xc and yc always >0
q(1,i) = -asin( xc / q(3,i) );
q(2,i) = zc-1;

%Perform the orientation inverse kinematics
R3_6= subs(R3_6sym(1:3,1:3),{'q1','q2','q3'},[q(1:3,i)]);
[eAngles,caseN]=rot2eulerZYX(R3_6);
eAngles=smoothEulerAngles(prevAngles,eAngles);
q(4,i)=eAngles(1,1);
q(5,i)=eAngles(1,2);
q(6,i)=eAngles(1,3);
prevAngles=eAngles;
end

% Question 5 - Perform Forward Kinematic to verify result

for a=1:nSteps

%Get the position by subsituting the generalized coordinates into
%the 4th column of the robots DH homogenous matrix
pSHV(:,a)=subs(T0_6sym(1:3,4),{'q1','q2','q3','q4','q5','q6'},q(:,a));
end

figure; plot3(pSHV(1,:),pSHV(2,:),pSHV(3,:))
xlabel('x');ylabel('y');zlabel('z');
set(gca,'xlim',[-1 1],'ylim',[-1 1],'zlim',[0 2])
figure('Forward Kinematics Verification')
view([-10,10])

% Question 6 -
% Create the Robot

Create Links
L(1) = Link('revolute', 'd', 1, 'a', 0, 'alpha', 0); %theta is q1
L(2) = Link('prismatic', 'theta', 0, 'a', 0, 'alpha', -pi/2); %d is q2
L(3) = Link('prismatic', 'theta', 0, 'a', 0, 'alpha', 0); %d is q3
L(4) = Link('revolute', 'd', 0, 'a', 0, 'alpha', -pi/2);
L(5) = Link('revolute', 'd', 0, 'a', 0, 'alpha', pi/2);
L(6) = Link('revolute', 'd', d6, 'a', 0, 'alpha', 0);

robot = SerialLink(L);

if 0
    %Perform Inverse Kinematics and save to file
    Ttarget=[RO_6,[0;0;0];0 0 0 1]; %Build target homogenous matrix
    qGuess=[0 0.9 0 0 0];
    for i=1:nSteps
        Ttarget(1:3,4) = lemCoord(1:3,i)
        qRbt(:,i)=robot.ikunc(Ttarget,qGuess);
        qGuess=qRbt(:,i)
    end
    save('Results2')
else
    load('Results2') %Load IK results (for plotting w/o rerunning IK)
end

% Perform Forward Kinematics to verify IK results
for i=1:size(qRbt,2)
    Trbs=robot.fkine(qRbt(:,i))
    pRbs(:,i)=Trbs(1:3,4)
end

plot3(lemCoord(1,:),lemCoord(2,:),lemCoord(3,:), 'color','g', 'linewidth',3) %hold on
plot3(pSHV(1,:),pSHV(2,:),pSHV(3,:), 'color','b', 'linewidth',2,...
    'linestyle','--')
plot3(pRbs(1,:),pRbs(2,:),pRbs(3,:), 'color','r', 'linewidth',2,...
    'linestyle','-) xaxis('x'); ylabel('y'); zlabel('z');

errorSHV=sqrt(sum((pSHV-lemCoord).^2,1));
errorMeanSHV=mean(errorSHV);
errorStdSHV=std(errorSHV);

errorRbs=sqrt(sum((pRbs-lemCoord).^2,1));
errorMeanRbs=mean(errorRbs);
errorStdRbs=std(errorRbs);

% Question 4 - Plot Generalized Coordinates
xValues=[1:nSteps]/nSteps;
figure
subplot(6,1,1)
plot(xValues,q(1,:),'linewidth',2)
hold on
plot(xValues,qRbt(1,:), 'color','t', 'LineStyle','--', 'linewidth',2)
set(gca,'xtick',[0:0.25:1], 'xticklabel',{['- pi/4','0','pi/4','0','pi/4'])
legend('SHV Methods', 'Robotic Toolbox')
% Question 5 - Plot Inverse Solution

figure
plot3(lemCoord(1,:),lemCoord(2,:),lemCoord(3,:), 'color', 'g', 'linewidth', 3)
hold on
plot3(pSHV(1,:), pSHV(2,:), pSHV(3,:), 'color', 'b', 'linewidth', 2,
      'linestyle', '-.')
plot3(pRbs(1,:), pRbs(2,:), pRbs(3,:), 'color', 'r', 'linewidth', 2,
      'linestyle', '-.')
xlabel('x'); ylabel('y'); zlabel('z');
set(gca, 'xlim', [-1 1], 'ylim', [-1 1], 'zlim', [0 2])
title('Forward Kinematics Verification')
view([-10,10])
legend('Command', 'SHV Methoda', 'Robotic Toolbox')

function [lemCoord,thetaOut]=lemn(theta,dw)
% Function to make Bernoulli's lemniscate
r=sqrt(cos(2*theta));
x=r.*sin(theta);
z=r.*cos(theta);
% Make the mirror image (could have also done this with a mirror projection
% but flip and using the negative does the same.
x=x;flipud(-x(1:end-1));
z=z;flipud(-z(1:end-1)) + 1;  % +1 is for vertical (z) offset
y=ones(size(z,1),1).*dw;

% Create Outputs
lemCoord=[x,y,z];  % Each Column is a vector
thetaOut=[theta;flipud(theta(1:end-1))];

%% Plot Lemniscate
figure
subplot(2,1,1)
plot(x,z)
axis equal
axis square
xlabel('x')
ylabel('z')

nPoints=max(size(x));
xValues=[1:nPoints]./nPoints;

subplot(2,1,2)
plot(xValues,x,'b','displayname','x_0')
hold on
plot(xValues,y,'g','displayname','y_0')
plot(xValues,z,'r','displayname','z_0')
set(gca,'xtick',[0:0.25:1], 'xticklabel', ['-pi/4','0','pi/4','0','pi/4'])
legend show
xlabel('	heta (rad)')
ylabel('End Eff. Coord')

function eAngles=smoothEulerAngles(prevAngles,eAngles)
% Function to select euler angle based on minimizing jumps between succesive
% steps. This is needed because there are multiple possible solution of the
% euler angles. This function looks at the previous and minimizes the
% "jump".

if isempty(prevAngles);
    eAngles=eAngles(1,:);
else
    nAngles=size(eAngles,1);
    if nAngles>1
        for i=1:nAngles
            s(i)=sum((eAngles(i,:)-prevAngles).^2);
        end
        [y,idx]=min(s);
        eAngles=eAngles(idx,:);
    else
        eAngles=eAngles(1,:);
    end
end
function [eAngles,caseN]=rot2eulerZYZ(R,tol)

% rot2eulerZYZ - Extract ZYZ Euler angles for a rotation matrix
%
% [eAngles,caseN]=rot2eulerZYZ(R,tol)
% Inputs:
%    R - rotation matrix
%    tol - tolerance used when determining if a matrix entry
%           is 0 (optional argument, default is 1e-6)
% Outputs:
%    eAngles - row vector of euler angles ZYZ (phi, theta, psi)
%    caseN -
% Note that because several solutions are possible, this function may
% return two rows
%
% Reference:
% Robot Modeling and Control 2006 - Spong, Hutchinson, Vidyasagar
% (eq 2.26 through 2.37)

if nargin==1
    tol=1e-6;
end

13=R(1,3);
r23=R(2,3);
r33=R(3,3);
r31=R(3,1);
r32=R(3,2);

if abs(r13)>tol | abs(r23)>0  % checking see if r13 and r23 do not equal 0
    thetaP = atan2( sqrt(1-r33^2) , r33);
    phiP = atan2(r23,r13);
    psiP = atan2(r32,-r31);

    thetaM = atan2( -sqrt(1-r33^2) , r33);
    phiM = atan2(-r23,-r13);
    psiM = atan2(-r32,r31);

    caseN=1;

    eAngles=[phiP,thetaP,psiP;...
             phiM,thetaM,psiM];
else
    theta=0;
    r11=R(1,1);
    r12=R(1,2);
    if r33>0  % checking to if r33=1
        psi=0;
        phi=atan2(-r12,r11);
    else
        caseN=2;
        eAngles=[phi,theta,psi;...
                 phi,theta,psi];
    end
end
caseN=2;
else  r33<0   %checking to if r33=-1
    psi=0;
    phi=atan2(-r12,-r11);
    caseN=3;
end
eAngles=[phi,theta,psi];
end
function A=dhParameters(theta,d,a,alpha)

% dhParameters - Create a homogeneous transformation from DH parameters

% A=dhParameters(theta,d,a,alpha) or A=dhParameters([theta,d,a,alpha])
% Inputs:
% theta,d,a,alpha - DH parameters
% Outputs:
% A - homogeneous transformation matrix

%Reference:
% Robot Modeling and Control 2006 - Spong, Hutchinson, Vidyasagar (eq 3.10)
% Argument order follows arguments from Corke's robotic toolbox

if nargin==1
    d=theta(2);
    a=theta(3);
    alpha=theta(4);
    theta=theta(1);
end

ct=cos(theta);
st=sin(theta);
ca=cos(alpha);
sa=sin(alpha);

A = [  ct   -st*ca  st*sa  a*ct;     
       st   ct*ca  -ct*sa  a*st;     
       0     sa     ca     d;       
       0     0      0      1];
function [xmin,fval,exitflag] =fminuncWithIPOPT(problem)

% fminuncWithIPOPT - Function to use in place of MATLAB's optimization
% toolbox fminunc function. IPOPT is used instead of MATLAB's algorithm.
% %
% % [xmin,fval,exitflag] = fminuncWithIPOPT(problem)
% %
% See fminunc for argument descriptions.
%
% Example Problem:
% a=1;
% b=0;
% problem.objective = @(x) (sum(a.*x.^2)+b);
% problem.x0=0;
% x = fminuncWithIPOPT(problem);
%
% To do:
% Currently this is set up to handle numeric functions. Need to handle as
% analytic/symbolic functions. To do this, in gradObjFun, determine if the
% function is symbolic and if so, use the symbolic toolbox to calculate
% analytical derivatives instead of using gradest.
%
% The Options
options.ipt.opt.jac_c_constant = 'yes';
options.ipt.opt.hessian_approximation = 'limited-memory';
options.ipt.opt.mu_strategy = 'adaptive';
options.ipt.opt.tol = 1e-7;
options.ipt.opt.print_level = 0;

fun = problem.objective;

% The callback functions.
funcs.objective = @(x) objFun(x,fun);
funcs.gradient = @(x) gradObjFun(x,fun);

[xmin info] = ipopt(problem.x0,funcs,options);
exitflag=1;

if nargout > 1
    fval=funcs.objective(xmin);
end

function obj=objFun(x,fun)
obj = fun(x);

function grad=gradObjFun(x,fun)
grad = gradest(fun,x);
SerialLink.IKUNC Numerical inverse manipulator without joint limits

Q = R.ikunc(T) are the joint coordinates (1xN) corresponding to the robot
end-effector pose T (4x4) which is a homogenous transform, and N is the
number of robot joints.

[Q, ERR] = robot.ikunc(T) as above but also returns ERR which is the
scalar final value of the objective function.

[Q, ERR, EXITFLAG] = robot.ikunc(T) as above but also returns the
status EXITFLAG from fminunc.

[Q, ERR, EXITFLAG] = robot.ikunc(T, Q0) as above but specify the
initial joint coordinates Q0 used for the minimisation.

[Q, ERR, EXITFLAG] = robot.ikunc(T, Q0, options) as above but specify the
options for fminunc to use.

Trajectory operation:

In all cases if T is 4x4xM it is taken as a homogeneous transform
sequence and R.ikunc() returns the joint coordinates corresponding to
each of the transforms in the sequence. Q is MxN where N is the number
of robot joints. The initial estimate of Q for each time step is taken as
the solution from the previous time step.

ERR and EXITFLAG are also Mx1 and indicate the results of optimisation
for the corresponding trajectory step.

Notes:
- Requires fminunc from the Optimization Toolbox.
- Joint limits are not considered in this solution.
- Can be used for robots with arbitrary degrees of freedom.
- In the case of multiple feasible solutions, the solution returned
depends on the initial choice of Q0.
- Works by minimizing the error between the forward kinematics of the
joint angle solution and the end-effector frame as an optimisation.
The objective function (error) is described as:

  sum(sqr((inv(T)*robot.fkine(q) - eye(4)) * omega))

Where omega is some gain matrix, currently not modifiable.

Author:
Bryan Moutrie

See also SerialLink.ikcon, fmincon, SerialLink.ikine, SerialLink.fkine.

Copyright (C) Bryan Moutrie, 2013-2015
Licensed under the GNU Lesser General Public License
see full file for full statement

LICENSE STATEMENT:

This file is part of pHRWARE.

pHRWARE is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as
published by the Free Software Foundation, either version 3 of
the License, or (at your option) any later version.
function [qstar, error, exitflag] = ikunc(robot, T, q0, options)

% check if Optimization Toolbox exists, we need it
if ~exist('fminunc')
    error('rtb:ikunc:nosupport', 'Optimization Toolbox required');
end

% create output variables
T_sz = size(T,1);
qstar = zeros(T_sz,robot.n);
error = zeros(T_sz,1);
exitflag = zeros(T_sz,1);

problem.solver = 'fminunc';
problem.x0 = zeros(1, robot.n);
% problem.options = optimoptions('fminunc', ...
% 'Algorithm', 'quasi-newton', ...
% 'Display', 'off'); % default options for ikunc

if nargin > 2
    problem.x0 = q0;
end
% if nargin > 3
% problem.options = optimset(problem.options, options);
%
reach = sum(abs([robot.a, robot.d]));
omega = diag([1 1 1 3/reach]);

for t = 1:T_sz
    problem.objective = ...
    @(x) sum(sqr((T(:,t) \ robot.fkine(x)) - eye(4)) * omega);
%
[q_t, err_t, ef_t] = fminunc(problem);
[q_t, err_t, ef_t] = fminuncWithIPOPT(problem);

qstar(t,:) = q_t;
error(t) = err_t;
exitflag(t) = ef_t;

problem.x0 = q_t;
end

end

function s = sum(sqr(A))
    s = sum(A(:,).^2);
5. Main Problem (Doctoral)
Repeat the problem, but consider that the figure must be drawn on the plane
\[ \frac{x_0}{2} + \frac{y_0}{2} + \frac{z_0}{2} = 1 \]
The proportions of the figure must be the same as in the original case. Chose your own center for the figure.

Solution:

5.1 Manual Calculation
The MATLAB code and steps used for the rotated plane are mostly the same as the previous case. However, drawing the pattern on the rotated plane requires additional efforts to rotate the leminiscate onto the new plane. This necessary rotation can be determined using Rodrigues' Formula (code available in Appendix D), where \( K \) is the cross-product matrix.

\[ R = I + (\sin \theta)K + (1 - \cos \theta)K^2 \]  \hspace{1cm} (5.1.1)

\( K \) can be obtained by finding the angle \( \theta \) between two rotations. For instance, in this problem, we know that we want to the z-axis [0 0 1] on a vector normal to the new plane norm
\[ \frac{1}{\sqrt{3}} [1 \hspace{0.5cm} 1 \hspace{0.5cm} 1] \]. \( \theta \) can be obtained using the dot product formula:

\[ \theta = \cos^{-1}(z \cdot v) \]  \hspace{1cm} (5.1.2)

A new vector \( s \) can then be obtained by taking the cross product of vectors \( z \) and \( v \), the resulting row vector becoming the values \( k_1, k_2, \) and \( k_3 \) that comprise the \( K \) matrix:

\[ K = \begin{bmatrix} 0 & -k_3 & k_2 \\ k_3 & 0 & -k_1 \\ -k_2 & k_1 & 0 \end{bmatrix} \]  \hspace{1cm} (5.1.3)

\( K \) and \( \theta \) can then be substituted into (5.1.1) in order to obtain the rotation matrix. After applying \( R \) to the leminiscate, it must then be translated to the appropriate center. Lastly, the end effector must be reconsidered, with \( z \) being normal to the new plane. The \( x \) axes was determined by arbitrarily assigning a perpendicular vector to \( z \), so that the dot product would be zero. Lastly, the cross product of \( z \) and \( x \) will create the \( y \) orientation of the end effector. No other aspects of the MATLAB code were changed.

The calculated parameters \( q \) are shown in Figure 6. Though the results are less intuitive, it is clear that the robot no longer produces a symmetrical movement. There is now substantial
motion in \( q_1 \) as the robot must move forward slightly to follow the plane. Lastly, there is more noticeable wrist motion in \( q_4 \) and \( q_6 \), possibly suggesting that the wrist may use these orientations to avoid singularities. However, the constant zero angle in \( q_3 \) is somewhat surprising.

![Figure 6. Joint variables as a function of \( \theta \) from manually performing inverse kinematics](image)

Additionally, the verification of the robot's movements on the rotated plane are shown in Figure 7.

![Figure 5. Commanded (black) and actual (red) coordinates for the end effector on the XY plane shown from an isometric perspective (left) and on the YZ plane (right)](image)
5.2 Corke's Toolbox
Similarly, the only variation in the code for using the Robotics Toolbox was the rotation of the leminiscate using Rodrigues' formula. The resulting parameters and motion of the robot are shown in Figure 7 and Figure 8. Surprisingly, the toolbox is able to produce the correct motion for the rotated plane, though it had difficulty for the XZ plane (see Figure 4). The only explanation is that an older version of the Robotics Toolbox was used in order to ensure compatibility with MATLAB. The older toolbox may use different methods for solving inverse kinematics that happened to work more efficiently with the coordinates in the rotated plane.

Figure 7. Joint variables as a function of \( \theta \) using Corke's Robotics Toolbox to solve the inverse kinematics

Figure 7. Commanded (black) and actual (red) coordinates for the end effector on the XY plane shown from an isometric perspective (left) and on the YZ plane (right)
Appendix A: MATLAB Session Problem 2-22

>> syms theta phi
R1 = [ 1 0 0; 0 cos(theta) -sin(theta); 0 sin(theta) cos(theta)];
R2 = [cos(phi) 0 sin(phi); 0 1 0; -sin(phi) 0 cos(phi)];
R3 = [-1 0 0; 0 -1 0; 0 0 1];
R4 = [cos(-phi) 0 sin(-phi); 0 1 0; -sin(-phi) 0 cos(-phi)];
R5 = [1 0 0; 0 cos(-theta) -sin(-theta); 0 sin(-theta) cos(-theta)];
R = simplify(R1*R2*R3*R4*R5)
simplify(det(R))
simple(R.*R)

R =

[ 1 - 2*cos(phi)^2, -2*cos(phi)*sin(phi)*sin(theta),
  sin(2*phi)*cos(theta)]
[ -2*cos(phi)*sin(phi)*sin(theta), -2*cos(phi)^2*cos(theta)^2 + 2*cos(phi)^2 - 1,
  sin(2*phi)*cos(phi)^2]
[ sin(2*phi)*cos(theta), -sin(2*theta)*cos(phi)^2,
  2*cos(phi)^2*cos(theta)^2 - 1]

ans =

1

simplify:

[ 1, 0, 0]
[ 0, 1, 0]
[ 0, 0, 1]
clear
clc

display('Beginning Computation...')
%--------------------------------------------------------
%Part 1. Generate the Figure 8
%--------------------------------------------------------
display('Generating Figure 8...')
theta = -pi/4:0.01:pi/4;
fig8_top = [];
fig8_bottom = [];
for i=1:length(theta);
%Define Shape in Polar Form
r = sqrt(cos(2*theta(:,i)));
x = r*sin(theta(:,i));
y = r*cos(theta(:,i));
z = zeros(1,length(x));
fig8 = [x; y; z];
%Rotate and Translate to Desired Configuration
dw=2;
    center = [0;dw;1];
    fig8_trot(:,1) = transrotx(pi/2)*[fig8(:,1); 1];
    fig8_trot(:,2) = transrotx(pi/2)*[fig8(:,2); 1];
%For Rotated Plane
    %fig8_rot = rodrigues([0 0 1],[1 1 1]);
    %fig8_rot = [fig8_rot zeros(3,1); 0 0 0 1];
    %fig8_trot(:,1) = fig8_rot*trans(center)*[fig8(:,1); 1];
    %fig8_trot(:,2) = fig8_rot*trans(center)*[fig8(:,2); 1];
%Store Top and Bottom Halves of the Figure 8
    fig8_top = [fig8_top; fig8_trot(1:3,1)];
    fig8_bottom = [fig8_bottom; fig8_trot(1:3,2)];
end
%Combine Halves to Create Final Figure 8
fig8_final=[fig8_top(2:end,:); flipud(fig8_bottom)];
%--------------------------------------------------------
%Part 2. World Orientation of End Effector
%--------------------------------------------------------
display('Determining End Effector Orientation...')
R = rotx(-pi/2);
%For Rotated Plane
Rz = [-2;2;2];
Rx = [-2;1;1];
Ry = cross(Rx,Rz);
R = [Rx Ry Rz];
%--------------------------------------------------------
%Part 3. Inverse Kinematics
%--------------------------------------------------------
display('Performing Inverse Kinematics...')
%--------------------------------------------------------
%Creating the Robot
%--------------------------------------------------------
syms q1 q2 q3 q4 q5 q6 pi
%Create Three-Link Cylindrical Manipulator
d1 = 1;
d6 = 1;
A1 = transrotz(q1,[0;0;d1]);
A2 = transrotx(-pi/2, [0;0;q2]);
A3 = trans([0; 0; q3]);
T30 = A1*A2*A3;
%Create Spherical Wrist
A4 = transrotz(q4,[0;0;0])*transrotx(-pi/2,[0;0;0]);
A5 = transrotz(q5,[0;0;0])*transrotx(pi/2,[0;0;0]);
A6 = transrotz(q6,[0;0;0]);
T63 = A4*A5*A6;
%Combine
T60 = T30*T63;
%Calculate Joint Angles
%-----------------------------------
q=zeros(length(fig8_final),6);
for i=1:length(fig8_final)
    q(i,1) = atan2(fig8_final(i,2), fig8_final(i,1));
    q(i,2) = fig8_final(i,3)-1;
    q(i,3) = sqrt(fig8_final(i,1)^2 + fig8_final(i,2)^2)-d6;
    R30 = subs(T30([1,3,1,3],'.',[q1, q2, q3], q(i,1:3))*R;
    q(i,4) = atan2(R30(2,3),R30(1,3));
    q(i,5) = atan2(sqrt(1-R30(3,3);),R30(3,3));
    q(i,6) = atan2(R30(3,2),-R30(3,1));
end
%Part 4. Plot Joint Variables
%-----------------------------------
figure(2)
for i = 1:size(q,2)
    subplot(3,2,i)
    plot([theta(2:end) fliplr(theta)],q(:,i))
    title(titles{i})
    xlabel('Theta (rad)')
    ylabel(ylabels{i})
end
%Part 5. Verify
%-----------------------------------
display('Performing Forward Position Kinematics...')
figure(1)
plot3(fig8_final(:,1), fig8_final(:,2), fig8_final(:,3),'k*')
hold on
for i = 1:length(q);
    T_final = subs(T60,[q1, q2, q3, q4, q5, q6], q(i,1:6));
    coordinates = T_final([1,3,4]);
    plot3(coordinates(1), coordinates(2), coordinates(3),'ro')
    xlabel('x'); ylabel('y'); zlabel('z')
end
legend('Commanded', 'Actual')
title('Manual Calculation, Doctoral', 'FontWeight', 'bold')
display('Computation Completed.')
Appendix C: Main Problem MATLAB Code, Corke's Toolbox

clear
clc

display('Beginning Computation...')
%---------------------------------------------------------------
%Generate Figure 8
%---------------------------------------------------------------
display('Generating Figure 8...')
theta = -pi/4:0.01:pi/4;
fig8_top = [];
fig8_bottom = [];
for i=1:length(theta);
    % Define Shape in Polar Form
    r = sqrt(cos(2*theta(:,i)));
    x = [r*sin(theta(:,i)), -r*sin(theta(:,i))];
    y = [r*cos(theta(:,i)), -r*cos(theta(:,i))];
    z = zeros(1,length(x));
    fig8 = [x; y; z];
    % Rotate and Translate to Desired Configuration
    dw=2;
    center = [0;dw;1];
    fig8_trot(:,1) = transrotx(pi/2)*[fig8(:,1); 1];
    fig8_trot(:,2) = transrotx(pi/2)*[fig8(:,2); 1];
    % For Rotated Plane
    % fig8_rot = rodrigues([0 0 1],[1 1 1]);
    % fig8_rot = [fig8_rot zeros(3,1); 0 0 0 1];
    % fig8_trot(:,1) = fig8_rot*trans(center)*[fig8(:,1); 1];
    % fig8_trot(:,2) = fig8_rot*trans(center)*[fig8(:,2); 1];
    % Store Top and Bottom Halves of the Figure 8
    fig8_top = [fig8_top; fig8_trot(1:3,1)];
    fig8_bottom = [fig8_bottom; fig8_trot(1:3,2)];
end
% Combine Halves to Create Final Figure 8
fig8_final = [fig8_top(2:end,:); flipud(fig8_bottom)];
%---------------------------------------------------------------
% Compare with Corke's Robotics Toolbox
%---------------------------------------------------------------
display('Using Toolbox...')
% Create the Robot
d1=1;
d6=1;
robot(1) = Link([0,d1,0,0,0]);
robot(2) = Link([0,0,0,-pi/2,1]);
robot(3) = Link([0,0,0,0,1]);
robot(4) = Link([0,0,0,-pi/2,0]);
robot(5) = Link([0,0,0,pi/2,0]);
robot(6) = Link([0,d6,0,0,0]);
AwesomeBot = SerialLink(robot);
% Perform Inverse Kinematics
qguess = [1.5 0 0 0 0 0];
for i=1:length(fig8_final),
    H = [eye(3) [fig8_final(i,1);fig8_final(i,2);fig8_final(i,3)];0 0 0 1];
    q(:,i) = AwesomeBot.ikine(H,qguess,[1 1 1 1 1 1]);
    qguess = q(:,i);
end
% Verification
figure(1)
poly3(fig8_final(:,1), fig8_final(:,2), fig8_final(:,3),'k--')
hold on
for i=1:length(fig8_final),
H=AwesomeBot.fkine(q(:,i));
check=H(1:3,4);
plot3(check(1),check(2),check(3),'ro')
xlabel('x'); ylabel('y'); zlabel('z');
legend('Commanded','Actual')
title('Robotics Toolbox, Doctoral Problem','Fontweight','bold')
end

%------------------------------------------------------------------
%Compare q's
%------------------------------------------------------------------
display('Generating Figures...')
titles = {'q1','q2','q3','q4','q5','q6'};
ylabels = {'Rad','m','m','Rad','Rad','Rad'};
figure(2)
for i = 1:size(q,1)
 subplot(3,2,i)
 plot([theta(2:end) fliplr(theta)],q(i,:))
title(titles{i})
xlabel('Theta (rad)')
ylabel(ylabels{i})
end
display('Computation Completed.')
Appendix D: Rodrigues, Rotation, and Translation Functions

function R = rodrigues(z,v)
    v = v/norm(v);
    theta = acos(dot(z,v));
    s = cross(z,v);
    k1 = s(1,1);
    k2 = s(1,2);
    k3 = s(1,3);
    K = [ 0 -k3 k2; k3 0 -k1; -k2 k1 0 ];
    R = eye(3) + sin(theta)*K + (1-cos(theta))*K^2;
end

function R=rotx(angle)
    R=[1 0 0; 0 cos(angle) -sin(angle); 0 sin(angle) cos(angle)];
end

function R=roty(angle)
    R=[cos(angle) 0 sin(angle); 0 1 0; -sin(angle) 0 cos(angle)];
end

function R=rotz(angle)
    R=[cos(angle) -sin(angle) 0; sin(angle) cos(angle) 0; 0 0 1];
end

function T=trans(disp)
    T=[eye(3) disp; 0 0 0 1];
end

function T=transrotx(angle,disp)
    T=[rotx(angle) disp; 0 0 0 1];
end

function T=transroty(angle,disp)
    T=[roty(angle) disp; 0 0 0 1];
end

function T=transrotz(angle,disp)
    T=[rotz(angle) disp; 0 0 0 1];
end