Table of Contents

This is the code for the RBE 595 Parallel Walking Robotics Midterm

```
function Midterm

% The nominal parameters of the robot
simuParam = [92.1597 84.4488 0 305.4001 111.1565 0 604.8652;
    27.055 122.037 0 -56.4357 320.0625 0 604.8652;
    -119.2146 37.5882 0 -248.9644 208.9060 0 604.8652;
    -119.2146 -37.5882 0 -248.9644 -208.9060 0 604.8652;
    27.055 -122.037 0 -56.4357 -320.0625 0 604.8652;
    92.1597 -84.4488 0 305.4001 -111.1565 0 604.8652];

lmax = 1100;
```

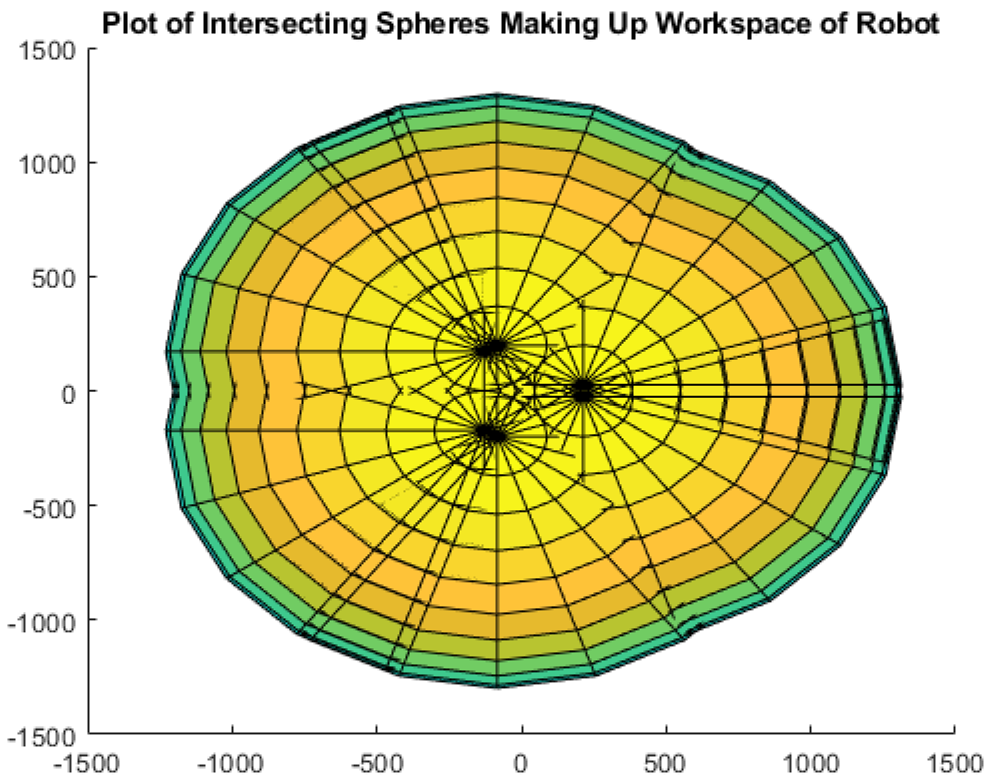
Illustrating the general workspace

We know that U is the ground Origin to the leg and S is upper platform origin to the leg. Since the robot is completely level that means that the centers of the circles are going to be in the same plane as U, i.e flat. Normally we move the center of the circle b/2 inwards, But in this case b/2 is equal to S due to the robot being level and all angles being 0. Therefore, we find that the centers of the circles are at U - S.

```
% First we extract U and S from table 2
si = simuParam(:,1:3)';
ui = simuParam(:,4:6)';
simuLen = simuParam(:,7);

% U - S
circleCenters = ui-si;
[m,n] = size(circleCenters);
% make the sphere
[x,y,z] = sphere;
% set the radius as the maximum leg length
r = lmax;
```

This plots all of the intersecting spheres.

```
figure(1)
hold on
for i=1:n
    surf(x*r+circleCenters(1,i),y*r+circleCenters(2,i),z*r
+circleCenters(3,i))
end
title('Plot of Intersecting Spheres Making Up Workspace of Robot')
hold off
```

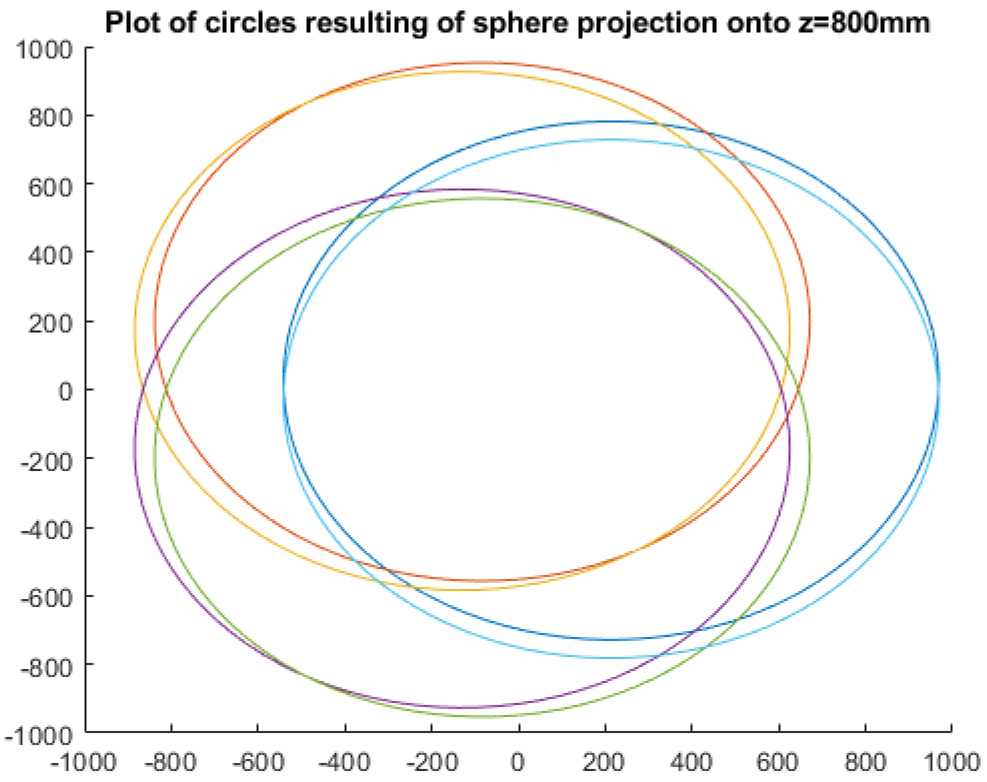


Plotting the spheres at 800mm

We are looking for the horizontal slice of the sphere generated at 800mm. The sphere equation is R^2 = x^2+y^2+z^2. Since z is the only thing that is changing, We can therefore rewrite the equation as x^2+y^2 = R^2-z^2. This way, we can solve for the radius of the circle at a specified z as sqrt(R^2-Z^2). In this case, we know that R is at it's max at 1100mm and z is set to 800mm.

```
circleRadius = sqrt(lmax^2-800^2);

% We now have the projected circles onto zplane of 800mm
% Since Z is always going to be zero from now on in this local frame,
% We can cut it out for ease of calculations

circleCenters = circleCenters(1:2,:);
```

```
% This is the basic code for drawing a circle in matlab, we can use it
 to
% draw our circles
% t = linspace(0,2*pi);plot(cos(t),sin(t))

t = linspace(0,2*pi);
figure(2)
hold on
% Plot the circles of the spheres that make up the workspace at z =
 800mm
for i=1:n
    plot(circleCenters(1,i) +
 circleRadius*(cos(t)),circleCenters(2,i)+ circleRadius*sin(t))
end
title('Plot of circles resulting of sphere projection onto z=800mm')
hold off
```



Finding Intersections

This depicts the workspace very well, it can clearly be seen that it is the white space on the inside that all the circles contain. However, to find the boundaries of the workspace, we need to determine where the intersections of all of the circles happen.

```
% calculate points of intersection
intersections = [];
```

```
for i=1:n-1
    %     find equations of the circles,
    %     intersection is where the two equations equal each other
    %     (x-h)^2 + (y-l)^2 = r^2
    syms x y;
    c1 = (x-circleCenters(1,i))^2+(y-circleCenters(2,i))^2 ==
 circleRadius^2;
    c2 = (x-circleCenters(1,i+1))^2+(y-circleCenters(2,i+1))^2 ==
 circleRadius^2;
    [interX,interY] = solve([c1,c2],[x,y]);

    %     This makes the output readable
    simpleInterX = double(interX);
    simpleInterY = double(interY);
    %     Append found intersections to matrix
    intersections = vertcat(intersections,
[simpleInterX,simpleInterY]);

    % This should give us all the different intersections of the
 circles
    % 2 per comparison for a total of 10 intersections
end

% Find the intersection of first circle to last circle
c1 = (x-circleCenters(1,1))^2+(y-circleCenters(2,1))^2 ==
 circleRadius^2;
c2 = (x-circleCenters(1,6))^2+(y-circleCenters(2,6))^2 ==
 circleRadius^2;
[interX,interY] = solve([c1,c2],[x,y]);
simpleInterX = double(interX);
simpleInterY = double(interY);
intersections = vertcat(intersections,[simpleInterX,simpleInterY]);

% This gives us our last intersection, for a total of 12
```

Determining relevant intersections

Now, we have a list of intersections, but we need to find all of the intersections that are contained within all of the circles. We can determine if a point is within a circle by checking to see if (x-circle_x)^2+(y-circle_y)^2<=R^2. So by definition, if an intersection is within all of the circles, then it is a part of the workspace.

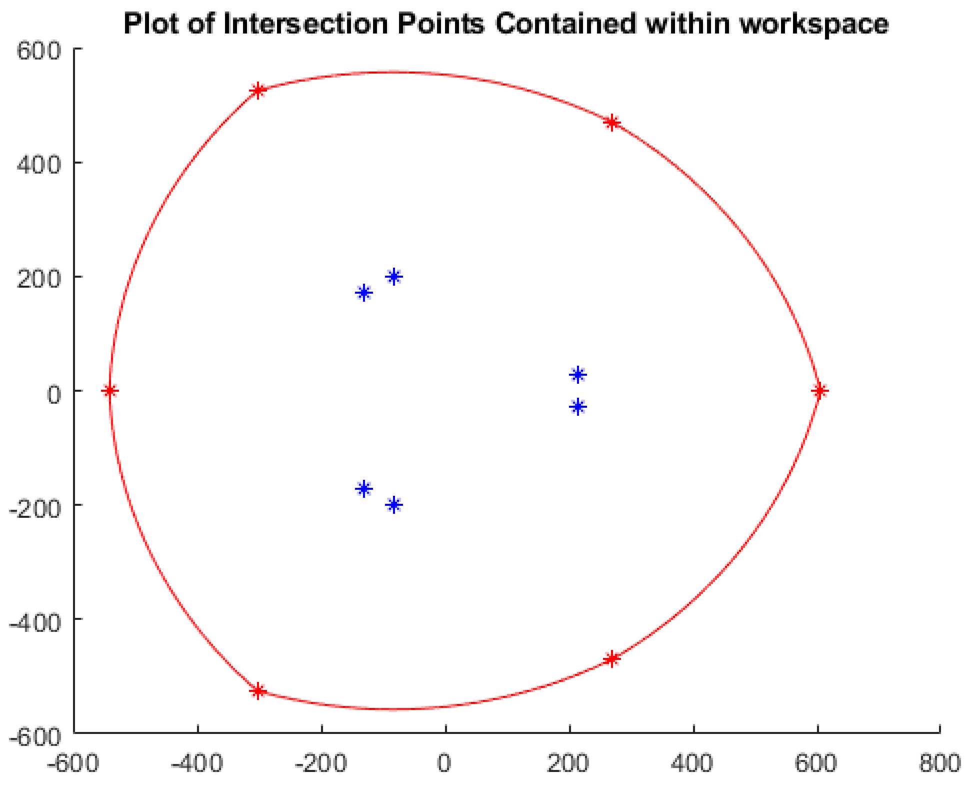
```
[o,p] = size(intersections);
updateInter = [];
for j = 1:o
    %     for every single intersection point
    count = 1;
    for i = 1:n
        %     determine if point is within circle i
        circDist = (intersections(j,1)-
circleCenters(1,i))^2+(intersections(j,2)-circleCenters(2,i))^2;
        if (circDist <= circleRadius^2)
            %         If it is, increase count
```

```
                count = count+1;
            end
        end
        %If point is within all of the circles add it to the updated list
        if (count >= 6)
            updateInter = vertcat(updateInter,
[intersections(j,1),intersections(j,2)]);
        end
end

% This gives us a list of all the intersections within all of the
% circles. These are the points that make up the workspace.
```

Drawing the Workspace

To properly depict the workspace, I want to draw arcs between the two points. I have the center of the circle, the radius of the circles, and the two points I want to draw an arc between. This is basic trigonometry to solve for the segment of the circle that I want.

```
figure(3)
hold on;
for i=1:size(updateInter)-1
    % intersection point 1
    P1 = updateInter(i,:)';
    % intersection point 2
    P2 = updateInter(i+1,:)';
    R = circleRadius;
    % For some reason it's the i+1 circle that gives us the correct
 workspace
    C = circleCenters(:,i+1);

    % Create the arc
    [x,y] = createArc(P1,P2,R,C);

    % Plot it
    plot(updateInter(i,1),updateInter(i,2),'r*')
    plot(x,y,'r-',C(1),C(2),'b*')
end
title('Plot of Intersection Points Contained within workspace')
% Do same thing with 1st intersection vs last, otherwise, workspace
 will
% not be complete
P1 = updateInter(6,:)';
P2 = updateInter(1,:)';
R = circleRadius;
C = circleCenters(:,1);
[x,y] = createArc(P1,P2,R,C);
plot(updateInter(6,1),updateInter(6,2),'r*')
plot(x,y,'r-',C(1),C(2),'b*')
```

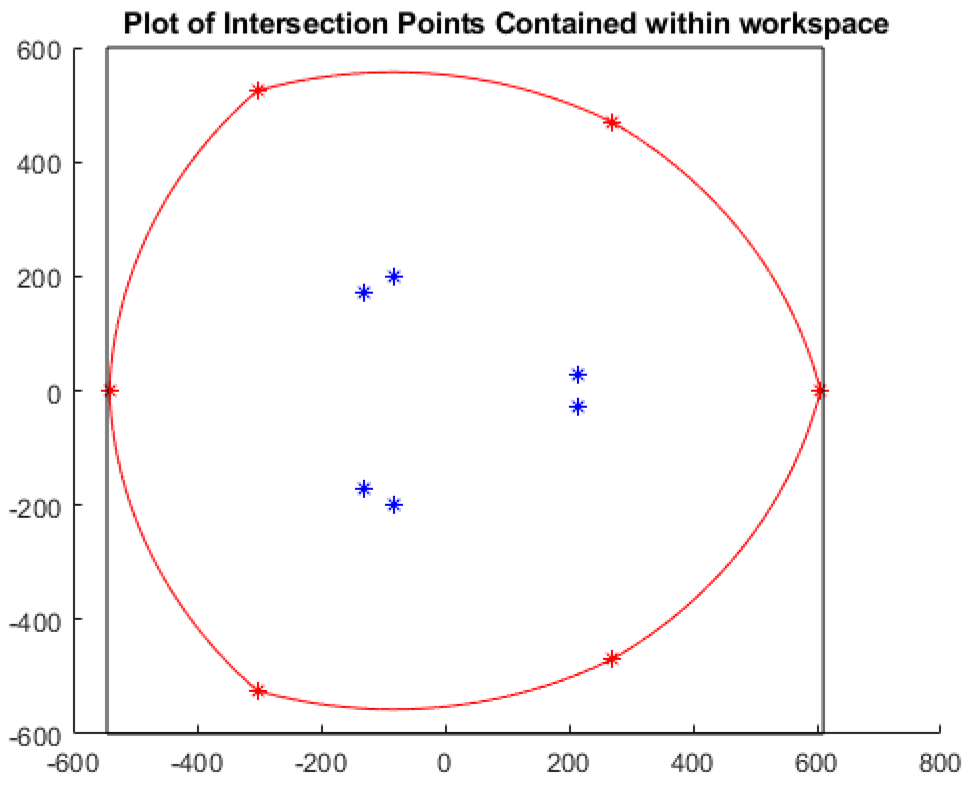


Dividing the workspace

Now, we need to divide the workspace into 5mm increments. This should result in around 40k points according to the discussion board. We already have the code needed for determining if a point is within the workspace as shown in determining if the intersection was within all of the circles. Now, we just need to create a rectangle of all the points from the max values of the workspace, i.e, the max intersection points. This rectangle should encompass the entire workspace. Once we have made the rectangle, it is a simple matter of going through all of the points in the rectangle at 5mm increments, and removing the points that are not inside of the workspace.

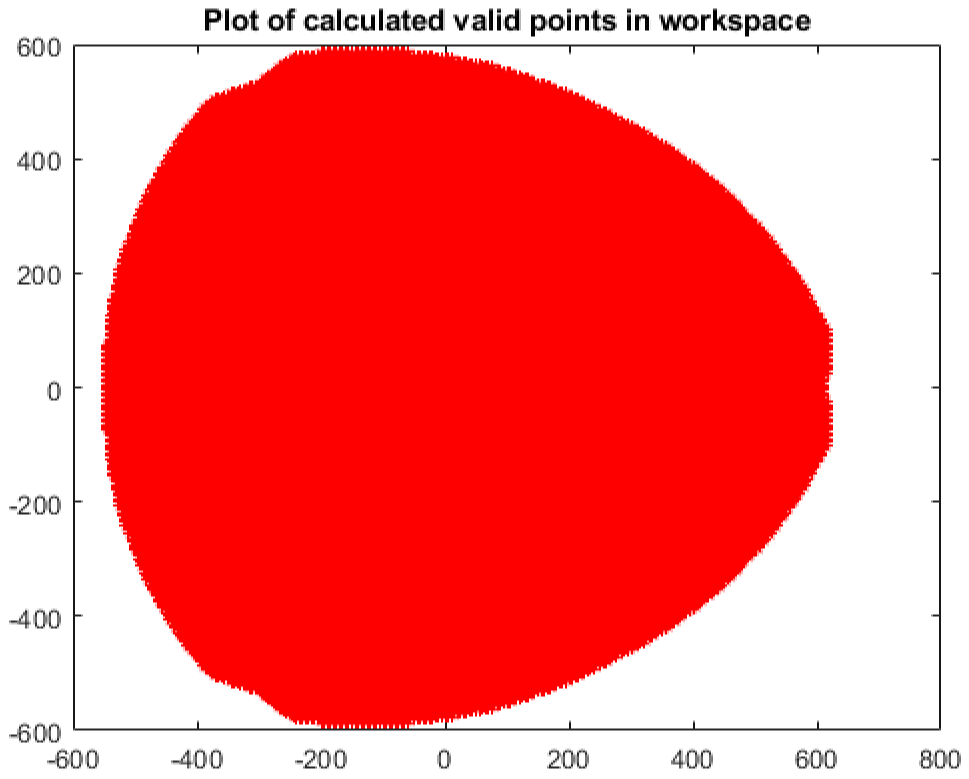
```
% intersections are at
%     [-302.769784433948,-524.412513347726;
%     270.635159474432,-468.753980526729;
%     605.539402561251,0;
%     270.635159474432,468.753980526729;
%     -302.769784433948,524.412513347726;
%     -541.270500359107,0];

% from the graph it is easy to see range of x values are between -545
 and
% 610. Rounding for simplicity.

% y range is -525 to 525 but taking into account the curves of the
 circles,
% round it to -600 and 600.
xRange = 545+610;
```

```
yRange = 600+600;
rectangle('Position',[-545 -600 xRange yRange])

% We can see that the rectangle fully encompasses the workspace. So,
 now we
% need to split the values into 5mm increments, and then check across
 all
% of them to see if they are inside the workspace.
rectXRange = [-545:5:610];
rectYRange = [-600:5:600];
updatePoints = [];

% This is necessary for the surf plot for RSS at the end.
% Use NaN as I used zeros initially and it ruined the final error
 graph
% as the zeros were plotted
graphPlot = NaN(232,241);

for k = 1:numel(rectXRange)
    %     For every single point in rectXRange
    for j = 1:numel(rectYRange)
        %         For every point in rectYRange
        %         create the x,y pair
        curPoint = [rectXRange(k),rectYRange(j)];
        count = 1;
        for i = 1:n
            %     check to see if current point is within workspace of
            %     circles
            circDist = (curPoint(1)-
circleCenters(1,i))^2+(curPoint(2)-circleCenters(2,i))^2;
            if (circDist <= circleRadius^2)
                count = count+1;
            end
        end
        if (count >= 6)
            % If point is within all circles, it is in workspace.
            % add it to the new list
            updatePoints = vertcat(updatePoints,
[curPoint(1),curPoint(2)]);
            % Denote that point is in workspace here in graphplot for
 later
            graphPlot(k,j) = 1;
        end
    end
end
hold off
```



Plot of Entire Workspace

```
figure(4)
plot(updatePoints(:,1), updatePoints(:,2),'r*')
title('Plot of calculated valid points in workspace')
```



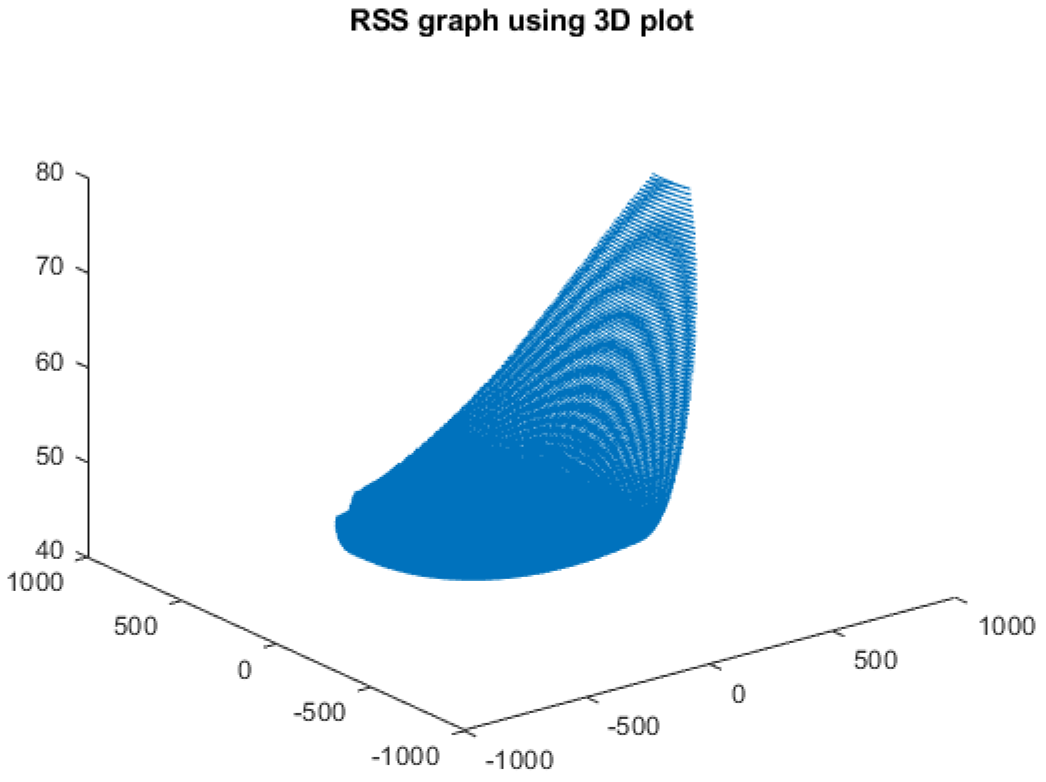
Calculating the forward Kinematic error Model and RSS error

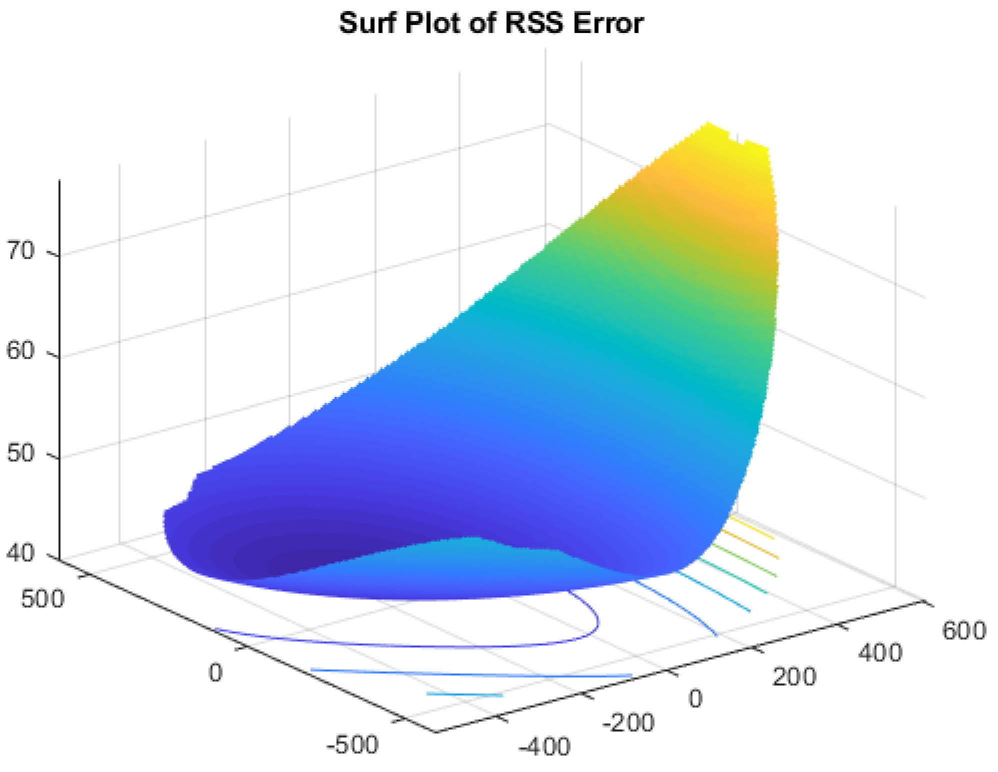
Now we need to calculate the fk error model. This can be calculated from equation 9 in the paper. We are assuming that the pd(deltal) = 0.

```
% pd(P) = inv(jacoV)*(pd(deltal)-jacoRo*pd(ro) <-- equation 9
% Can be rewritten as pd(deltal) = jacov*pd(P)+jacoRo*pd(ro)

% pd(ro) = [pd(si), pd(ui), pd(li)]

% Take real values from table 2
realParam = [96.6610, 81.7602, 1.0684, 305.2599, 115.0695, 2.6210,
 604.4299;
    22.2476, 125.2511, -0.5530, -55.2814, 322.9819, 4.2181, 607.2473;
    -122.4519, 36.6453, 4.3547, -244.7954, 208.0087, 3.9365, 600.4441;
    -120.6859, -34.4565, -4.9014, -252.5755, -211.8783, -3.0128,
 605.9031;
    24.7769, -125.0489, -4.8473, -53.9678, -320.6115, 4.3181,
 604.5251;
    91.3462 -80.9866 0.2515 302.4266 -109.4351 3.3812 600.0616];
sr = realParam(:,1:3)';
ur = realParam(:,4:6)';
realLen = realParam(:,7);
```

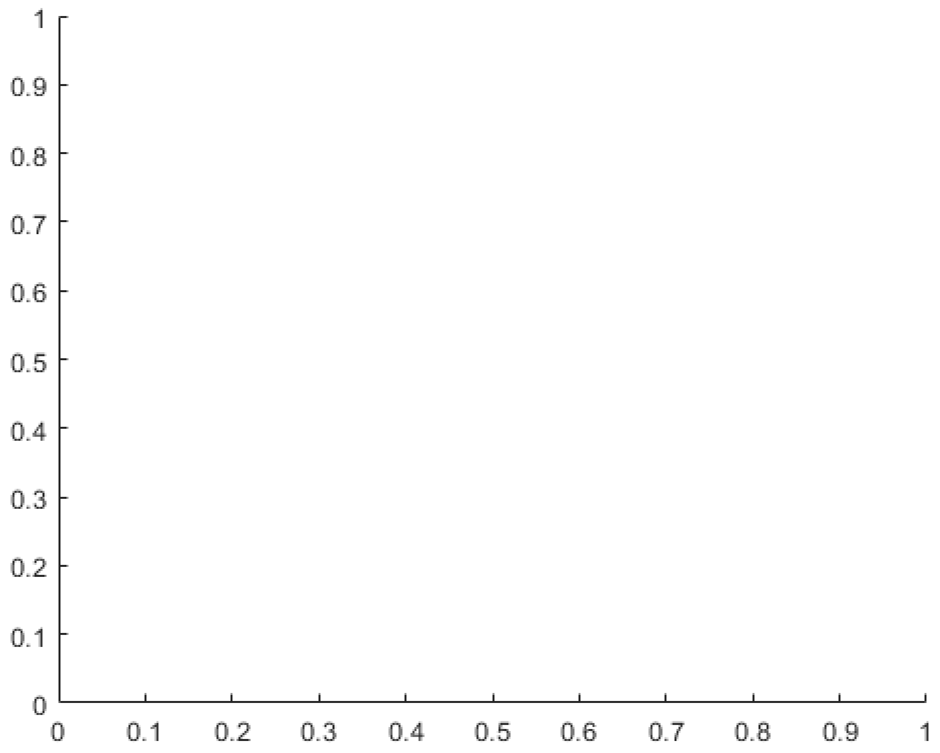
```
% 42 kinematic error (lecture 5)
dRo = [(sr-si)',(ur-ui)',realLen-simuLen];

for i=1:size(updatePoints)
   %     Need to put it into proper format as specified in
 instructions
   %     Will allow us to pass it into pod code
   %     Using pod code as it gives n and R, which are both needed
 for
   %     jaco calculations
   actualPoint(:,i) =
 [updatePoints(i,1),updatePoints(i,2),800,0,0,0];
   [L,l,n,R] = pod(actualPoint(:,i),si,ui); %Taken from Homework 2
   %     Calculate jv and jp

   %     I wasn't sure if I was supposed to use nominal or real
 values for si.
   %     I tried both, and using sr gives a graph that looks closer
 to the
   %     graph in the paper (figure 4).
   %     Since we are doing sr-si this makes sense as in this case sr
   %     would be si then.
   Jv = [n',cross(R*sr,n)'];

   % In the paper, jp is the identity matrix with dimensions of 6x42
   % This calculates individual rows, but now need to put them in
 format
   % shown in the paper (eq.8)
   jpRows = [n'*R, -n', [-1;-1;-1;-1;-1;-1]];
   idMat = zeros(5,7);
   Jp = [];
   for j=0:size(jpRows)-1
       %     Insert row into proper place
       JpMatTop = idMat(1:j,:);
       JpMatRow = jpRows(j+1,:);
       JpMatBot = idMat(j+1:end,:);
       JpMat = [JpMatTop;JpMatRow;JpMatBot];
       %     Append it to the final matrix
       Jp = horzcat(Jp,JpMat);
   end

   % Now we have Jv and Jp and pd(ro) and since pd(deltal) = 0, we
 can now solve for pd(P)

   % Need to reshape pd(ro) first to a 1x42 as shown in paper so that
 matrix math will work eq.8
   dRoReshape = [];
   for k=1:size(dRo)
       % Take each row of dRo, and append it into a 1x42 matrix
       dRoReshape = horzcat(dRoReshape,dRo(k,:));
   end
   %     Plug into equation 9 for full kinematic error
   kinError = Jv\(0-Jp*dRoReshape');
   %     root of sum of squares for each valid point in workspace
```

```
    RSS(i) = sqrt(kinError(1)^2+kinError(2)^2+kinError(3)^2);
end

figure(5)
plot3(updatePoints(:,1), updatePoints(:,2),RSS)
title('RSS graph using 3D plot')

% If we want it as a surf plot like what is shown in 4.1. Need to
 massage
% the values in z so that they are not just a vector anymore.
% Need z to be a matrix with dimensions of workspace

rssCount = 1;
for i=1:size(graphPlot,1)
    for j=1:size(graphPlot,2)
        % Go through graphPlot, if there is a point within workspace
        % (denoted by 1) put the RSS value corresponding to it into
 that
        % spot
        if graphPlot(i,j) == 1
            %  The way I set up RSS, it is a 1x40k matrix, so we need
 external
            %  counter to make sure that it goes through all of the
 values in
            %  RSS
            graphPlot(i,j) = RSS(rssCount);
            rssCount = rssCount+1;
        end
    end
end


figure(6)
surfc(rectXRange,rectYRange,graphPlot','LineStyle','none')
title('Surf Plot of RSS Error')
```


RSS graph using 3D plot
80
70
60
50
40
1000
500
0
-500
-1000
-1000
-500
0
500
1000


Surf Plot of RSS Error
70
60
50
40
500
0
-500
-400
-200
0
200
400
600

```
end
```



createArc Function

```
function [x,y] = createArc(P1,P2,R,C)
% This function takes in two points, and calculates the arc between
 them
% based on a given radius and center of the circle
a = atan2(P1(2)-C(2),P1(1)-C(1));
b = atan2(P2(2)-C(2),P2(1)-C(1));
b = mod(b-a,2*pi)+a; % Ensure that arc moves counterclockwise

t = linspace(a,b,1000);
x = C(1)+R*cos(t);
y = C(2)+R*sin(t);
end
```

Pod Function

```
function[L,l,n,R] = pod(P,s,u)
% From jacobianV and pod in homework 2
% S and U are geometric, and thus are passed in
Xp=P(1:3,1);
a=P(4)*pi/180;
b=P(5)*pi/180;
```

```
c=P(6)*pi/180;
R1=[1,0,0;0,cos(a),-sin(a);0,sin(a),cos(a)];
R2=[cos(b),0,sin(b);0,1,0;-sin(b),0,cos(b)];
R3=[cos(c),-sin(c),0;sin(c),cos(c),0;0,0,1];
R=R1*R2*R3;

for i=1:6
    L(:,i)=R*s(:,i)+Xp-u(:,i);
    l(i)=norm(L(:,i),2);
    n(:,i)=L(:,i)/l(i);
end
end
```

Published with MATLAB® R2017b