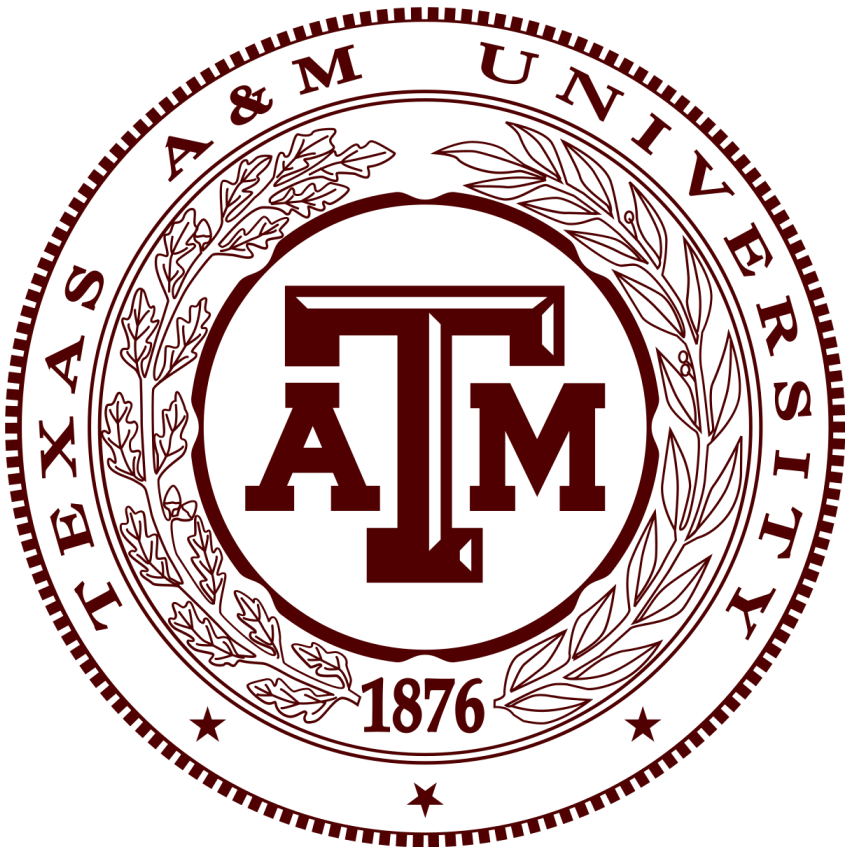


An Analysis of Dubin's Vehicle and the Traveling Salesman Problem

Texas A&M University

MEEN 612-600

December 10th, 2021

Roman Yoder, Joel Fielek

Final Project 2
Joel Fielek
Roman Yoder
Abstract:

The following is an investigation of Dubin's Vehicle problem and the traveling salesman problem. When given a set turning radius and heading Dubin's Vehicle can be used to determine the shortest path between these two points. This concept is relatively obvious when looking at simplified problems, however when the coordinates and heading become more complex or multiply it would be beneficial to create a program that determines this. The first section of this report details the creation and implementation of an algorithm that can be used to solve any Dubin's Vehicle, and can be applied iteratively to any set of headings and points on a two dimensional plane. However alongside Dubin's Vehicle is another well known problem that is the Traveling Salesman Problem. When proposing a set of points it is most often desired to understand the shortest path required to travel to each individual point. While simple examples of this problem may be easily analyzed by hand, more complex arrays and scenarios require the creation and implementation of a program that can solve this. Utilizing the Dubin's Vehicle Plan created previously, a program that can solve the Traveling Salesman Problem was created and tested against a number of applications. These programs were then demonstrated and it is shown that they are effective at solving complex sets of points and headings, and effectively solving the Traveling Salesman problem using Dubin's Vehicle.

Table of Contents

Final Project 2
Joel Fielek
Roman Yoder

Final Project 2
Joel Fielek
Roman Yoder

Introduction

Lester Dubin proposed that when given a vehicle with a set heading, and turning radius on a Euclidean plane that there is at a maximum a path that contains at most a maximum curvature and straight lines. What this really means is that to connect these two points and headings the most that will need to happen is that the vehicle will have to follow a curve with a radius equivalent to its minimum turning radius, a straight line of some distance, and then a final curve to match the heading. This technique seems to have the most immediate application to technologies such as trains and airplanes, which have various restrictions placed on their freedom of movement. However technique proves useful in robotics as it can be used to plan optimized paths for vehicle based robotics. However applying Dubin's methods to actual examples can be quite difficult and complex as the more locations and headings are added to the potential path tend to make finding the optimal path confounding. To make the most use of Dubin's Vehicle it would be best to create a program that could take two points and two headings as inputs, and then output the most optimal path for these two points.

Once this program is created it could then be used and applied to a classic logistics problem, which is the Traveling Salesman Problem. This problem is proposed as a method of determining the shortest path to each location, when given a set of locations and distances between those locations. If all of these locations were Dubin's Vehicle locations with set headings, then the program created previously could be used to create paths between all of the locations, and determine the distance between them all. Given this information a nodal network can then be created, and evaluated to determine which paths form the optimal path between all these points, and the Travelling Salesman Problem can be solved.

This report will be used to detail the process and results of creating the Dubin's Vehicle program, and code. It will then detail the implementation of this code to generate satisfactory results, and all possible situations that a Dubin's Vehicle can encounter when trying to find the optimal route from location to location. After this it will be shown how this program will be applied to the Traveling Salesman Problem and how the nodal networks were formed and analyzed when coming to a conclusion using these methods.

Final Project 2
Joel Fielek
Roman Yoder

Body

The first step to approaching the Dubin's Vehicle problem is to draw the correct geometric constraints:

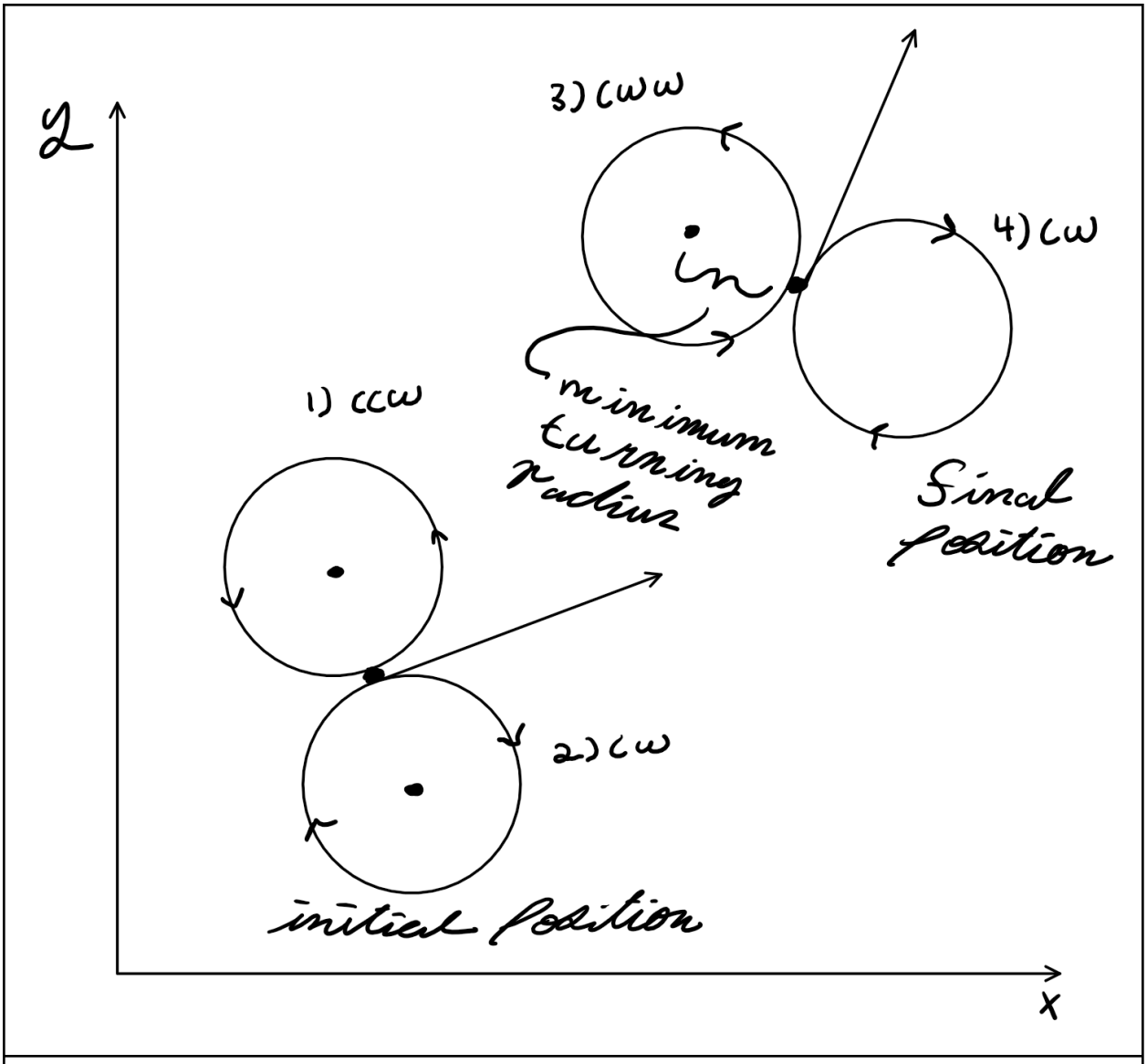


Figure 1: Geometric Problem Set Up

Clearly, Given a second point and heading that lies outside of the minimum turning radius it is immediately advantageous to start a clockwise or counterclockwise rotation, this rotation occurs until a tangent line is coincident with a tangent line of either of the second points turning circles. This coincident tangent line must also align in the direction of travel between both circles. As such between two points and given two sets of opposite spinning circles there exists only four solutions as shown.

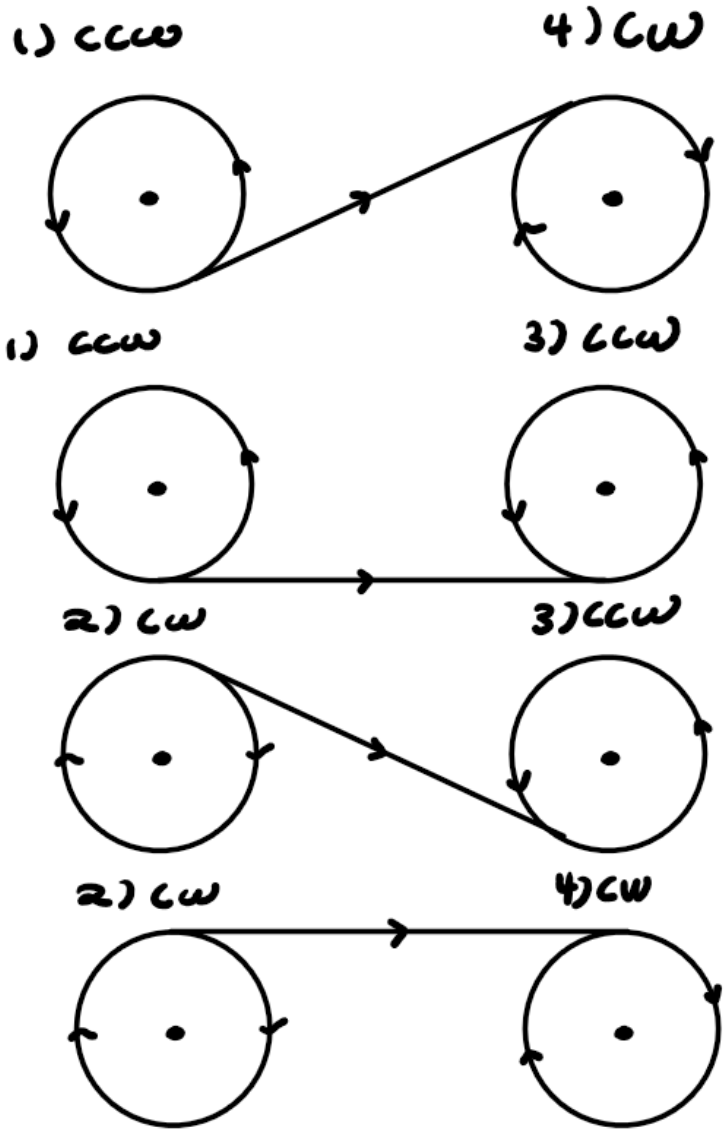


Figure 2: Possible solutions for CSC(Curve Straight Curve)

Then, from the possible trajectories presented the optimum is found simply by summing the arc and line lengths and selecting the lowest valued path. However, complications arise when the desired point and heading lies within the minimum 4 times the turning radius. When this occurs a possible solution is known as the teardrop maneuver drawn below:

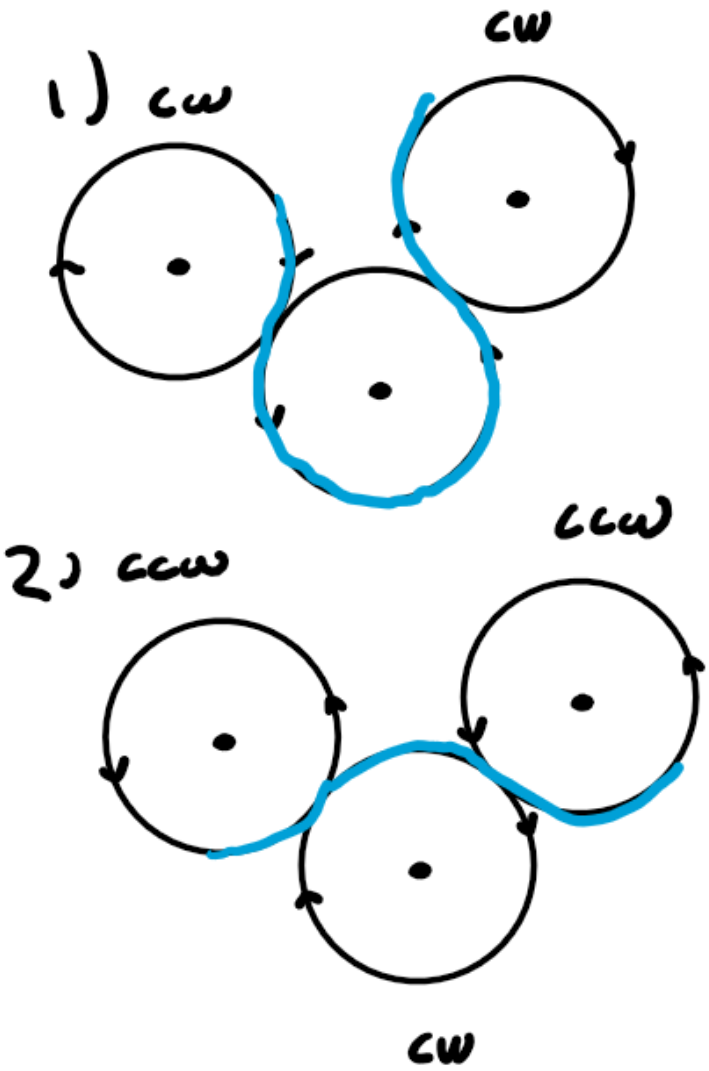


Figure 3: Teardrop Solutions CCC (Curve Curve Curve)

Notice, for this teardrop to be a solution both the start and end circle must be spinning in the same direction, opposite the center circle.

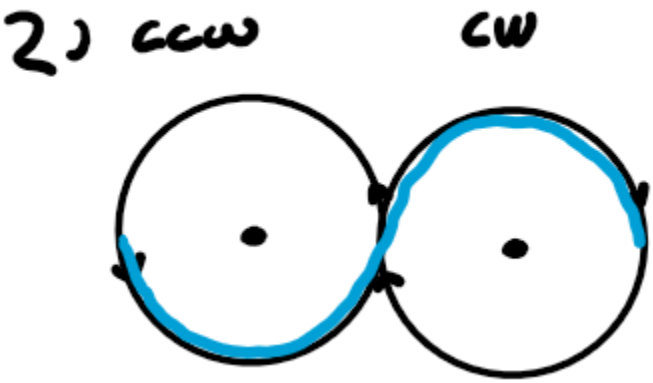


Figure 4: CC (curve curve) solution

Figure 4 represents a unique solution to the CSC solution in which the tangent line between the circle is zero, resulting in a CC solution.

The algorithm used to develop the optimum path was implemented with all original software. This was done by first checking if CCC was a solution via the four radius condition, then using geometric identities to solve the path lengths of all CCC and CSC solutions or just CSC; and select the path with the shortest possible cost. Mathematical impossibilities had to be ruled out, such as, paths that instantly changed direction opposite the rotation of the circle or paths where there was no real solution to the tangent line of the CSC problem. The results of this algorithm are presented in the following section.

The traveling salesman problem finds the optimal solution from a starting point to hit every location once and then return to the desired start point. The optimal solution is typically bounded with what is known as a minimum spanning tree or MST. This minimum spanning tree is the minimum cost path to hit every node without returning to the original point. The solution to the TSP problem is typically bounded as follows:

$$MST < TSP < 2 * MST$$

Where the cost of the optimum TSP solution lies between MST and twice the MST cost.

Final Project 2
Joel Fielek
Roman Yoder

Results:

Dubin's Vehicle Results:

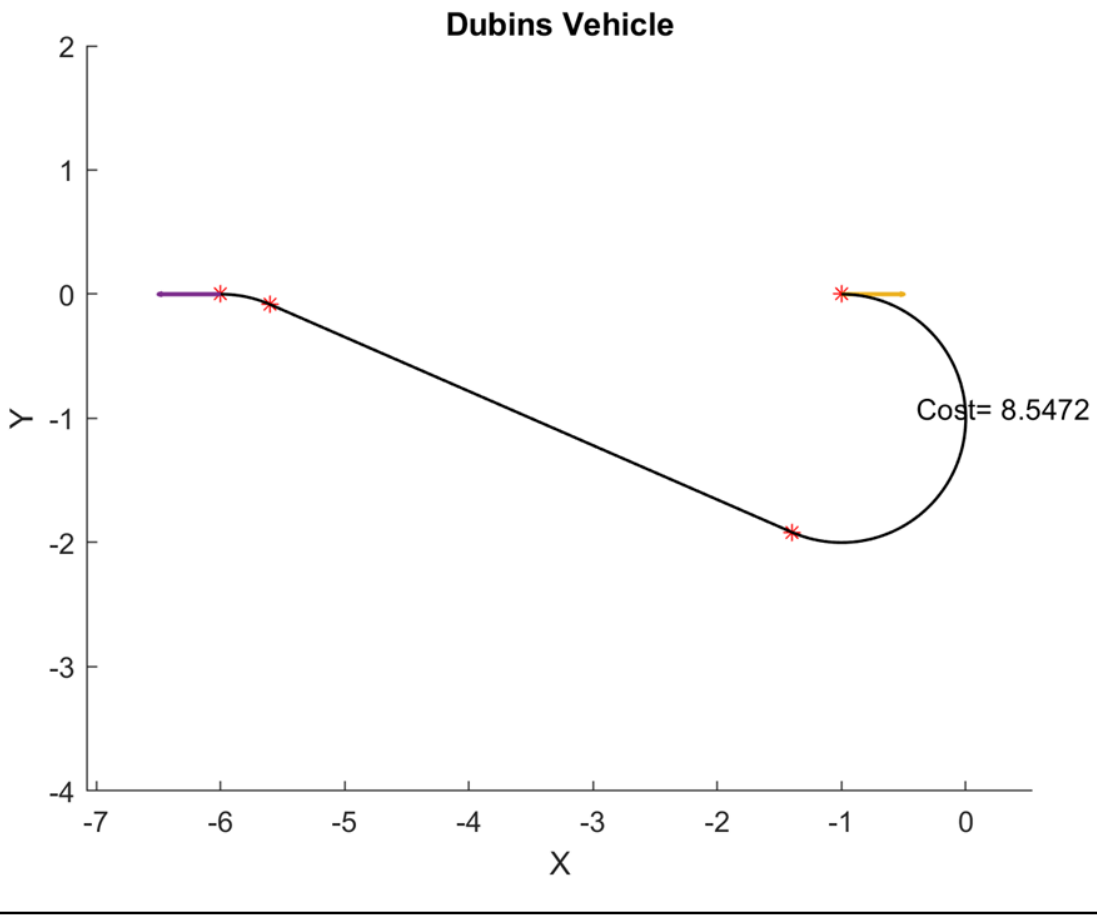


Figure 5: Functionality for negative numbers

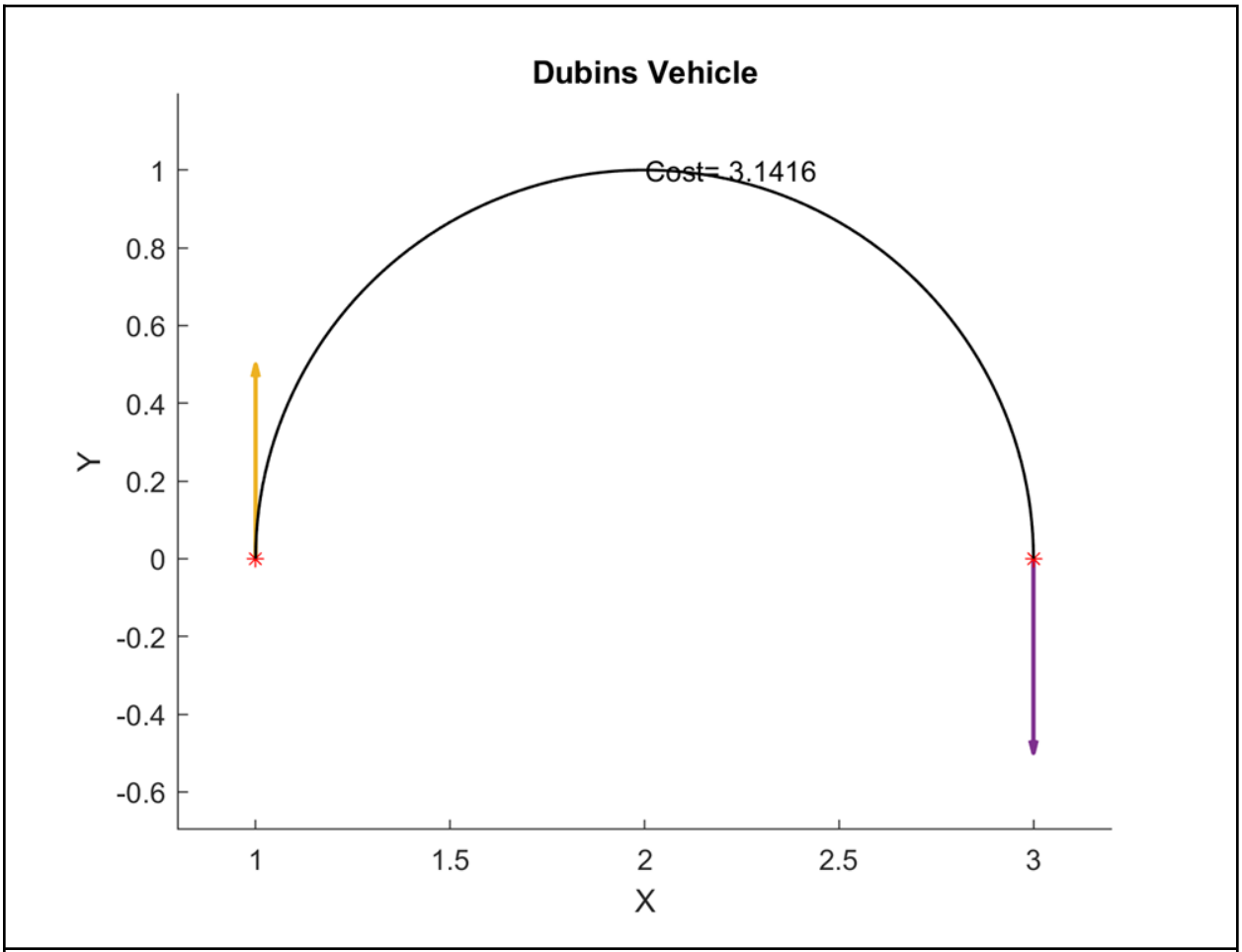


Figure 6: Pivot Maneuver

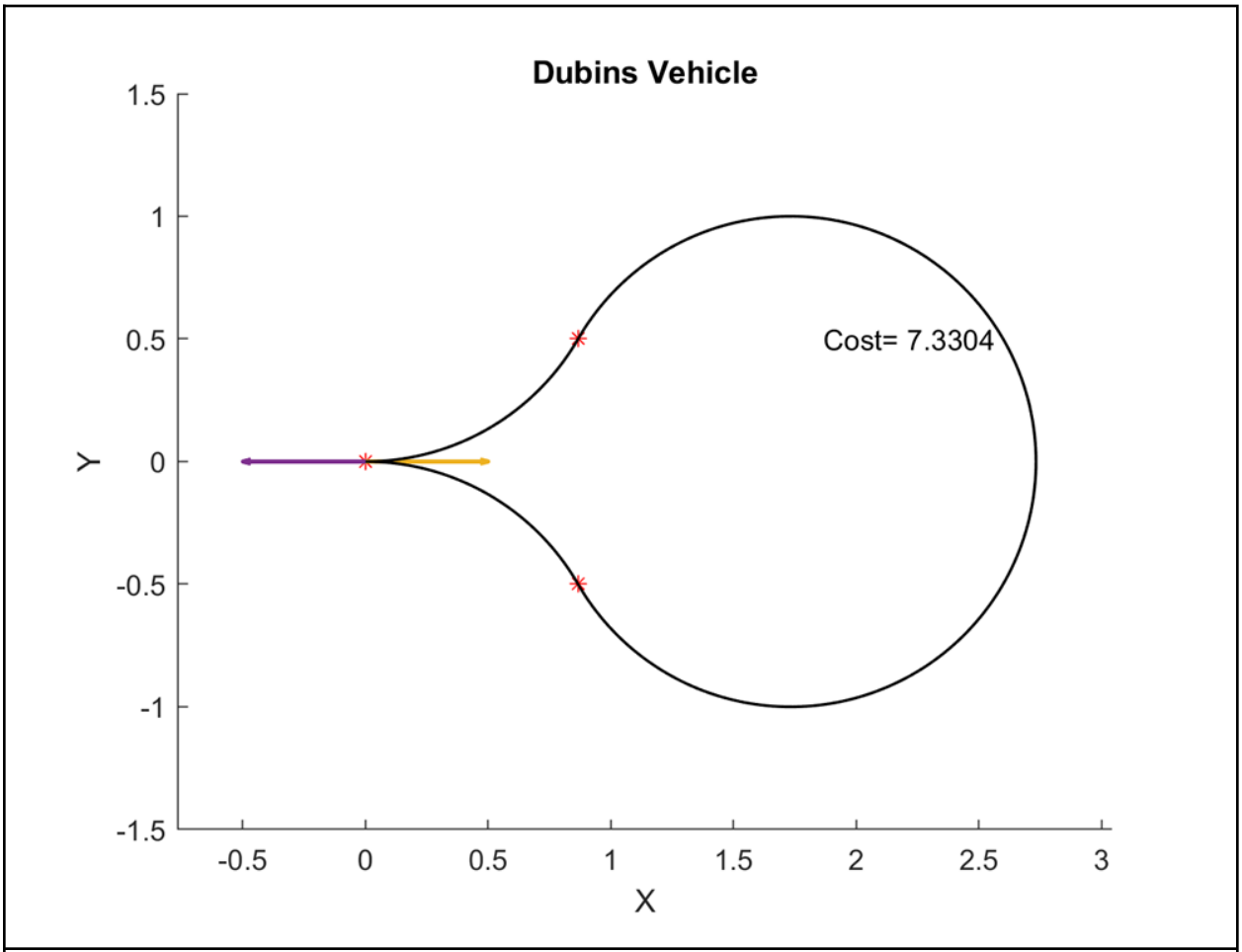


Figure 7: Teardrop Maneuver

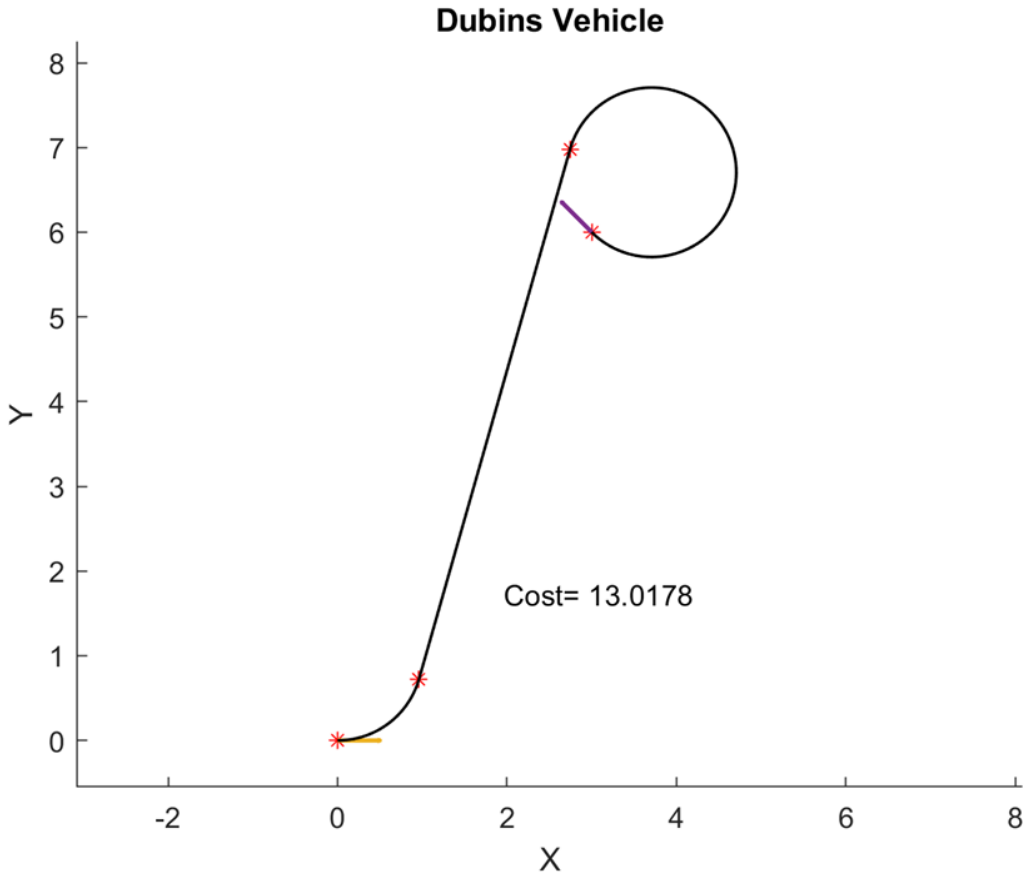


Figure 8: Curve Straight Curve Solution

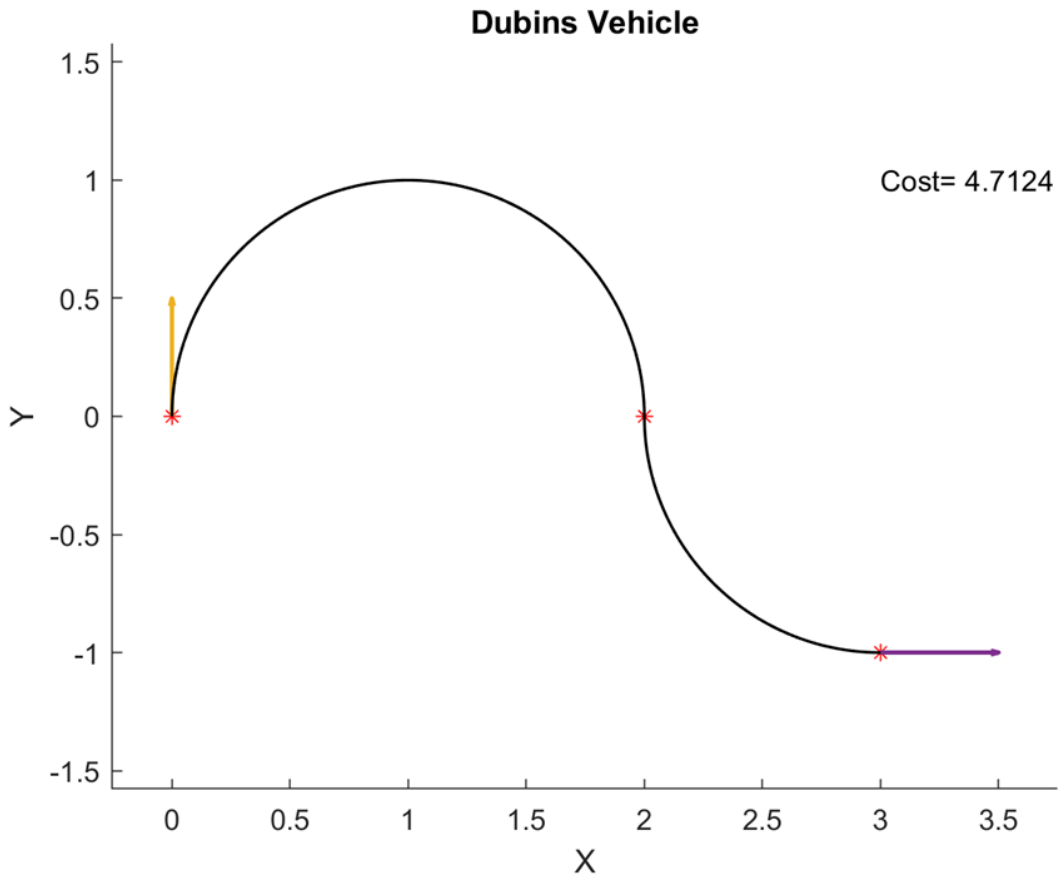


Figure 9: CC Maneuver

Travelling salesman results:
Given the selected nodes below:

Table 1: Network Nodes		
Node Number:	Position (X,Y)	Heading (X,Y)
1	(1,3)	[-1,0]
2	(5,6))	[1,0]
3	(20,5)	[0.6,0.8]
4	(15,2)	[-.2425,0.9701]
5	(20,1)	[0,1]

MATLAB is used to draw a directional graph:

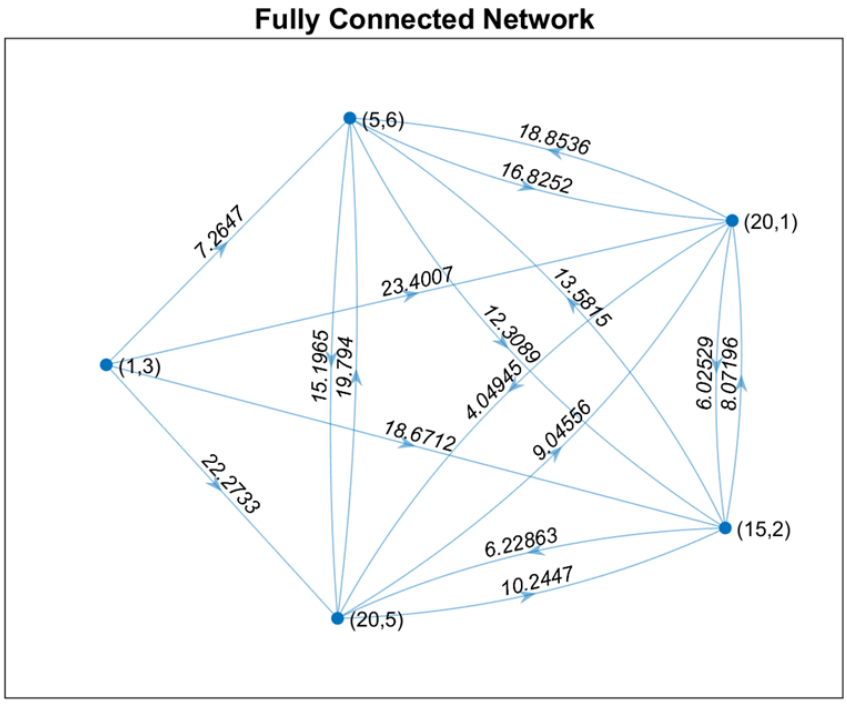


Figure 10: Fully Connected Directional Network with cost determined by Dubin's Vehicle algorithm originating from node 1

Unfortunately, MATLAB lacks the functionality to solve a directional bigraph in the minimum spanning tree function, as such best estimates are made for the MST cost. First, the non directional graph is created and utilized in the MST function:

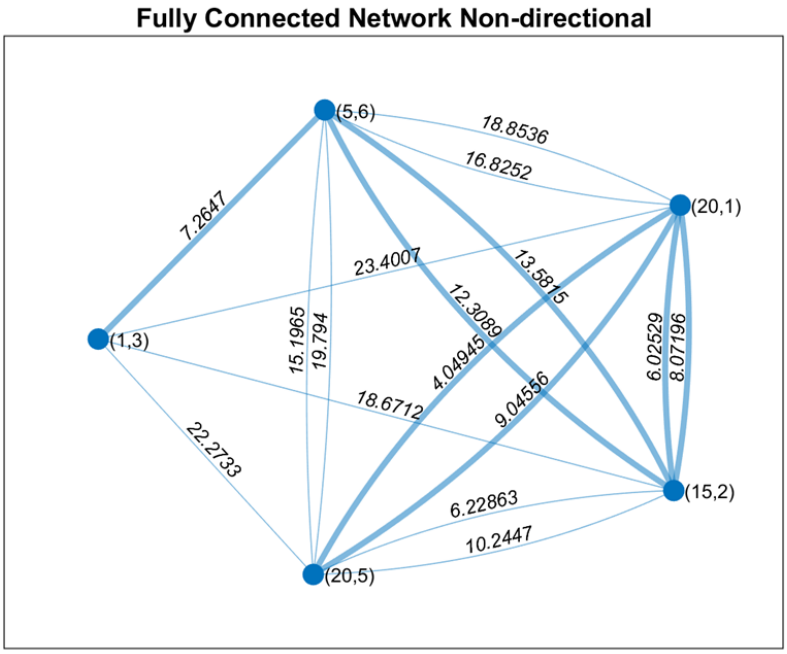


Figure 11: The solutions to the MST function for non-directional paths presents two solutions, but does not differentiate between path directions

To verify this solution a rooted tree is created from the starting node:

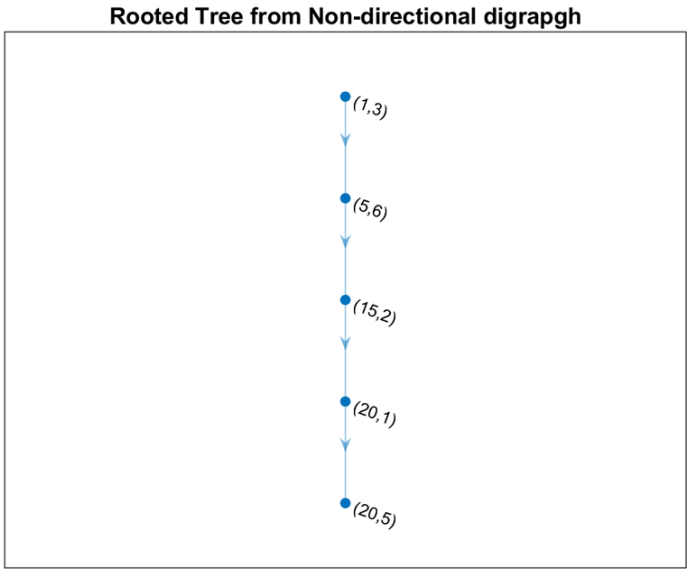


Figure 12: Rooted Tree

We can further confirm this solution by removing the paths in the reverse direction of the MST solution from the non-directional digraph and resolving the MST function repeatedly. First as the solution is guaranteed to use the second point of (5,6) the paths approaching this node from following nodes are removed.

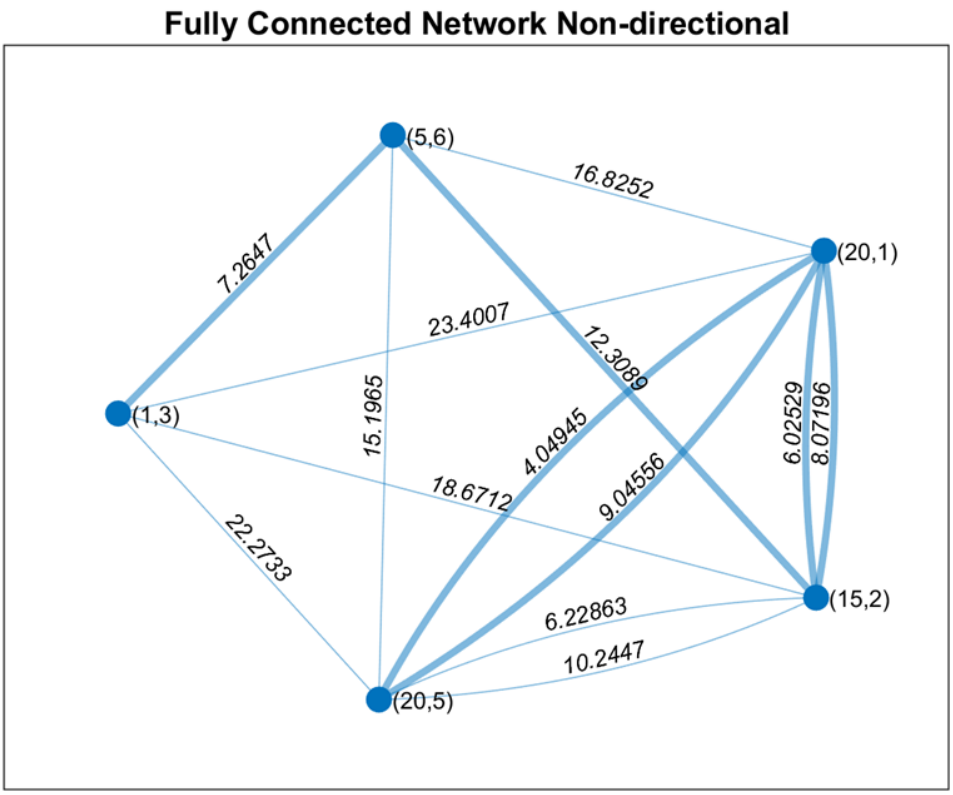


Figure 13: MST reverse edges removed from first solution

This then guarantees that the 3rd node will be (15,2), but we are now faced with the complexity of a circular connection. The reverse paths are removed in the current solution:

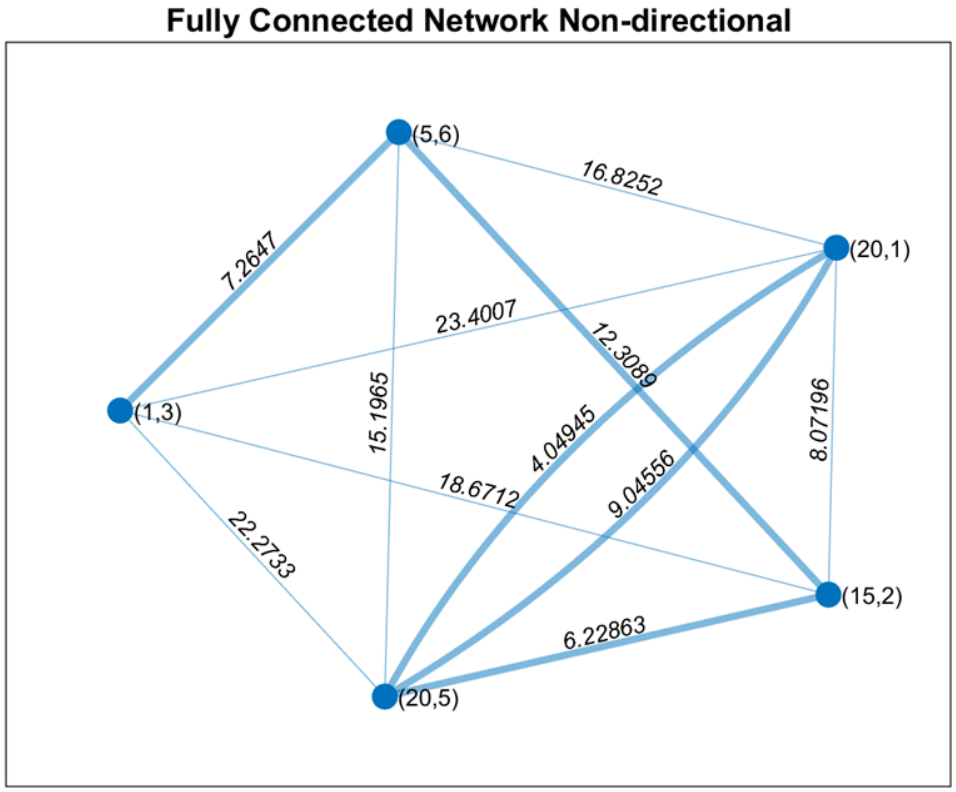


Figure 14

But Figure 14 fails to account for the difference in direction between ‘4.049’ and ‘9.045’ remedying this error provides the following MST solution:

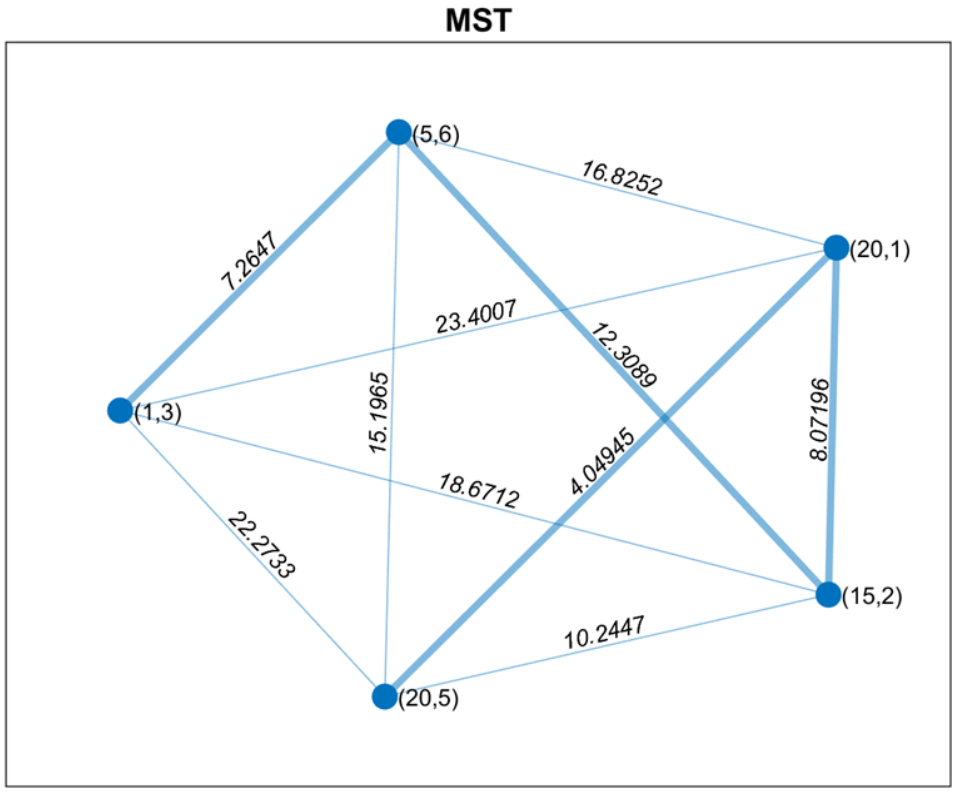


Figure 15: Total MST cost of 31.69

This matches the rooted tree previously provided.

In order to implement the travelling salesman problem an online toolbox was used from [1]. First a distance matrix was created to provide the associated cost between nodes to the algorithm.

Table 2: Distance matrix

Start Node/End Node	1	2	3	4	5
1	0	7.26	22.27	18.67	23.40
2	7.26	0	15.19	12.30	16.82
3	20.769	19.79	0	10.244	9.04
4	20.03	13.58	6.22	0	8.07
5	24.93	18.85	4.04	6.0253	0

Notice, as in the network map, the cost from one node is not the same in the reverse direction and the associated cost from a node to itself is zero. This distance matrix was then passed to the TSP solver to provide the following order of stops.

Table 3: Node Order				
2	4	5	3	1

This solution can then be generated in a for loop with the Dubin's plot function to generate the following solution.

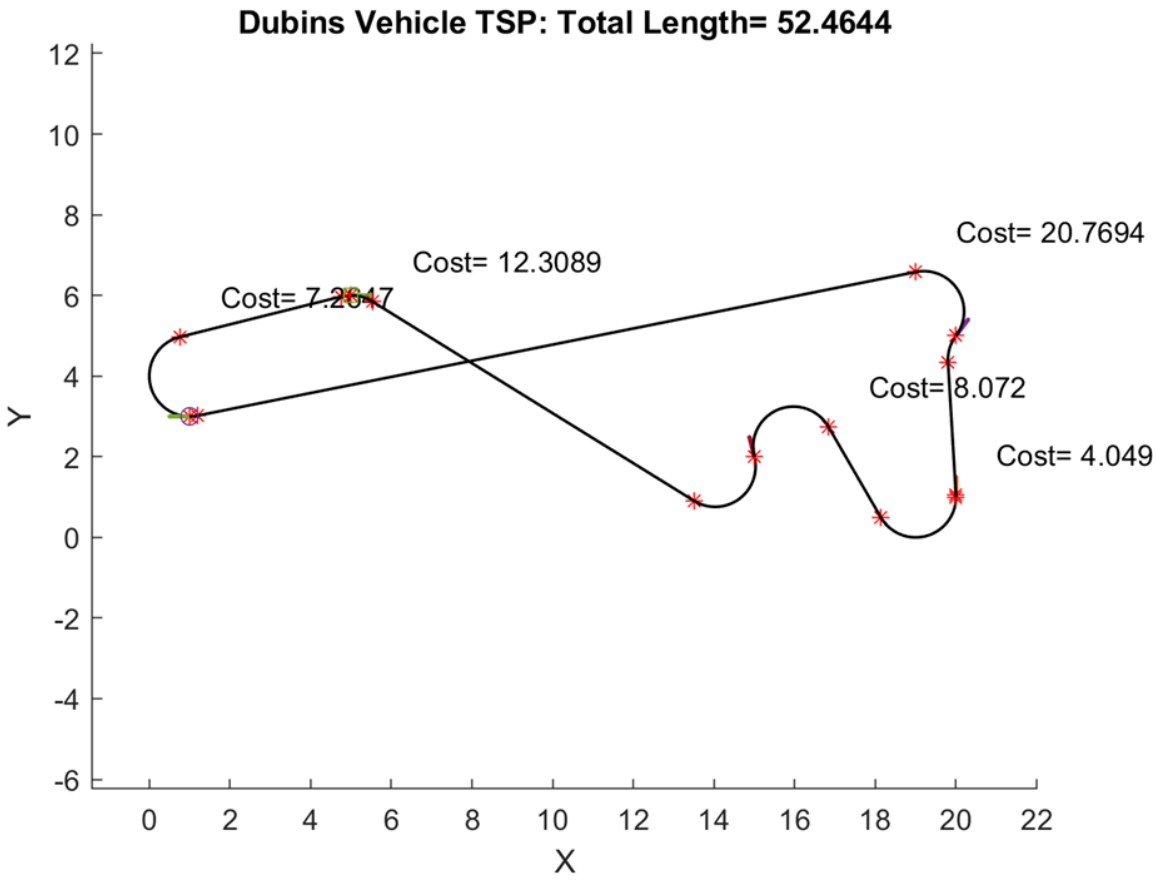


Figure 16: Dubin's Vehicle Network Solution

Bounding this solution:

$$MST < TSP < 2 * MST$$
$$31.69 < 52.46 < 63.39$$

Final Project 2
Joel Fielek
Roman Yoder
Flow Charts:

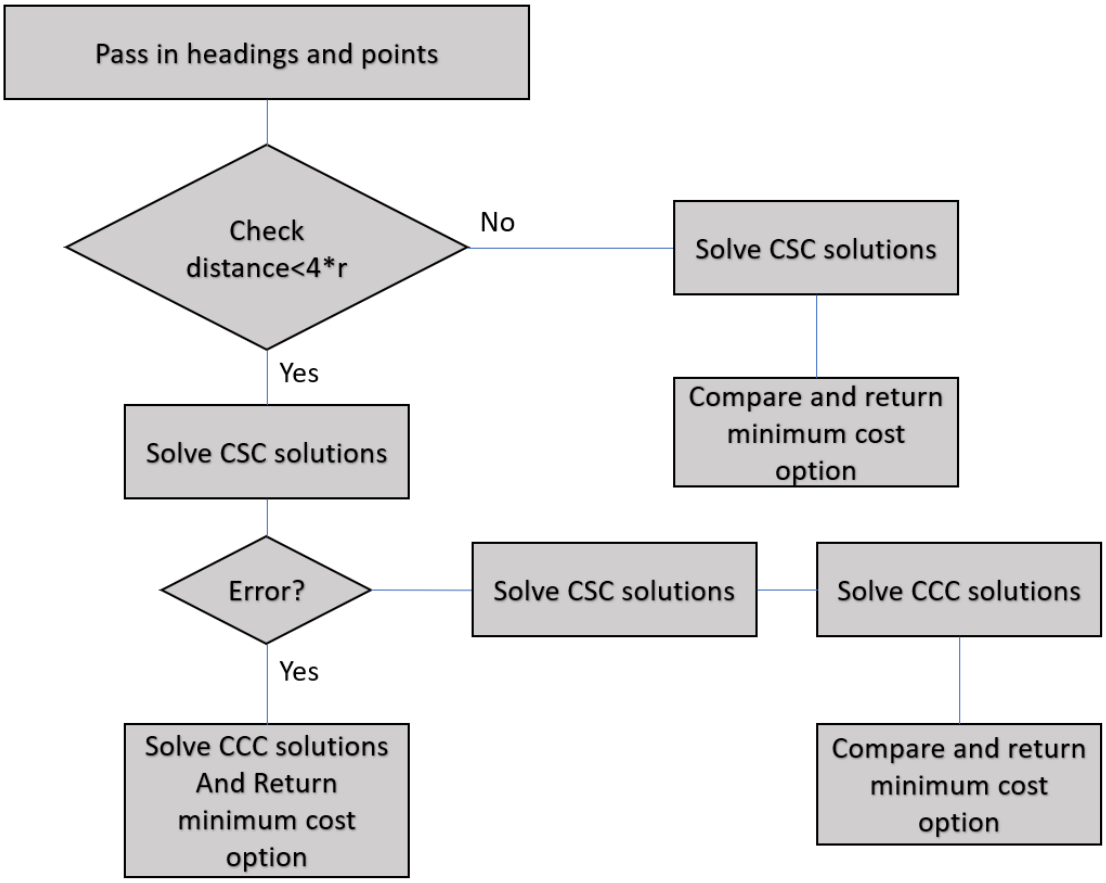


Figure 17: Dubin's Vehicle Flow Chart

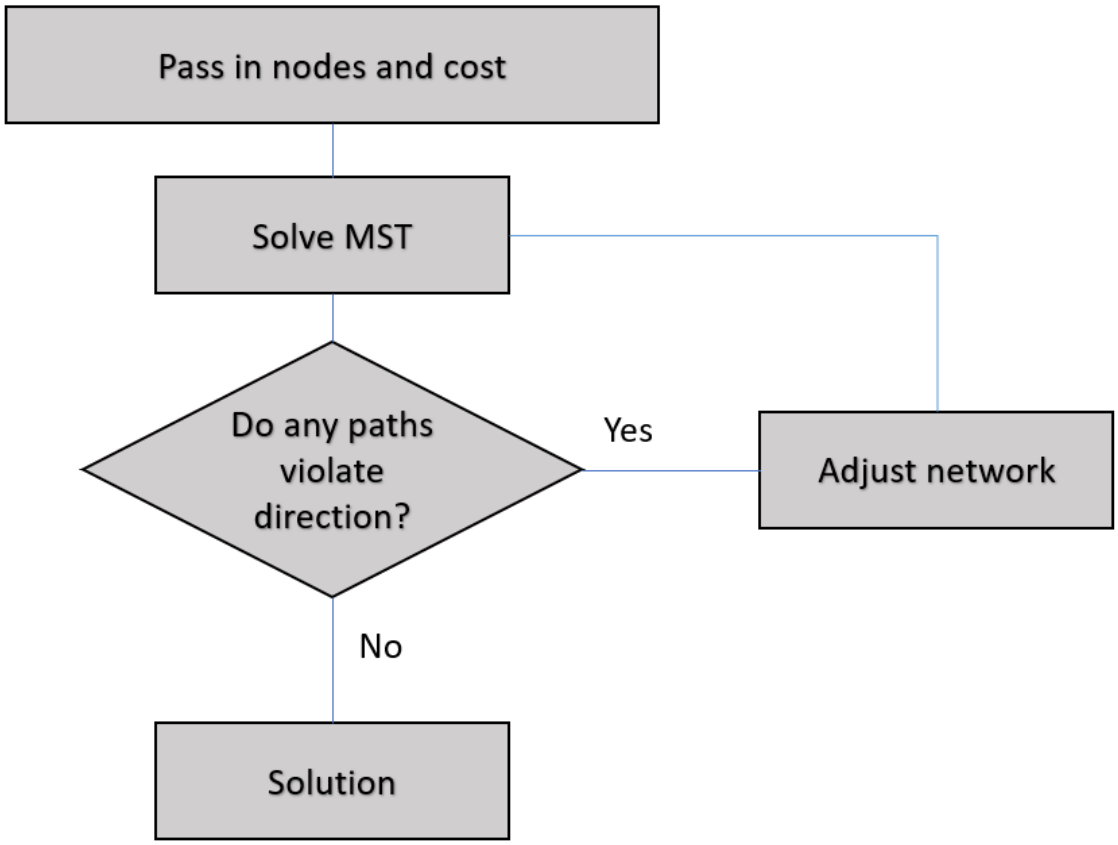


Figure 18: Simplified MST algorithm

Conclusion

Although the solution to the Dubin's Vehicle problem is often easily obtained from an observer and geometric arguments. Structuring a computer program to solve this problem is complex. Often, during the development of the code the program would return solutions which while they were the minimum length, were unachievable in the real world. An example of this were solutions in which the linear portion of the CSC line would have instantaneously changed direction from the curve leading into it, as such the program then accounted for the fact that rotation of the curve had to match the rotation generated by the straight line around the circle's center. Another issue that had to be resolved was that the program would occasionally return the total change in angle for a curve to be 2π this would add an unnecessary length to the solution; thus, these angles had to be normalized with an if statement. The results are mathematically beautiful and once presented it is clear that they meet the optimum path.

The results from the MSP calculation can be confirmed by hand calculating alternatives. It is found that this is the minimum spanning tree. When the MSP was implemented in MATLAB the user decided what edge lengths to remove or add to cause the solution to converge to the real value of the MSP. Given a much larger network this process would be cumbersome and it would be optimal to design an algorithm that can know and account for the direction of travel of each edge. Given more time this would have been implemented. The TSP problem was solved from a given algorithm, developing this algorithm individually could take a multitude of days, but would be a useful exercise for understanding. As expected this solution lied within the bounds of MSP and 2*MSP. The TSP problem solved is also known as an asymmetric problem as costs from one node to another do not match the cost in the reverse direction, typically these are easier to solve then symmetric TSP problems.

The next step moving forward in the Dubin's algorithm would be to implement a solver that can account for obstacle avoidance. This would have relecencey in autonomous vehicles that either need to detect, or know ahead of time, objects in its path from start to finish and avoid these. Once this algorithm is created it could again be coupled with a TSP algorithm to allow a multitude of applications such as drone flight path planning.

Additional questions arise given additional or modified constraints. What if some of the points have specified headings and others do not? How can the Dubin's vehicle algorithm be modified for this?

Overall this project was a useful exploration into the implementation of optimization algorithms for solving real world problems.

Citations

[1]Santhanakrishnan Narayanan (2021). Travelling Salesman Problem (https://www.mathworks.com/matlabcentral/fileexchange/64654-travelling-salesman-problem), MATLAB Central File Exchange. Retrieved December 8, 2021.

Final Project 2
Joel Fielek
Roman Yoder

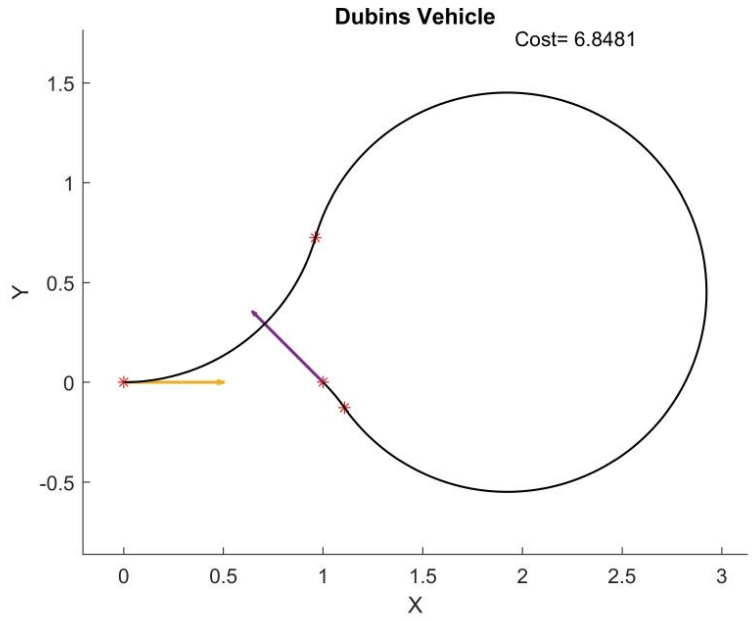
Appendix

Code (This does not include embedded function):

```
p1=[0,0];
%start point
p2=[1 0];
%end point
h1=[1 0];
%heading 1
h2=[-0.707 0.707];
%heading 2
r=1;
%radius
[solutiontype,startendcircle,CCCcircle,optpathpoints,theta1,theta2,theta12,theta3,theta4,theta3
```

```
solutiontype = 1
startendcircle = 1×5
         0    1.0000    0.2930   -0.7070    4.0000
CCCcircle = 1×3
    1.9233    0.4515         0
optpathpoints = 1×5
    0.9617    0.7257    1.1082   -0.1278         0
theta1 = 4.7124
theta2 = 6.0054
theta12 = 1.2930
theta3 = 0.6178
theta4 = 0.7854
theta34 = 0.1676
thetaCCC1 = 2.8638
thetaCCC2 = 3.7594
thetaCCC12 = -5.3876
cost = 6.8481
```

```
figure(1)
clf;
PLOT = Dubinsplot(solutiontype,startendcircle,CCCcircle,optpathpoints,theta1,theta2,theta12,the
```




```
PLOT =
  Figure (1) with properties:

      Number: 1
        Name: ''
       Color: [1 1 1]
    Position: [488 342 560 420]
       Units: 'pixels'

  Show all properties
```

Okay what does each output represent?

Solution Type Denotes wether it is CCC or a CSC solution

1=CCC 0=CSC

Start end circle gives the solution to the first and the final circle used in the path as well as their rotational relation.

first two points [X1,Y1] =centert of first circle

second two point [X2,Y2]=centert of final circle

last index= roational relation

1= Cw to CW

2= CW to CCW

3=CCW to CCW

4=CCW to CCW

optpathpoints represents the points that lie on the first and second circle in the optimum path [X1,Y1],[X2,Y2], [rotational relation]

CSCCIRCLE represents the points of the circle for CCC if the solution is CSC this will be N/A

theta1 is the rotation in the positive z axis from the +X AXIS to the starting point of the first circle

theta2 is the rotation in the positive z axis from the +X AXIS to the exiting point of the first circle

theta12 is the diffrence and direction from the start point to the exit point

theta3, theta4, and theta34 follow the same convention on the last circle but from entrence point to finish.

thetaCC1,thetaCC2,thetaCCC12 follow the same conevention but on the CCC circle if it exists

cost represents the total length

MST

```
N=[[1 3],[-1 0];
```

```
    [5 6],[1 0];
    [20 5],[3 4]/norm([3 4]);
    [15 2],[-1 4]/norm([-1 4]);
    [20 1],[0 1]]
```

```
N = 5×4
    1.0000    3.0000   -1.0000         0
    5.0000    6.0000    1.0000         0
   20.0000    5.0000    0.6000    0.8000
   15.0000    2.0000   -0.2425    0.9701
   20.0000    1.0000         0    1.0000
```

```
network=zeros(15,7)
```

```
network = 15×7
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     ⋮
```

```
h=0;
q=0;
size(N,1)
```

```
ans = 5
```

```
for i=1:size(N,1)
    start=N(i,:);
    for l=1:size(N,1)
        finish=N(l,:);

        [~,~,~,~,~,~,~,~,~,~,~,~,~,cost]=optpath(start(1:2),finish(1:2),start(3:4),finish(3:4),
        %optpath(p1,p2,h1,h2,r)

        if not(prod(start==finish))&not(l==1)
            h=h+1;
            network((h),:)=[start(1:2),finish(1:2),cost,i,l];
        end

    end
end
network
```

```
network = 16×7
    1.0000    3.0000    5.0000    6.0000    7.2647    1.0000    2.0000
    1.0000    3.0000   20.0000    5.0000   22.2733    1.0000    3.0000
    1.0000    3.0000   15.0000    2.0000   18.6712    1.0000    4.0000
    1.0000    3.0000   20.0000    1.0000   23.4007    1.0000    5.0000
    5.0000    6.0000   20.0000    5.0000   15.1965    2.0000    3.0000
    5.0000    6.0000   15.0000    2.0000   12.3089    2.0000    4.0000
```

```
    5.0000    6.0000   20.0000    1.0000   16.8252    2.0000    5.0000
   20.0000    5.0000    5.0000    6.0000   19.7940    3.0000    2.0000
   20.0000    5.0000   15.0000    2.0000   10.2447    3.0000    4.0000
   20.0000    5.0000   20.0000    1.0000    9.0456    3.0000    5.0000
      ⋮
```

```
%Start nodes
s=transpose(network(:,6))
```

```
s = 1×16
     1     1     1     1     2     2     2     3     3     3     4     4     4 ···
```

```
%End nodes
t=transpose(network(:,7))
```

```
t = 1×16
     2     3     4     5     3     4     5     2     4     5     2     3     5 ···
```

```
%Path Cost
weights=transpose(network(:,5));
A=transpose(string(N(:,1:2)));
names=append('(',A(1,:),',',A(2,:),')');
G = digraph(s,t,weights,names);
figure(2)
```

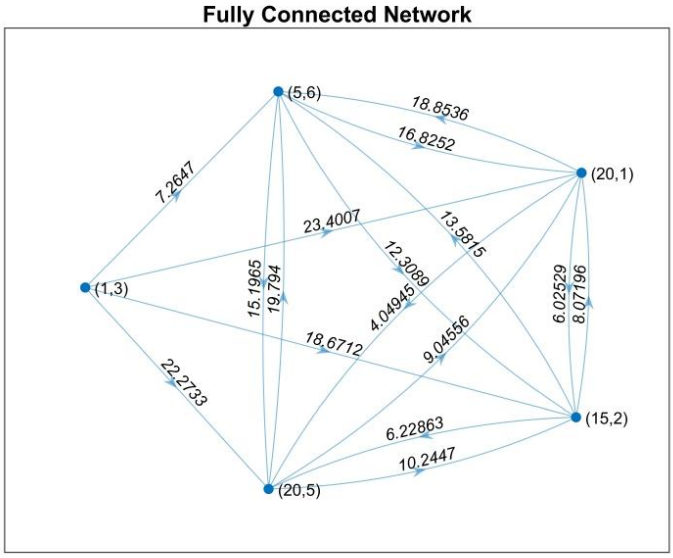
```
ans =
  Figure (1) with properties:

      Number: 1
        Name: ''
       Color: [1 1 1]
    Position: [488 342 560 420]
       Units: 'pixels'

  Show all properties
```

```
clf;
p=plot(G,'EdgeLabel',G.Edges.Weight);
title('Fully Connected Network')
```

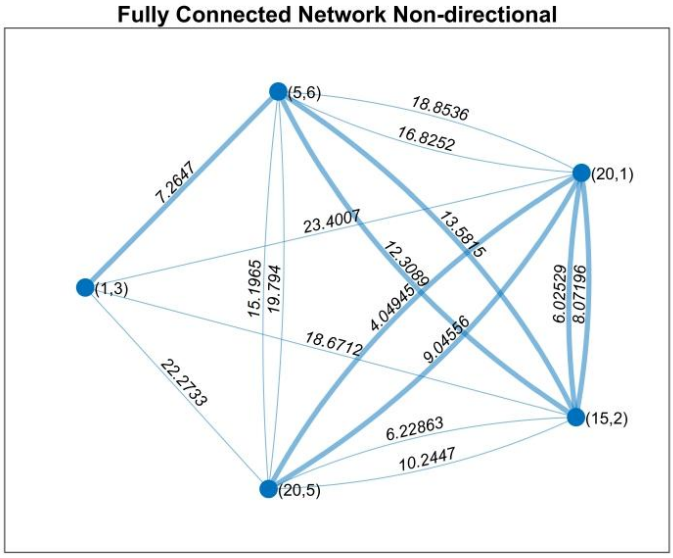
Final Project 2
Joel Fielek
Roman Yoder



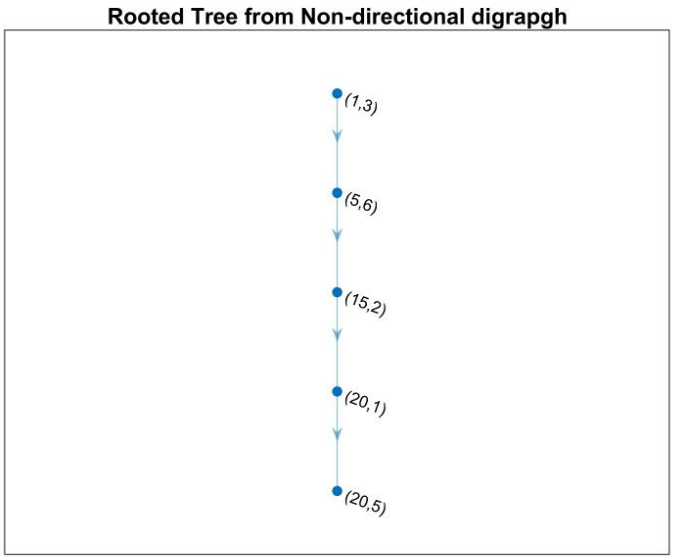

```
s = 1×16
     1     1     1     1     2     2     2     3     3     3     4     4     4 ···
t = 1×16
     2     3     4     5     3     4     5     2     4     5     2     3     5 ···
weights = 1×16
    7.2647   22.2733   18.6712   23.4007   15.1965   12.3089   16.8252   19.7940 ···
```

```
%Start nodes
s=transpose(network(:,6));
%End nodes
t=transpose(network(:,7));
%Path Cost
weights=transpose(network(:,5));
A=transpose(string(N(:,1:2)));
names=append('(',A(1,:),',',A(2,:),')');
figure(3);
clf;
G = graph(s,t,weights,names);

p=plot(G,'EdgeLabel',G.Edges.Weight);
title('Fully Connected Network Non-directional')
[T,pred]=minspantree(G);
highlight(p,T)
```

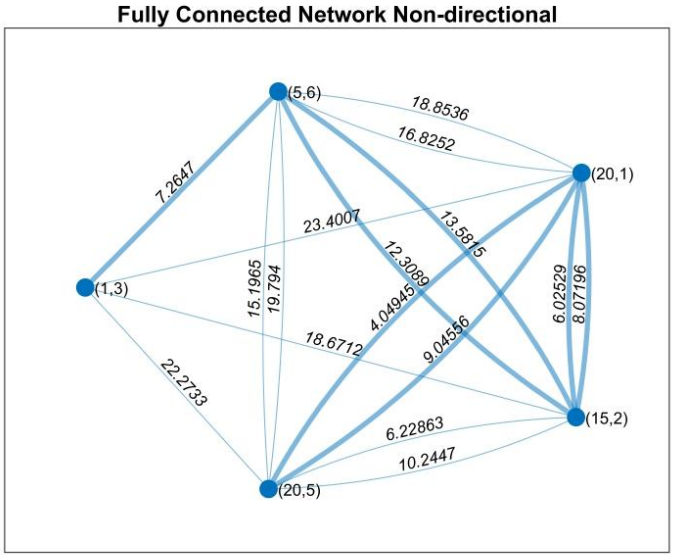



```
figure(4);
clf;
rootedTree = digraph(pred(pred~=0),find(pred~=0),[],G.Nodes.Name);
plot(rootedTree)
title('Rooted Tree from Non-directional digrapgh')
```




```
figure(5);
clf;
G = graph(s,t,weights,names);

p=plot(G,'EdgeLabel',G.Edges.Weight);
title('Fully Connected Network Non-directional')
[T,pred]=minspantree(G);
highlight(p,T)
```



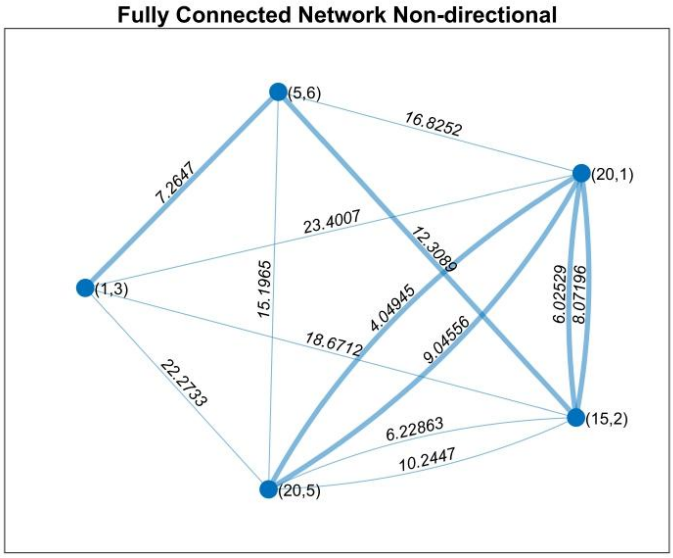

```
clf;
edgetab=G.Edges
```

edgetab = 16×2 table

	EndNodes		Weight
1	'(1,3)'	'(5,6)'	7.2647
2	'(1,3)'	'(20,5)'	22.2733
3	'(1,3)'	'(15,2)'	18.6712
4	'(1,3)'	'(20,1)'	23.4007
5	'(5,6)'	'(20,5)'	15.1965
6	'(5,6)'	'(20,5)'	19.7940
7	'(5,6)'	'(15,2)'	12.3089
8	'(5,6)'	'(15,2)'	13.5815
9	'(5,6)'	'(20,1)'	16.8252
10	'(5,6)'	'(20,1)'	18.8536
11	'(20,5)'	'(15,2)'	10.2447
12	'(20,5)'	'(15,2)'	6.2286
13	'(20,5)'	'(20,1)'	9.0456
14	'(20,5)'	'(20,1)'	4.0494

	EndNodes		Weight
15	'(15,2)'	'(20,1)'	8.0720
16	'(15,2)'	'(20,1)'	6.0253

```
G = rmedge(G,[8 10 6]);
% G = rmedge(G,'(15,2)','(5,6)',);
% G = rmedge(G,'(20,1)','(15,2)',);
% G= rmedge(G,'(20,5)','(20,1)',);
p=plot(G,'EdgeLabel',G.Edges.Weight);
title('Fully Connected Network Non-directional')
[T,pred]=minspantree(G);
highlight(p,T);
```



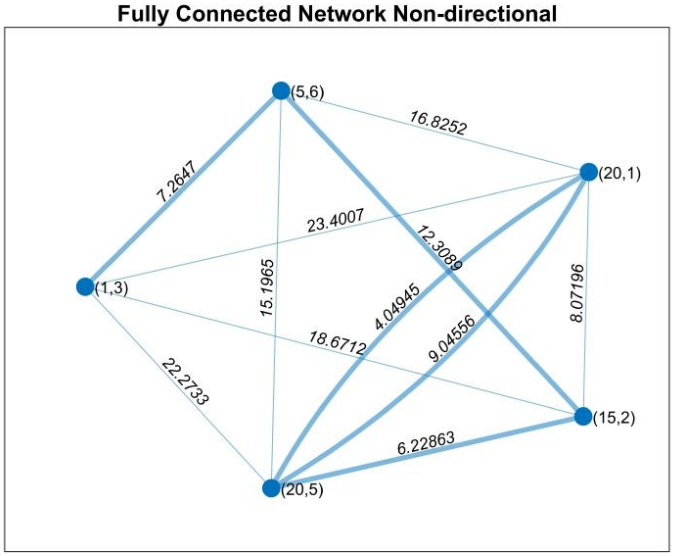

```
G.Edges
```

ans = 13×2 table

	EndNodes		Weight
1	'(1,3)'	'(5,6)'	7.2647
2	'(1,3)'	'(20,5)'	22.2733
3	'(1,3)'	'(15,2)'	18.6712
4	'(1,3)'	'(20,1)'	23.4007
5	'(5,6)'	'(20,5)'	15.1965
6	'(5,6)'	'(15,2)'	12.3089

	EndNodes		Weight
7	'(5,6)'	'(20,1)'	16.8252
8	'(20,5)'	'(15,2)'	10.2447
9	'(20,5)'	'(15,2)'	6.2286
10	'(20,5)'	'(20,1)'	9.0456
11	'(20,5)'	'(20,1)'	4.0494
12	'(15,2)'	'(20,1)'	8.0720
13	'(15,2)'	'(20,1)'	6.0253

```
G = rmedge(G,[13 8]);
%G = addedge(G,'(20,5)','(20,1)',9.04556);
p=plot(G,'EdgeLabel',G.Edges.Weight);
title('Fully Connected Network Non-directional');
[T,pred]=minspantree(G);
highlight(p,T)
```



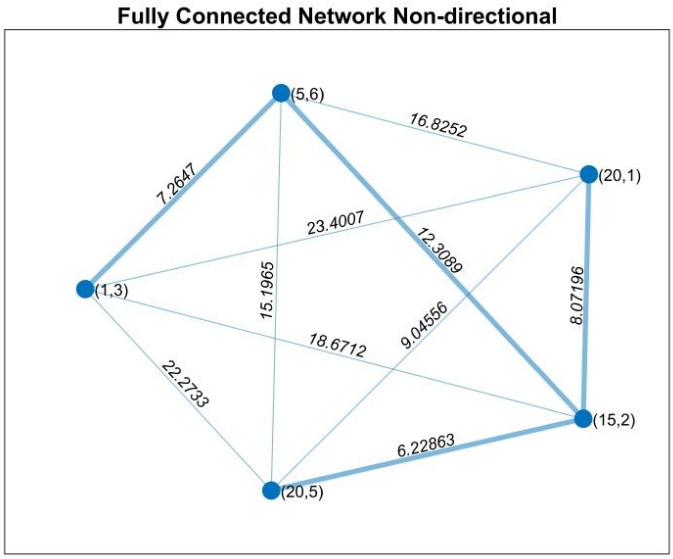

```
G.Edges
```

ans = 11×2 table

	EndNodes		Weight
1	'(1,3)'	'(5,6)'	7.2647
2	'(1,3)'	'(20,5)'	22.2733

	EndNodes		Weight
3	'(1,3)'	'(15,2)'	18.6712
4	'(1,3)'	'(20,1)'	23.4007
5	'(5,6)'	'(20,5)'	15.1965
6	'(5,6)'	'(15,2)'	12.3089
7	'(5,6)'	'(20,1)'	16.8252
8	'(20,5)'	'(15,2)'	6.2286
9	'(20,5)'	'(20,1)'	9.0456
10	'(20,5)'	'(20,1)'	4.0494
11	'(15,2)'	'(20,1)'	8.0720

```
G = rmedge(G,10);
%G = addedge(G,'(20,5)','(20,1)',4.04945);
p=plot(G,'EdgeLabel',G.Edges.Weight);
title('Fully Connected Network Non-directional');
[T,pred]=minspantree(G);
highlight(p,T)
```



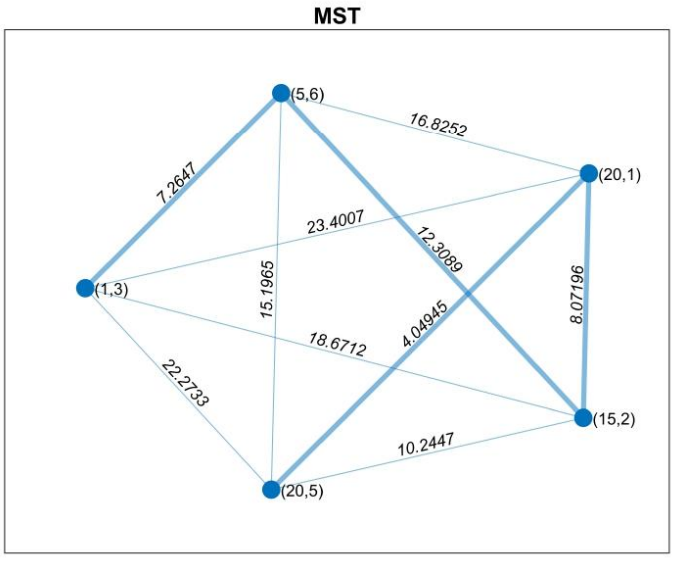

```
G.Edges
```

ans = 10×2 table

	EndNodes		Weight
1	'(1,3)'	'(5,6)'	7.2647
2	'(1,3)'	'(20,5)'	22.2733
3	'(1,3)'	'(15,2)'	18.6712
4	'(1,3)'	'(20,1)'	23.4007
5	'(5,6)'	'(20,5)'	15.1965
6	'(5,6)'	'(15,2)'	12.3089
7	'(5,6)'	'(20,1)'	16.8252
8	'(20,5)'	'(15,2)'	6.2286
9	'(20,5)'	'(20,1)'	9.0456
10	'(15,2)'	'(20,1)'	8.0720

```
G = rmedge(G,[9 8]);
G = addedge(G,'(20,5)','(20,1)',4.04945);
G = addedge(G,'(20,5)','(15,2)',10.2447);

p=plot(G,'EdgeLabel',G.Edges.Weight);
title('MST');
[T,pred]=minspantree(G);
highlight(p,T)
```



TSP

```
N=[[1 3],[-1 0];
    [5 6],[1 0];
    [20 5],[3 4]/norm([3 4]);
    [15 2],[-1 4]/norm([-1 4]);
    [20 1],[0 1]]
```

```
N = 5×4
    1.0000    3.0000   -1.0000         0
    5.0000    6.0000    1.0000         0
   20.0000    5.0000    0.6000    0.8000
   15.0000    2.0000   -0.2425    0.9701
   20.0000    1.0000         0    1.0000
```

```
distanceMatrix=zeros(5,5);
for i=1:size(N,1)
    start=N(i,:);
    for l=1:size(N,1)
        finish=N(l,:);
        [~,~,~,~,~,~,~,~,~,~,~,~,~,cost]=optpath(start(1:2),finish(1:2),start(3:4),finish(3:4),
        %optpath(p1,p2,h1,h2,r)
        if (l==i)
            cost=0;
        end
        distanceMatrix(i,l)=cost;
    end
end
distanceMatrix
```

```
distanceMatrix = 5×5
         0    7.2647   22.2733   18.6712   23.4007
    7.2647         0   15.1965   12.3089   16.8252
   20.7694   19.7940         0   10.2447    9.0456
   20.0309   13.5815    6.2286         0    8.0720
   24.9372   18.8536    4.0494    6.0253         0
```

Plot TSP

```
tourorder=[cell2mat(tours)]
```

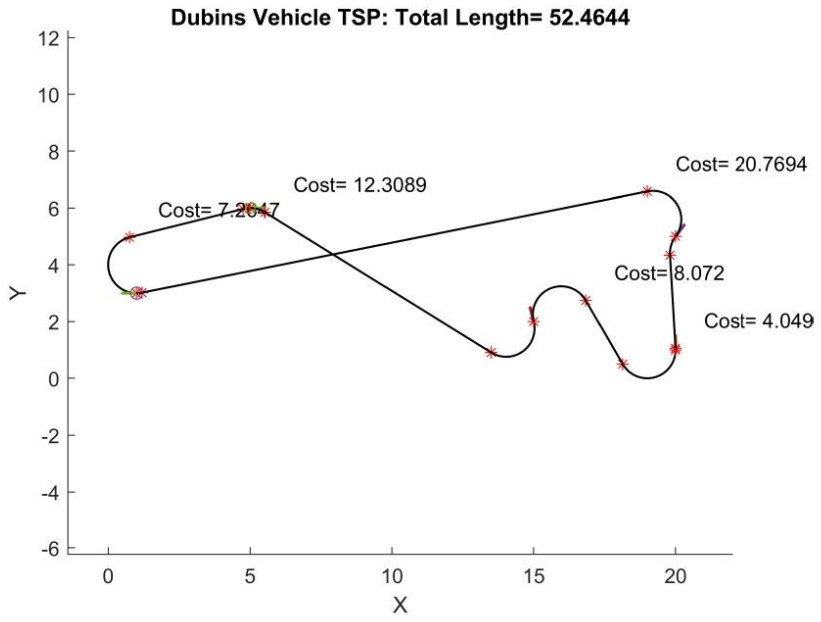
```
tourorder = 1×5
     2     4     5     3     1
```

```
totalcost=0;
figure(6);
clf;
hold on
startpoint=[0 0];
pointsec=[0 0];
for i=1:length(tourorder)
    if i==1
        p1=N(1,1:2);
        h1=N(1,3:4);
```

```
        p2=N(tourorder(1),1:2);
        h2=N(tourorder(1),3:4);
        startpoint=p1;
        pointsec=p2;
    else
        p1=N(tourorder(i-1),1:2);
        h1=N(tourorder(i-1),3:4);
        p2=N(tourorder(i),1:2);
        h2=N(tourorder(i),3:4);
    end
    figure(3);
    [solutiontype,startendcircle,CCCcircle,optpathpoints,theta1,theta2,theta12,theta3,theta4,th
    PLOT = Dubinsplot(solutiontype,startendcircle,CCCcircle,optpathpoints,theta1,theta2,theta1
    totalcost=totalcost+cost;
end



plot(startpoint(1),startpoint(2),'o')
plot(pointsec(1),pointsec(2),'s')
title(append('Dubins Vehicle TSP: Total Length= ',num2str(totalcost)))
hold off
```



Final Project 2
Joel Fielek
Roman Yoder

```
function
[solutiontype,startendcircle,CCCcircle,optpathpoints,theta1,theta2,theta12,theta3,theta4,theta34,thetaCCC1,t
hetaCCC2,thetaCCC12,cost] = optpath(p1,p2,h1,h2,r)

[circle1,circle2]=circlesCSC(p1, h1, r);
[circle3,circle4]=circlesCSC(p2, h2, r);
CSCcircles=[circle1;circle2;circle3;circle4];
% FROM 1; FROM 2;TO 1;TO 2

CSCgrouped = CSCgrouper(CSCcircles);
PathpointsCSC = CSCpathpoints(CSCgrouped,r);
try
   [totallengthCSC,theta1r,theta2r,theta12r,theta3r,theta4r,theta34r] =
CSClength(CSCgrouped,PathpointsCSC,CSCcircles,p1,p2,r);
   %[totallength,theta1r,theta2r,theta12r,theta3r,theta4r,theta34r] =
CSClength(CSCgrouped,pathpoints,CSCcircles,p1,p2,r)

   CCClengths=inf*ones(4,1);

   if norm((p2-p1))<(4*r);

      CCCgroupedCSCcircles=[CSCgrouped(1,:);CSCgrouped(4,:)];

      %CSCcircles=[circle1;circle2;circle3;circle4]
      %solve intermediatary circles
      [CCC,pathpointsCCC]=CCC_center(CCCgroupedCSCcircles,r);

[totallengthCCC,theta1rC,theta2rC,theta12rC,theta3rC,theta4rC,theta34rC,thetaCCC1r,thetaCCC2r,thetaC
CC12r]=CCClength(CCCgroupedCSCcircles,CCC,pathpointsCCC,p1,p2,r);
      %[totallength,theta1r,theta2r,theta12r,theta3r,theta4r,theta34r,thetaCCC1r,thetaCCC2r,thetaCCC12r] =
CCClength(CCCgroupedCSCcircles,CCC,points,p1,p2,r)
      if min(totallengthCCC)<min(totallengthCSC);
         solutiontype=1;
         [cost,I] = min(totallengthCCC);

selection=[CCCgroupedCSCcircles(1,:);CCCgroupedCSCcircles(1,:);CCCgroupedCSCcircles(2,:);CCCgroup
edCSCcircles(2,:)];

         startendcircle=selection(I,:);
         CCCcircle=CCC(I,:);
         optpathpoints=pathpointsCCC(I,:);
         theta1=theta1rC(I);
         theta2=theta2rC(I);
         theta12=theta12rC(I);
         theta3=theta3rC(I);
         theta4=theta4rC(I);
         theta34=theta34rC(I);
         thetaCCC1=thetaCCC1r(I);
```

```
        thetaCCC2=thetaCCC2r(I);
        thetaCCC12=thetaCCC12r(I);

    else
        solutiontype=0;
        [cost,I] = min(totallengthCSC);
        startendcircle=CSCgrouped(I,:);
        CCCcircle=['N/A ','N/A'];
        optpathpoints=PathpointsCSC(I,:);
        theta1=theta1r(I);
        theta2=theta2r(I);
        theta12=theta12r(I);
        theta3=theta3r(I);
        theta4=theta4r(I);
        theta34=theta34r(I);
        thetaCCC1='N/A';
        thetaCCC2='N/A';
        thetaCCC12='N/A';

        if min(totallengthCCC)==min(totallengthCSC)
            fprintf('Warning:There are equivilent lengthed solutons for CSC and CCC, defualt to CSC')
        end
    end

  else
    solutiontype=0;
    [cost,I] = min(totallengthCSC);
    startendcircle=CSCgrouped(I,:);
    CCCcircle=['N/A ','N/A'];
    optpathpoints=PathpointsCSC(I,:);
    theta1=theta1r(I);
    theta2=theta2r(I);
    theta12=theta12r(I);
    theta3=theta3r(I);
    theta4=theta4r(I);
    theta34=theta34r(I);
    thetaCCC1='N/A';
    thetaCCC2='N/A';
    thetaCCC12='N/A';


  end
catch
  fprintf('Warning:There is no solution for CSC defualting to CCC')
  CCClengths=inf*ones(4,1);
  CCCgroupedCSCcircles=[CSCgrouped(1,:);CSCgrouped(4,:)];
  %CSCcircles=[circle1;circle2;circle3;circle4]
  %solve intermediatary circles
```

```
   [CCC,pathpointsCCC]=CCC_center(CCCgroupedCSCcircles,r);

[totallengthCCC,theta1rC,theta2rC,theta12rC,theta3rC,theta4rC,theta34rC,thetaCCC1r,thetaCCC2r,thetaC
CC12r]=CCClength(CCCgroupedCSCcircles,CCC,pathpointsCCC,p1,p2,r);
   solutiontype=1;
   [cost,I] = min(totallengthCCC);

selection=[CCCgroupedCSCcircles(1,:);CCCgroupedCSCcircles(1,:);CCCgroupedCSCcircles(2,:);CCCgroup
edCSCcircles(2,:)];
   startendcircle=selection(I,:);
   CCCcircle=CCC(I,:);
   optpathpoints=pathpointsCCC(I,:);
   theta1=theta1rC(I);
   theta2=theta2rC(I);
   theta12=theta12rC(I);
   theta3=theta3rC(I);
   theta4=theta4rC(I);
   theta34=theta34rC(I);
   thetaCCC1=thetaCCC1r(I);
   thetaCCC2=thetaCCC2r(I);
   thetaCCC12=thetaCCC12r(I);
end
end

function PLOT =
Dubin'splot(solutiontype,startendcircle,CCCcircle,optpathpoints,theta1,theta2,theta12,theta3,theta4,theta34,t
hetaCCC1,thetaCCC2,thetaCCC12,cost,p1,p2,h1,h2,r)

hold on

%Start point
plot(p1(1),p1(2),'r*');
plot(p2(1),p2(2),'r*');

%headings
dp1 = h1;
dp2 = h2;

quiver(p1(1),p1(2),dp1(1),dp1(2),0.5,'LineWidth',1.5);
quiver(p2(1),p2(2),dp2(1),dp2(2),0.5,'LineWidth',1.5);

%pathpoints
plot(optpathpoints(1),optpathpoints(2),'r*');
plot(optpathpoints(3),optpathpoints(4),'r*');

%FIRST CIRCLE
if theta1<theta2
   angleInitial=theta1;
```

```
   angleFinal=theta2;
else
   angleInitial=theta2;
   angleFinal=theta1;
end
%
% if theta12>0&theta2<theta1&theta1==0
%     3
%     angleInitial=angleFinal;
%     angleFinal=2*pi;
% end

if theta12>0
   theta = linspace(angleInitial,angleFinal);
else
   theta = linspace(angleFinal,angleInitial);
end

if theta12>0&theta1>theta2
   theta = [linspace(theta1,2*pi),linspace(0,theta2)];
elseif theta12<0&theta2>theta1
   theta = [linspace(theta2,2*pi),linspace(0,theta1)];
end

%initial angle of the arc in degrees
%final angle of the arc in degrees
centre=startendcircle(1,1:2);                                   %centre of the arc
radius=r;                                       %radius of the arc
xcoords = centre(1)+radius*cos(theta);      %x coordinates
ycoords = centre(2)+radius*sin(theta);      % y coordinates
plot(xcoords,ycoords,'black','LineWidth',1);                    %plot the arc

%LAST CIRCLE

% if theta34>0&theta3>theta4&theta4==0
%
% end

if theta3<theta4&theta
   angleInitial=theta3;
   angleFinal=theta4;
else
   angleInitial=theta4;
   angleFinal=theta3;
end

% if theta34>0&theta3>theta4&theta4==0
%     31
```

```
%     angleInitial=angleFinal;
%     angleFinal=2*pi;
% end

if theta34>0
   theta = linspace(angleInitial,angleFinal);
else
   theta = linspace(angleFinal,angleInitial);
end

if theta34>0&theta3>theta4
   theta = [linspace(theta3,2*pi),linspace(0,theta4)];
elseif theta34<0&theta4>theta3
   theta = [linspace(theta4,2*pi),linspace(0,theta3)] ;
end
%initial angle of the arc in degrees
%final angle of the arc in degrees
centre=startendcircle(1,3:4);                                   %centre of the arc
radius=r;                                       %radius of the arc
xcoords = centre(1)+radius*cos(theta);      %x coordinates
ycoords = centre(2)+radius*sin(theta);      % y coordinates
plot(xcoords,ycoords,'black','LineWidth',1)                             %plot the arc

if solutiontype==0
   %CSC
   X=[optpathpoints(1),optpathpoints(3)];
   Y=[optpathpoints(2),optpathpoints(4)];
   plot(X,Y,'black','LineWidth',1);
elseif solutiontype==1
   %CCC
   if thetaCCC1<thetaCCC2
   angleInitial=thetaCCC1;
   angleFinal=thetaCCC2;
else
   angleInitial=thetaCCC2;
   angleFinal=thetaCCC1;
end

% if thetaCCC12>0&thetaCCC2>thetaCCC1&thetaCCC1==0
%     32
%     angleInitial=angleFinal;
%     angleFinal=2*pi;
% end

if thetaCCC12>0
   theta = linspace(angleInitial,angleFinal);
else
   theta = linspace(angleFinal,angleInitial);
```

```
end

if thetaCCC12>0&thetaCCC1>thetaCCC2
   theta = [linspace(thetaCCC1,2*pi),linspace(0,thetaCCC2)];
elseif thetaCCC12<0&thetaCCC2>thetaCCC1
   theta = [linspace(thetaCCC2,2*pi),linspace(0,thetaCCC1)];
end


   %initial angle of the arc in degrees
   %final angle of the arc in degrees
   centre=CCCcircle;                                  %centre of the arc
   radius=r;                                  %radius of the arc
%     theta = linspace(angleInitial,angleFinal);
   xcoords = centre(1)+radius*cos(theta);  %x coordinates
   ycoords = centre(2)+radius*sin(theta);   % y coordinates
   plot(xcoords,ycoords,'black','LineWidth',1)                              %plot the arc
end
title('Dubin's Vehicle');
xlabel('X');
ylabel('Y');
text(optpathpoints(1,1)+1,optpathpoints(1,2)+1,append('Cost=',' ',num2str(cost)));
axis equal;
axis padded;
PLOT=figure(1);
end
```