Dep. of Mechanical Eng.

Robotics

Dr. Saeed Behzadipour

Homework 7

Yashar Zafari
99106209

December 26, 2023

Robotics

Homework 7

Yashar Zafari
99106209

Questions

Path Planning using Road Mapping

We want to solve the problem of the previous homework using the road mapping method and Dijkstra's algorithm. To do this, write a program that:

- Reads the number of obstacles N, the coordinates of the two ends of the lines forming the obstacles (each obstacle is a line segment), the coordinates of the starting point, and the coordinates of the target point from a text file called "input.txt" in the current directory. The format of the text file is as follows:

```
N
Sx1, Sy1, Ex1, Ey1
...
SxN, SyN, ExN, EyN
X_start, Y_start
X_end, Y_end
```

- Calculates the shortest path and plots it along with all obstacles on a figure.

Solution

There is a script file in order to run the function and plot the results, called "`script`".

The representing text file of the previous homework in the format specified, is stored in the text file named "`input1`". I wrote "`PathDijkstraPoint`" function to find the shortest possible path through the obstacles from the start point to the final, which is as below:

```
function P = PathDijkstraPoint(dir)
% Reading the input text file
txtread=readmatrix(dir,"Range",[1 1]);
% Number of the obstacles' vertices, or in another words, the
   total
% number of the obstacles' sides.
numPoints=txtread(1,1);
% Storing the coordinates of the obstacles' vertices
obsverts=reshape(txtread(2:1+numPoints,:)',2,[]);
obs={}; % Pre-allocating a cell for clustering the obstacles.
numVert=size(obsverts,2);
startCol=1;
% Clustering the obstacles.
```

```
% If the first point is repeated again in the matrix, then
   from the start
% to that next repeatition index is for one obstacle, and then
    by
% truncating the matrix from start to that point, the same
   process is done
% to find the other obstacle.
while startCol <= numVert-1
    for endCol= startCol + 1:numVert
        if obsverts(:,startCol)==obsverts(:,endCol)
            obs{end+1}=obsverts(:,startCol:endCol);
            startCol=endCol;
            break;
        end
    end
    startCol=startCol+1;
end
numObs=size(obs,2);
start=txtread(end-1,[1 2])';
final=txtread(end,[1 2])';
% Storing the nodes of the graph
nodes=[start unique(obsverts',"rows")' final];
AdjMat=zeros(size(nodes,2));
flag=1;
% Findig the Adjacency Matrix for representing the graph
% In this matrix, a_{ij} represents the distance between the i
   -th node and
% j-th node. If it's zero, then there is no edge between those
    nodes. Also
% since the edges do not have direction, the matrix is
   symmetric having
% zeros on its diagonal.
for i=1:size(nodes,2)-1
    for j=i+1:size(nodes,2)
        for k=1:numObs
                        % Finding the intersection points of
                           the augmented obstacle and
            % the edge betweem the i-th node and j-th
            [xtmp,ytmp]=polyxpoly(nodes(1,[i j]),nodes(2,[i j
               ]),obs{k}(1,:),obs{k}(2,:),"unique");
            tmp=[xtmp';ytmp'];
                        % If it has more than one intersection
                            points, then it can be
            % one of these cases. Case 1: It goes through to
               obstacle,
            % which is not allowable. Case 2: It is one side
               of the
```

```
            % augmented obstacle which is allowable. To find
               this, I take
            % the mean of the intersection points, and find
               whether the
            % intesection points and their mean is inside or
               on the shape of
            % obstacle. If any of them is strictly inside the
               shape, then
            % it means that the corresponding edge passes
               through the
            % augmented obstacle.
            if  size(tmp,2)>1
                [in,on]=inpolygon([tmp(1,:) mean(tmp(1,:),"all
                   ")],[tmp(2,:) mean(tmp(2,:),"all")],obs{k
                   }(1,:),obs{k}(2,:));
                if any(in&~on)
                    flag=0;
                    break
                end
            end
        end
        if flag
            AdjMat(i,j)=norm(nodes(:,i)-nodes(:,j));
            AdjMat(j,i)=norm(nodes(:,i)-nodes(:,j));
        end
        flag=1;
    end
end
% Finding the shortest path in Adjacency Matrix using Dijkstra
    Algorithm
[P,dist]=dijkstraAdjMat(AdjMat,1,size(nodes,2));
P=nodes(:,P);
% Plotting the results
figure;
hold on;
grid off;
box on;
axis equal;
xlabel('X');
ylabel('Y');
xlim([min(nodes(1,:))-1 max(nodes(1,:))+1]);
ylim([min(nodes(2,:))-1 max(nodes(2,:))+1]);
title(['Path found by Dijkstra (Minimum Distance = ', num2str(
   round(dist,3)) ,')']);
scatter(start(1), start(2),'b','filled');
scatter(final(1), final(2),'b','filled');
for i = 1:numObs
```

```
    plot(obs{i}(1,:),obs{i}(2,:),'color','red','LineWidth'
        ,1.5);
end
plot(P(1,:),P(2,:),"LineStyle","--")
end
```

First since the obstacle vertices are not labeled for each obstacle, I should implement an algorithm to cluster the points in the input and form each obstacle as a single entity. I use the fact that in a closed shape if we start from a point and maintain our rotation direction and check the vertices, we will end up in the first vertex that we started the rotation from. Thus I used this method to cluster the points in the input to seperate obstacles.

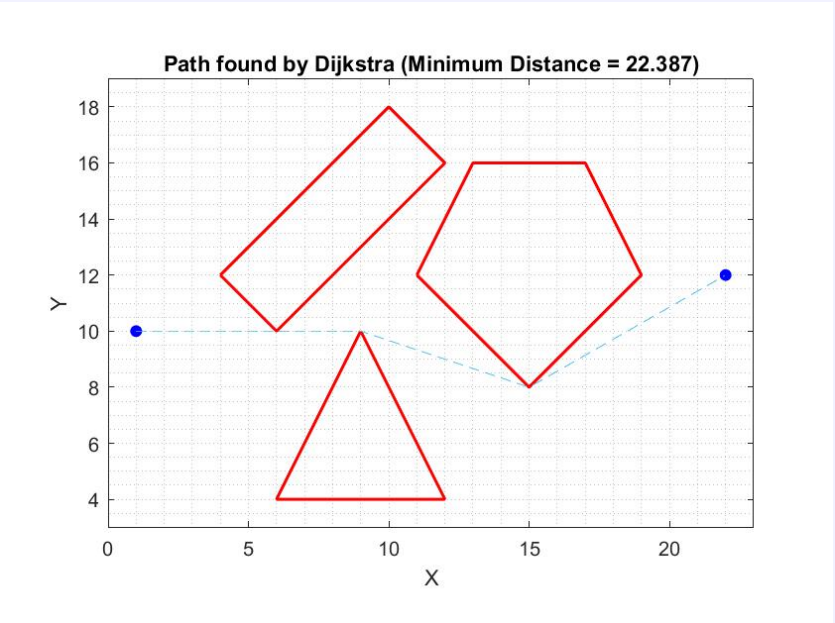
Now after forming "nodes", I find the Adjacency Matrix which represents the graph between the nodes with its possible edges. For finding this matrix, I select each pair of nodes, draw the line between them, check whether this line intersects any of the obstacles. If the intersection only occurs in one point, one the selected nodes is a vertex of the obstacles, or it passes through a vertex of another obstacle. These cases are acceptable as edges. But if the intersection occurs in more than one point, there are two cases again here. One possible case is that the selected pair is actually a pair of obstacle's vertices. Here we should check that if the pair is consecutive in the vertices of obstacle, because if the pair isn't consecutive, the edge drawn between them is actually inside the obstacle. The other case is that the considered edge actually passes through the obstacles, which isn't acceptable. Now if the edge between node i and node j isn't acceptable, a_{ij} is set to zero, else it's set to the distance of the two.

After forming the adjacency matrix, I use the following function to implement the Dijkstra Algorithm to find the shortest path between the start and the end point: kmlmlmlm

```
function [shortestPath,totalCost]=dijkstraAdjMat(adjMat,
    startNode,endNode)
numNodes=size(adjMat,1);
unvisited=true(1,numNodes); % All nodes are initially
    unvisited
% Initial distance from start to each node is infinite
dist=inf(1,numNodes);
% Array for storing the previous node visited in the shortest
    path,
% initialized with -1, which mean no previous node has been
    visited
prev=-1*ones(1,numNodes);
dist(startNode)=0; % Distance from start to itself is always 0
while any(unvisited)
    % Finding the node with the smallest distance
    nextNode=find(unvisited & dist==min(dist(unvisited)),1);
    unvisited(nextNode)=false; % Now marking it as visited
    % Iterating over neighbors
    for neighbor=find(adjMat(nextNode,:) & unvisited)
        newDist=dist(nextNode)+adjMat(nextNode,neighbor);
        if newDist<dist(neighbor) % Checking if the new path
```

```
            is shorter
             dist(neighbor)=newDist;
             prev(neighbor)=nextNode; % Updating previous node
        end
    end
end
% Tracing the shortest path from end node to start node
% If still the value of the end node is infinite, then no
   shortest path is
% found.
if isinf(dist(endNode))
    shortestPath=[];
    totalCost=inf;
    disp('No path exists between the selected nodes');
else
    totalCost=dist(endNode);
    shortestPath=endNode;
    while prev(shortestPath(1))>0
        shortestPath=[prev(shortestPath(1)),shortestPath];
    end
end
end
```

First every node is unvisited and the distances are initialized to infinty except the start node being zero. Next the shortest distance is found and marked as visited. By checking the neighbors, it checks other paths too, and if the new path is shorter, it updates the previous node. Using a "`while`" loop, the process continues until it reaches the final node and no unvisited node is left. Now by checking the values of the previous node, it backtracks the path to the start point.
Here below is the result, which the shortest path found from start point to the final point is 22.387 units:



Bonus Question (25%)

Assume the robot geometry is modeled as a non-rotating convex polygon with m edges. The coordinates of the m vertices of the robot in the initial point are given at the end of the input.txt file:

```
m
X1, Y1
...
Xm, Ym
```

Again, find and plot the shortest path from the starting point to the end using obstacle modification and Dijkstra's algorithm.

Solution

The representing text file of the previous homework in the format specified, is stored in the text file named "`input2`". I wrote "`PathDijkstraShaped`" function to find the shortest possible path through the obstacles from the start point to the final, which is as below:

```
function P = PathDijkstraShaped(dir)
% Reading the input text file
txtread=readmatrix(dir,"Range",[1 1]);
% Number of the obstacles' vertices, or in another words, the
   total
% number of the obstacles' sides.
numPoints=txtread(1,1);
% Storing the coordinates of the obstacles' vertices
obsverts=reshape(txtread(2:1+numPoints,:)',2,[]);
% Truncating the data matrix since I already stored the
   obstacles' data
start=txtread(2+numPoints,[1 2])';
final=txtread(3+numPoints,[1 2])';
txtread=txtread(4+numPoints:end,:);
% Number of the vertices of the  robot geometry
numVert=txtread(1,1);
% Storing the coordinates of the robot's vertices
vertices=reshape(txtread(2:1+numVert,[1 2])',2,[]);
obs={}; % Pre-allocating a cell for clustering the obstacles.
numVertx=size(obsverts,2);
startCol=1;
% Clustering the obstacles.
% If the first point is repeated again in the matrix, then
   from the start
% to that next repeatition index is for one obstacle, and then
    by
% truncating the matrix from start to that point, the same
   process is done
% to find the other obstacle.
while startCol <= numVertx-1
```

```
    for endCol= startCol + 1:numVertx
        if obsverts(:,startCol)==obsverts(:,endCol)
            obs{end+1}=obsverts(:,startCol:endCol);
            startCol=endCol;
            break;
        end
    end
    startCol=startCol+1;
end
numObs=size(obs,2);
% Finding the vectors from the robot's vertices to the start
   point which
% will be used for augmenting the obstacles using the
   Minkowski method
vectors=-vertices+start.*[1 1 1];
aug=cell(1,numObs);
for i=1:numObs
    % Augmentation of the obstacles using the Minkowski method
    augObsTmp=unique(reshape(repmat(obs{i},[1 1 numVert])+
       permute(repmat(vectors,[1 1 size(obs{i},2)]),[1 3 2])
       ,2,[])','rows')';
    % Finding the convex hull formed by the new generated
       point by the
    % augmentation.
    k=ConvexHullInd(augObsTmp);
    aug{i}=augObsTmp(:,k);
end
% Storing the nodes of the graph
nodes=[start cell2mat(aug) final];
AdjMat=zeros(size(nodes,2));
flag=1;
% Findig the Adjacency Matrix for representing the graph
% In this matrix, a_{ij} represents the distance between the i
   -th node and
% j-th node. If it's zero, then there is no edge between those
    nodes. Also
% since the edges do not have direction, the matrix is
   symmetric having
% zeros on its diagonal.
for i=1:size(nodes,2)-1
    for j=i+1:size(nodes,2)
        for k=1:numObs
            % Finding the intersection points of the augmented
                obstacle and
            % the edge betweem the i-th node and j-th
            [xtmp,ytmp]=polyxpoly(nodes(1,[i j]),nodes(2,[i j
               ]),aug{k}(1,:),aug{k}(2,:),"unique");
```

```
            tmp=[xtmp';ytmp'];
            % If it has more than one intersection points,
               then it can be
            % one of these cases. Case 1: It goes through to
               obstacle,
            % which is not allowable. Case 2: It is one side
               of the
            % augmented obstacle which is allowable. To find
               this, I take
            % the mean of the intersection points, and find
               whether the
            % intesection points and their mean is inside or
               on the shape of
            % obstacle. If any of them is strictly inside the
               shape, then
            % it means that the corresponding edge passes
               through the
            % augmented obstacle.
            if size(tmp,2)>1
                [in,on]=inpolygon([tmp(1,:) mean(tmp(1,:),"all
                   ")],[tmp(2,:) mean(tmp(2,:),"all")],aug{k
                   }(1,:),aug{k}(2,:));
                if any(in&~on)
                    flag=0;
                    break
                end
            end
        end
        if flag
            AdjMat(i,j)=norm(nodes(:,i)-nodes(:,j));
            AdjMat(j,i)=norm(nodes(:,i)-nodes(:,j));
        end
        flag=1;
    end
end
% Finding the shortest path in Adjacency Matrix using Dijkstra
    Algorithm
[P,dist]=dijkstraAdjMat(AdjMat,1,size(nodes,2));
P=nodes(:,P);
animateRobotMotion(P,dist,nodes,vertices,start,final,aug,obs);
end

%% Plotting the movement of the robot

function animateRobotMotion(P, dist, nodes, vertices, start,
   final, aug, obstacles)
fig=figure;
```

```
axis equal;
hold on;
grid minor;
box on;
xlabel('X');
ylabel('Y');
title(['Robot Motion Animation. Minimum Distance: ' num2str(
   round(dist,3))]);
xlim([min([nodes(1,:),vertices(1,:)])-1,max([nodes(1,:),
   vertices(1,:)])+1]);
ylim([min([nodes(2,:),vertices(2,:)])-1,max([nodes(2,:),
   vertices(2,:)])+1]);
for i = 1:size(obstacles,2)
    plot(obstacles{i}(1,:),obstacles{i}(2,:),'k-','LineWidth'
       ,2);
    plot(aug{i}(1,:),aug{i}(2,:),'r--','LineWidth',0.5);
end
plot(P(1,:),P(2,:))
robotPlot=fill(vertices(1,:),vertices(2,:),'cyan');
scatter(start(1),start(2),'bo','filled');
scatter(final(1),final(2),'ro','filled');
pause(1);
for i = 1:length(P)-1
    stepVector=(P(:,i+1)-P(:,i))/50;
    for j = 1:50
        currentPos=get(robotPlot,'Vertices');
        newPos=currentPos'+stepVector.*[1 1 1];
        set(robotPlot, 'Vertices', newPos');
        pause(0.05);
        if ~ishghandle(fig)
            return;
        end
    end
end
end
```

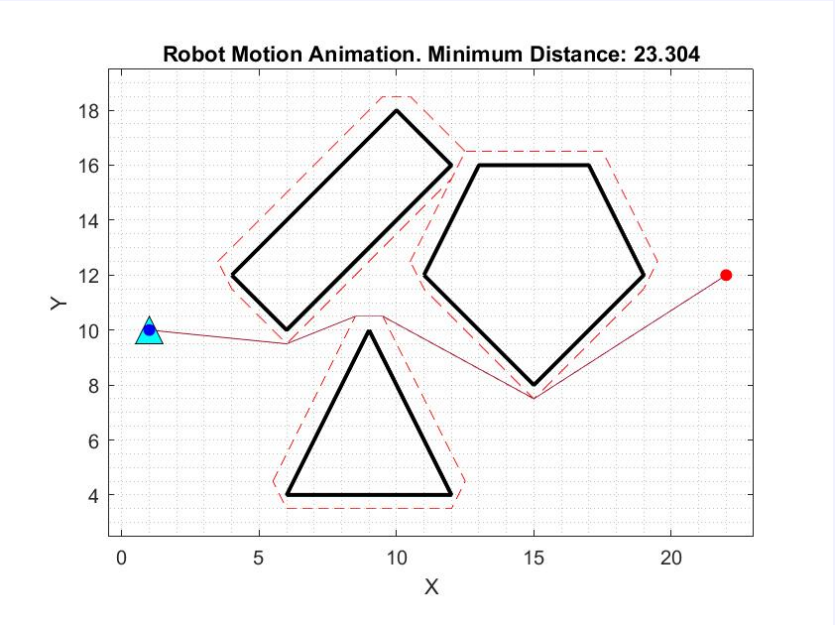
The procedure of this code is almost the same as "`PathDijkstraPoint`". Here after finding the vectors from the vertices of the robot's geometry to the start point, the augmented obstacles are generated using the Minkowski method to avoid the collision. In the augmentation process, the convex hull of the generated points should be found, which "`ConvexHullInd`" does so:

```
function hullInd = ConvexHullInd(points)
numPoints=size(points,2);
hullInd=[];
[~, startIdx]=min(points(1,:)); % Finding the leftmost point
currentIdx=startIdx;
hullPoints=[];
```

```
repeat=true;
while repeat
    hullPoints(end+1)=currentIdx; % Adding index to hull
       points
    nextIdx=mod(currentIdx,numPoints)+1; % Starting from the
       next point
    for i = 1:numPoints
        if i == currentIdx
            continue;
        end
        % Using cross product to check if it turn left,
           wraping the hull
        direction = det([points(:,nextIdx)-points(:,currentIdx
           ), points(:,i)-points(:,currentIdx)]);
        if direction > 0
            nextIdx = i;
        end
    end
    currentIdx = nextIdx;
    % Checking if we have wrapped around to the first hull
       point
    if nextIdx == startIdx
        repeat = false;
    end
end
% Adding the first hull point to the end of the array to get
   wrapped convex
% hull
hullInd = [hullPoints hullPoints(1)];
end
```

In this function, first the left most point or in another words, the point with least x-axis coordinate is found. Next by selecting the another point, the cross product of the vector from the first point to the second point and the vector from the first point to the other points is calculated and by the sign of its result(positive being for counter-clockwise), we determine that the other points are on the left side of the first vector. In this manner we wrap the points in the counter-clockwise direction and create the convex hull.
Here is the result of this section in which the robot has a geometry and shortest path

found has the distance of 23.304 units:



Note that the animation of robot's movement can be watched in the live script file "`script`".