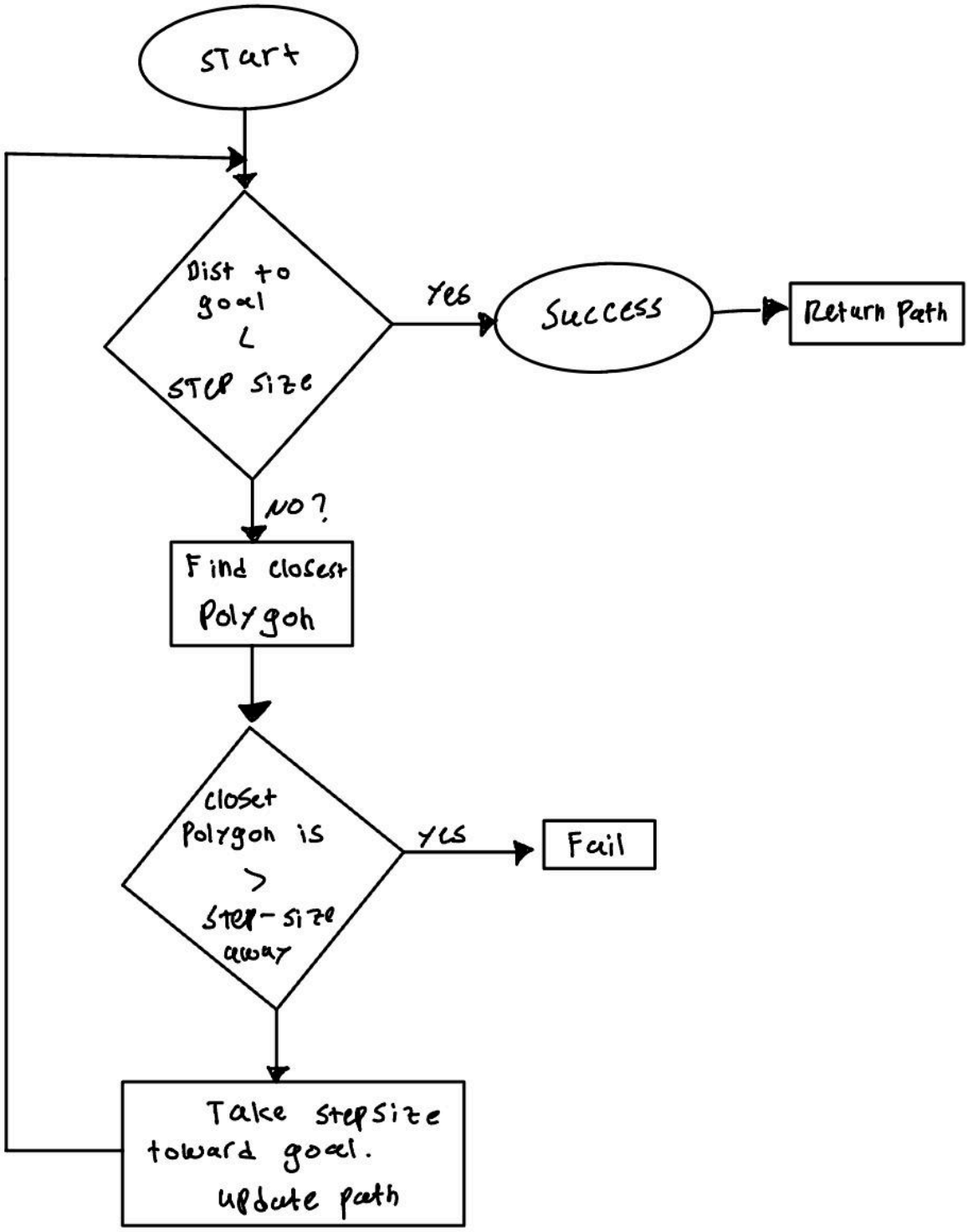
ME145 Robotic Planning and Kinematic: Lab 2

Bug 1 Algorithm

Elijah Perez

Winter 2025, Feb 27

Sketched Flow chart:

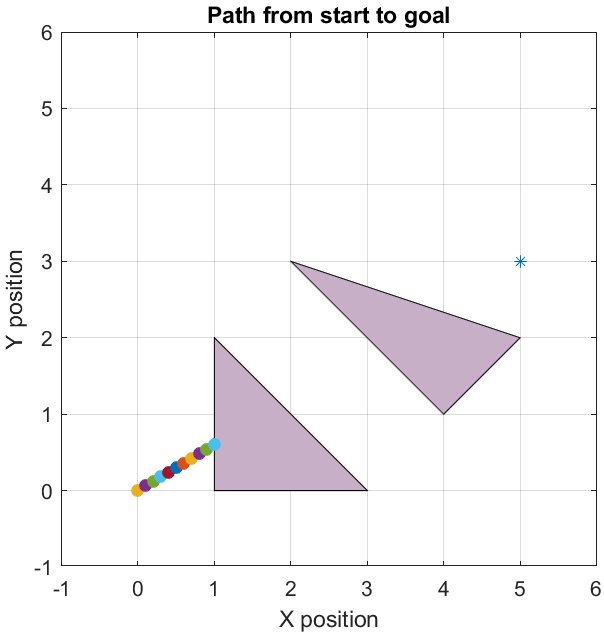


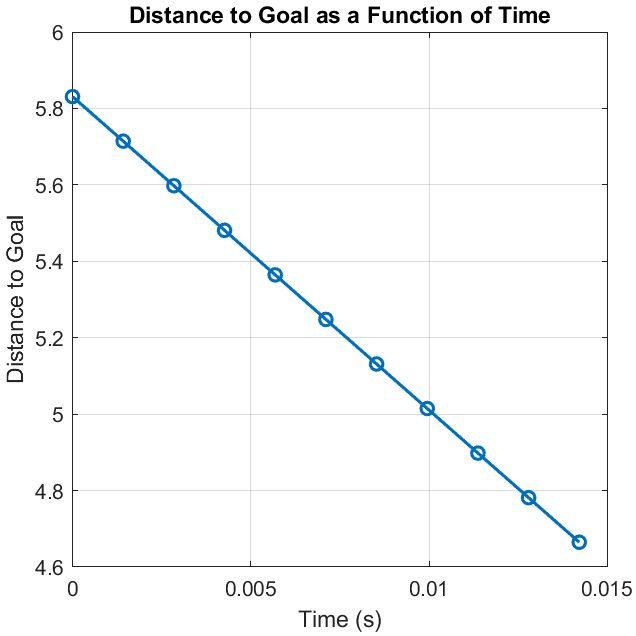
BugBase Implementation:

```
function Path = computeBug(Pstart,Pgoal,Obsticle,stepsize)
    %% initializing
    Pcurrent = Pstart;
    Path = Pstart;
    CurrentObsticle = 0;

    while abs(Pcurrent(1,1) - Pgoal(1,1)) >  stepsize*1.5 && abs(Pcurrent(1,2) - Pgoal(1,2)) > stepsize*1.5
        if length(Path(:,1))>=2
            fprintf("Fail: An obsticle as been hit")
        break
        else
        end

        %% main line intersection to line segment

            for i = 1:length(Obsticle(:,1,1))+1
            Counter(i) = i ;
            end
            Counter(end) = 1;

        for j = 1:length(Obsticle(1,1,:)) %checking each obsticles distance
            if j == CurrentObsticle
                if j == length(Obsticle(1,1,:))
                j= j+1;
                end

            else
                for i = 1:length(Obsticle(:,1,1)) % Iterating through each obsticles segment

                    x1 = Pcurrent(1);
                    y1 = Pcurrent(2);
                    x2 = Pgoal(1);
                    y2 = Pgoal(2);
                    P1 = Obsticle(Counter(i),:,j);
                    P2 = Obsticle(Counter(i+1),:,j);

                    x3 = P1(1);
                    y3 = P1(2);
                    x4 = P2(1);
                    y4 = P2(2);

                    denominator = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);

                    % Checking for parallel lines
                    if denominator == 0
                        intersection(i,:,j) = [NaN NaN];
                        d(i,j) = 0;
                    end

                    t = ((x1 - x3) * (y3 - y4) - (y1 - y3) * (x3 - x4)) / denominator;
```

```
                for i = length(Path(:,1)): length(Path(:,1))+length(x)-1
                    Path(i,1) = x(n);
                    Path(i,2) = Pcurrent_to_PNext(x(n));
                    n = n+1;
                end

            break
            else
            end
            %% Going from point A to point B
            % Pcurrent = Path(end,:);

            if PNext == Pcurrent
                PNext    = [x2 y2];
                return
            end

            [a,b,c] = computeLineThroughTwoPoints(Pcurrent,PNext);
            x1 = Pcurrent(1);
            y1 = Pcurrent(2);
            x2 = PNext(1);
            y2 = PNext(2);
            PositionX = [x1,x2];

            Delta = sqrt((x1-x2)^2 + (y1 -y2)^2)/stepsize;
            x = linspace(min(PositionX),max(PositionX),Delta);
            Pcurrent_to_PNext = @(x) (a/b)*x+c;  % Direct line from start to goal

            n = 1;
            for i = length(Path(:,1)): length(Path(:,1))+length(x)-1
                Path(i,1) = x(n);
                Path(i,2) = Pcurrent_to_PNext(x(n));
                n = n+1;
            end

            Pcurrent = Path(end,:);

        end
    end
```

```
                    t = ((x1 - x3) * (y3 - y4) - (y1 - y3) * (x3 - x4)) / denominator;
                    u = -((x1 - x2) * (y1 - y3) - (y1 - y2) * (x1 - x3)) / denominator;

                    if u >= 0 && u <= 1
                        intersect_x = x1 + t * (x2 - x1);
                        intersect_y = y1 + t * (y2 - y1);
                        intersection(i,:,j) = [intersect_x, intersect_y];
                        d(i,j) = norm([x1 y1]-[intersect_x intersect_y]);
                    else
                        intersection(i,:,j) = [NaN NaN]; % Void lol
                        d(i,j) = NaN;
                    end
                end

                if all(isnan(d(:,j)), 'all')
                    Obsticle_Hit =  [NaN NaN];
                else                                % If obstecle has been hit
                    finder = find(min(d(:,j)) == d(:,j));
                    Obsticle_Hit = intersection(finder,:,j);% I dentifiy where hit is at
                    Obsticle_Hit = Obsticle_Hit(1,:);
                    PNext = Obsticle_Hit;
                    CurrentObsticle = j;
                    CurrentSegmentNumber = [Counter(finder) Counter(finder+1)] ;

                    if sqrt((Pgoal(1)-Pcurrent(1))^2+(Pgoal(2)-Pcurrent(2))^2) <= sqrt((PNext(1)-Pcurrent(1))^2+(PNext(2)-Pcurrent(2))^2)
                        Obsticle_Hit =  [NaN NaN];
                    else
                    end

                break
                end
            end
        end

        if isnan(Obsticle_Hit)
            PNext = Pgoal;
            [a,b,c] = computeLineThroughTwoPoints(Pcurrent,PNext);

            x1 = Pcurrent(1);
            y1 = Pcurrent(2);
            x2 = PNext(1);
            y2 = PNext(2);
            PositionX = [x1,x2];

            Delta = sqrt((x1-x2)^2 + (y1 -y2)^2)/stepsize;
            x = linspace(min(PositionX),max(PositionX),Delta);

            Pcurrent_to_PNext = @(x) (a/b)*x+c;  % Direct line from start to goal

            n = 1;
             for i = length(Path(:,1)): length(Path(:,1))+length(x)-1
```





Input for bug base:

```
clc
close all
clear

Pstart = [0 0];
Pgoal  = [5 3];
P(:,:,1) = [1 0; 1 2 ; 3 0];
P(:,:,2) = [2 3; 4 1; 5 2];
stepsize = 0.1;

tic;
Path = computeBug(Pstart,Pgoal,P,stepsize);
```

Output: Plus above graphs

P	*3x2x2 double*
Path	*11x2 double*
Pgoal	[5,3]
Pstart	[0,0]
stepsize	0.1000

```
Fail: An obsticle as been hitTotal Path Length: 1.17 units
Computation Time: 0.0121 seconds
>>
```

ii) The BugBase algorithm detects obstacles in its path but fails when one is encountered. In contrast, the Bug 1 algorithm identifies an obstacle in its path and, upon reaching it, circumnavigates the obstacle to find the shortest route to the goal. To achieve this, it leverages previously implemented functions, such as computeTangentVectorToPolygon. This function calculates key parameters, including the minimum segment distance, minimum vertex distance, and the U vector. The U vector is crucial for navigating around obstacles, as it runs parallel to segments and turns at vertices. Once the algorithm has fully circled the obstacle, it returns to the point on the obstacle that provides the shortest path to the goal. From there, it continues toward the goal. If another obstacle is detected, the process repeats.

Bug1 implementation:

Inputs:

```
clc
close all
clear

Pstart = [0 0];
Pgoal  = [5 3];
P(:,:,1) = [1 0; 1 2 ; 3 0];
P(:,:,2) = [2 3; 4 1; 5 2];
stepsize = 0.1;

tic;
Path = computeBug1(Pstart,Pgoal,P,stepsize);
```

Outputs:

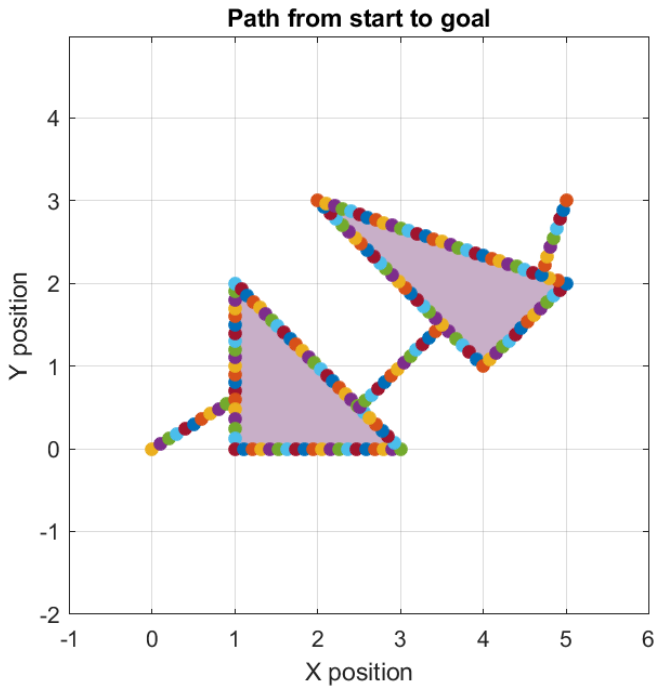
computation_time	0.3820
P	*3x2x2 double*
Path	*245x2 double*
Pgoal	[5,3]
Pstart	[0,0]
stepsize	0.1000

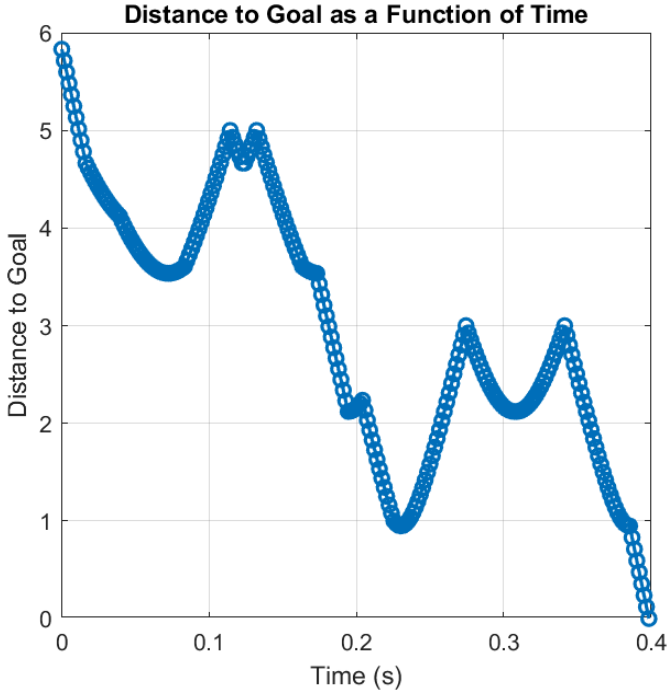
Graphs are below code.

```
function Path = computeBug1(Pstart,Pgoal,Obsticle,stepsize)
    %% initializing
    Pcurrent = Pstart;
    Path = Pstart;
    CurrentObsticle = 0;

    while abs(Pcurrent(1,1) - Pgoal(1,1)) >  stepsize*1.5 && abs(Pcurrent(1,2) - Pgoal(1,2)) > stepsize*1.5
        %% main line intersection to line segment

            for i = 1:length(Obsticle(:,1,1))+1
            Counter(i) = i ;
            end
            Counter(end) = 1;

        for j = 1:length(Obsticle(1,1,:))
            if j == CurrentObsticle
                if j == length(Obsticle(1,1,:))
                j= j+1;
                end

            else
                for i = 1:length(Obsticle(:,1,1))
                    P1 = Obsticle(Counter(i),:,j);
                    P2 = Obsticle(Counter(i+1),:,j);

                    x1 = Pcurrent(1);
                    y1 = Pcurrent(2);
                    x2 = Pgoal(1);
                    y2 = Pgoal(2);
                    x3 = P1(1);
                    y3 = P1(2);
                    x4 = P2(1);
                    y4 = P2(2);

                    denominator = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);

                    % Checking for parallel lines
                    if denominator == 0
                        intersection(i,:,j) = [NaN NaN];
                        d(i,j) = 0;
                    end

                    t = ((x1 - x3) * (y3 - y4) - (y1 - y3) * (x3 - x4)) / denominator;
                    u = -((x1 - x2) * (y1 - y3) - (y1 - y2) * (x1 - x3)) / denominator;
                    if u >= 0 && u <= 1
                        intersect_x = x1 + t * (x2 - x1);
                        intersect_y = y1 + t * (y2 - y1);
                        intersection(i,:,j) = [intersect_x, intersect_y];
                        d(i,j) = norm([x1 y1]-[intersect_x intersect_y]);
                    else
                        intersection(i,:,j) = [NaN NaN]; % Void lol
                        d(i,j) = NaN;
                    end
                end

                if all(isnan(d(:,j)), 'all')
                    Obsticle_Hit =  [NaN NaN];
                else
                    finder = find(min(d(:,j)) == d(:,j));
                    Obsticle_Hit = intersection(finder,:,j);% I dentifiy where hit is at
                    Obsticle_Hit = Obsticle_Hit(1,:);
                    PNext = Obsticle_Hit;
                    CurrentObsticle = j;
                    CurrentSegmentNumber = [Counter(finder) Counter(finder+1)] ;

                    if sqrt((Pgoal(1)-Pcurrent(1))^2+(Pgoal(2)-Pcurrent(2))^2) <= sqrt((PNext(1)-Pcurrent(1))^2+(PNext(2)-Pcurrent(2))^2)
                        Obsticle_Hit =  [NaN NaN];
                    else
                    end

                break
                end
            end
        end

        if isnan(Obsticle_Hit)
            PNext = Pgoal;
            [a,b,c] = computeLineThroughTwoPoints(Pcurrent,PNext);

            x1 = Pcurrent(1);
            y1 = Pcurrent(2);
            x2 = PNext(1);
            y2 = PNext(2);
            PositionX = [x1,x2];

            Delta = sqrt((x1-x2)^2 + (y1 -y2)^2)/stepsize;
            x = linspace(min(PositionX),max(PositionX),Delta);

            Pcurrent_to_PNext = @(x) (a/b)*x+c;  % Direct line from start to goal

```

```
            n = 1;
             for i = length(Path(:,1)): length(Path(:,1))+length(x)-1
                Path(i,1) = x(n);
                Path(i,2) = Pcurrent_to_PNext(x(n));
                n = n+1;
            end

        break
        else
        end
        %% Going from point A to point B
        % Pcurrent = Path(end,:);

        if PNext == Pcurrent
            PNext    = [x2 y2];
        end

        [a,b,c] = computeLineThroughTwoPoints(Pcurrent,PNext);
        x1 = Pcurrent(1);
        y1 = Pcurrent(2);
        x2 = PNext(1);
        y2 = PNext(2);
        PositionX = [x1,x2];

        Delta = sqrt((x1-x2)^2 + (y1 -y2)^2)/stepsize;
        x = linspace(min(PositionX),max(PositionX),Delta);
        Pcurrent_to_PNext = @(x) (a/b)*x+c;  % Direct line from start to goal

        n = 1;
        for i = length(Path(:,1)): length(Path(:,1))+length(x)-1
            Path(i,1) = x(n);
            Path(i,2) = Pcurrent_to_PNext(x(n));
            n = n+1;
        end

        Pcurrent = Path(end,:);

        if abs(Pcurrent(1,1) - Pgoal(1,1)) <  stepsize*1.5 && abs(Pcurrent(1,2) - Pgoal(1,2)) < stepsize*1.5
            break
        else
        end

        %% Going around object
        c=0;
        n=0;
        for m = CurrentSegmentNumber(1):length(Obsticle(:,1,1))+CurrentSegmentNumber(1)
            n = n+1;
            if m >= length(Counter)
                c=c+1;
                Order(n) = c;
            else
                Order(n) = m;
            end
        end


        for i = 1:length(Obsticle(:,1,1))

            x1 = Obsticle(Order(i),1,CurrentObsticle); % point 1
            y1 = Obsticle(Order(i),2,CurrentObsticle);
            x2 = Obsticle(Order(i+1),1,CurrentObsticle); % point 2
            y2 = Obsticle(Order(i+1),2,CurrentObsticle);
            PositionY = [y1,y2];

            q = Obsticle_Hit;
            S = [x2,y2] - [x1,y1];
            W = q-[x1,y1];
            Z = (dot(W,S))/(dot(S,S)); % scalar 0-1 if with in segment

            k=0;
            if x1 ==x2 % verticle
                Delta = sqrt((x1-x2)^2 + (y1 -y2)^2)/stepsize;
                y = (max(PositionY))-((max(PositionY)-min(PositionY))*(1-Z)):stepsize:max(PositionY);

                for n = length(Path(:,1)): length(Path(:,1))+length(y)-1
                    k = k+1;
                    Path(n,1) = x1;
                    Path(n,2) =y(k);
                end
            else % None verticle
                Pcurrent = [Path(end,1) Path(end,2)];
                PNext    = [x2 y2];

                if PNext == Pcurrent
                    PNext  = [x1 y1];
                end

                [a,b,c] = computeLineThroughTwoPoints(Pcurrent,PNext);

                x1 = Pcurrent(1);
```

```
                x1 = Pcurrent(1);
                y1 = Pcurrent(2);
                x2 = PNext(1);
                y2 = PNext(2);

                Delta = sqrt((x1-x2)^2 + (y1 -y2)^2)/stepsize;
                x = linspace(x1,x2,Delta);
                Pcurrent_to_PNext = @(x) (a/b)*x+c;  % Direct line from start to goal

                k =0;
                for N = length(Path(:,1)): length(Path(:,1))+length(x)-1
                    k = k+1;
                    Path(N,1) = x(k);
                    Path(N,2) = Pcurrent_to_PNext(x(k));
                end
            end
        end
        %% Going to Hit spot to complete 1 loop around object

        Pcurrent = [Path(end,1) Path(end,2)];
        PNext    = Obsticle_Hit;
        [a,b,c] = computeLineThroughTwoPoints(Pcurrent,PNext);

        x1 = Pcurrent(1);
        y1 = Pcurrent(2);
        x2 = PNext(1);
        y2 = PNext(2);
        PositionY = [y1,y2];

        if x1==x2
            Delta = sqrt((x1-x2)^2 + (y1 -y2)^2)/stepsize;
            y = linspace(min(PositionY),max(PositionY),Delta);
            k =0;
            for n = length(Path(:,1)): length(Path(:,1))+length(y)-1
                k = k+1;
                Path(n,1) = x1;
                Path(n,2) =y(k);
            end

        else
            Delta = sqrt((x1-x2)^2 + (y1 -y2)^2)/stepsize;
            x = linspace(x1,x2,Delta);
            Pcurrent_to_PNext = @(x) (a/b)*x+c;  % Direct line from start to goal

            k =0;
            for N = length(Path(:,1)): length(Path(:,1))+length(x)-1
                k = k+1;
                Path(N,1) = x(k);
                Path(N,2) = Pcurrent_to_PNext(x(k));
            end
        end
        %% Going to closest point from object to goal
        P = Obsticle(:,:,CurrentObsticle);
        [~,~,JumpPoint] = computeDistancePointToPolygon(P,Pgoal);

        X = 0;
        Y = 0;
        A = length(Path(:,1));
        B = length(Path(:,1));

        threshold = stepsize*1;
        step = Pgoal;
        JumpPoint = JumpPoint(1,:);
        while abs(step(1,1) - JumpPoint(1,1)) >  threshold || abs(step(1,2) - JumpPoint(1,2)) > threshold
            A = A+1;
            Path(A,1) = Path(B-X,1) ;
            Path(A,2) = Path(B-X,2) ;
            X = X+1;
            Y = Y+1;
            step = Path(A,:);
        end

        PNext = Pgoal;
        Pcurrent = JumpPoint;
    end
end
```


Path from start to goal
Y position
X position
4
3
2
1
0
-1
-2
-1
0
1
2
3
4
5
6


Distance to Goal as a Function of Time
Distance to Goal
Time (s)
6
5
4
3
2
1
0
0
0.1
0.2
0.3
0.4