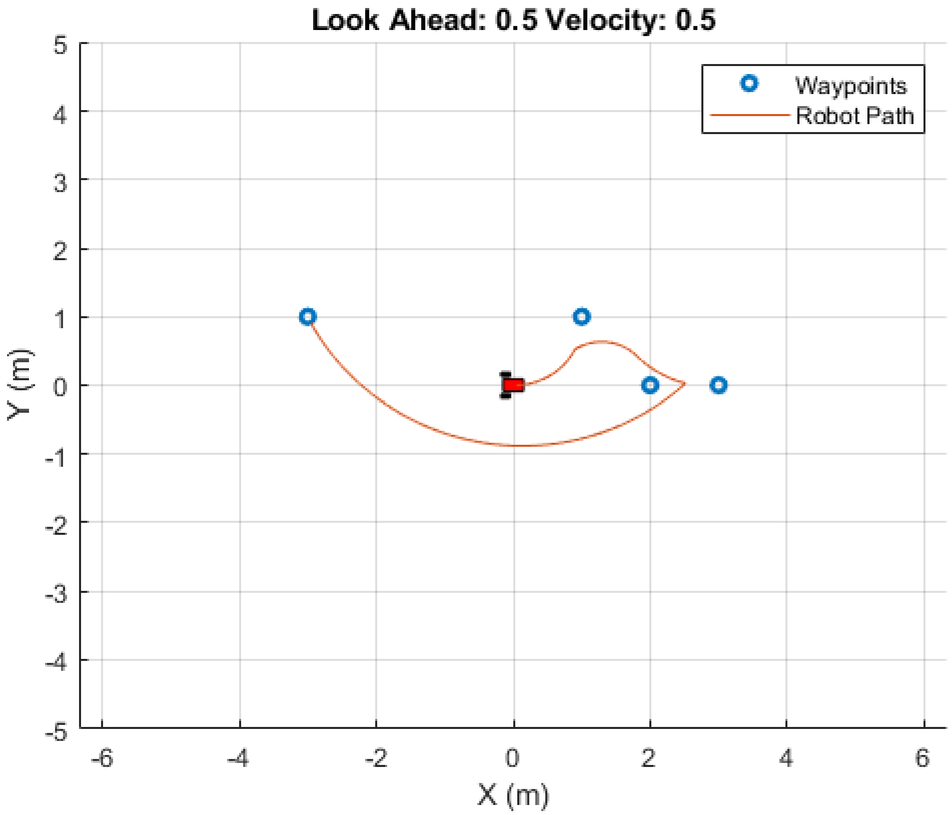
Table of Contents

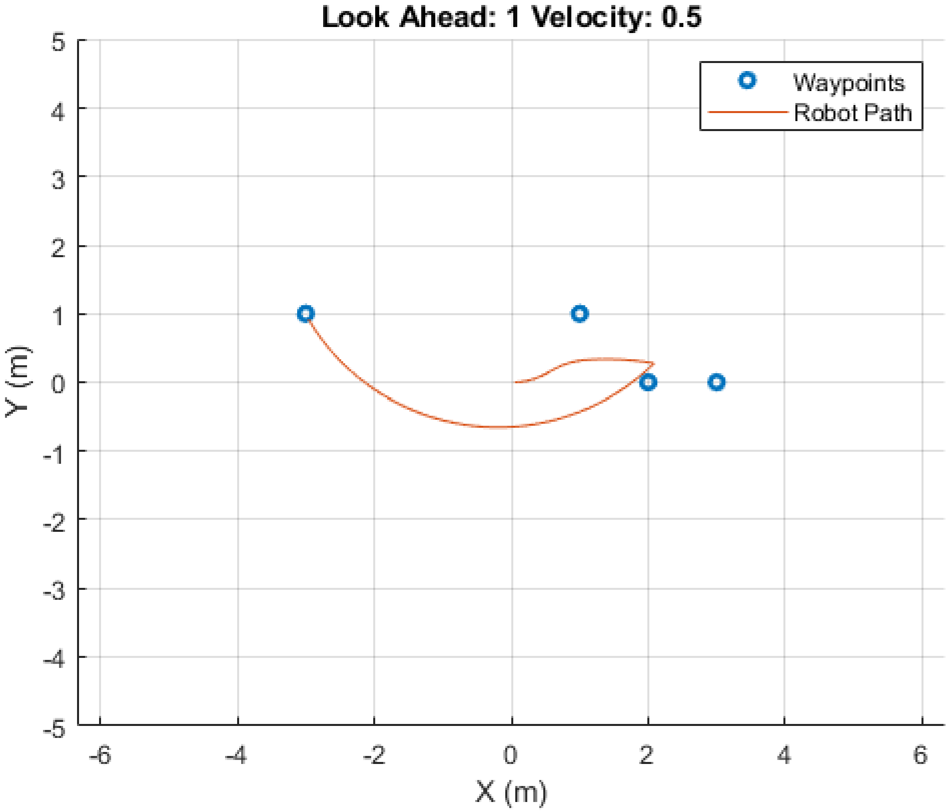
.... 1
Draw Robot at initial pose 2
Functions 9
robot animation function 9
Functions that you need to implement 14

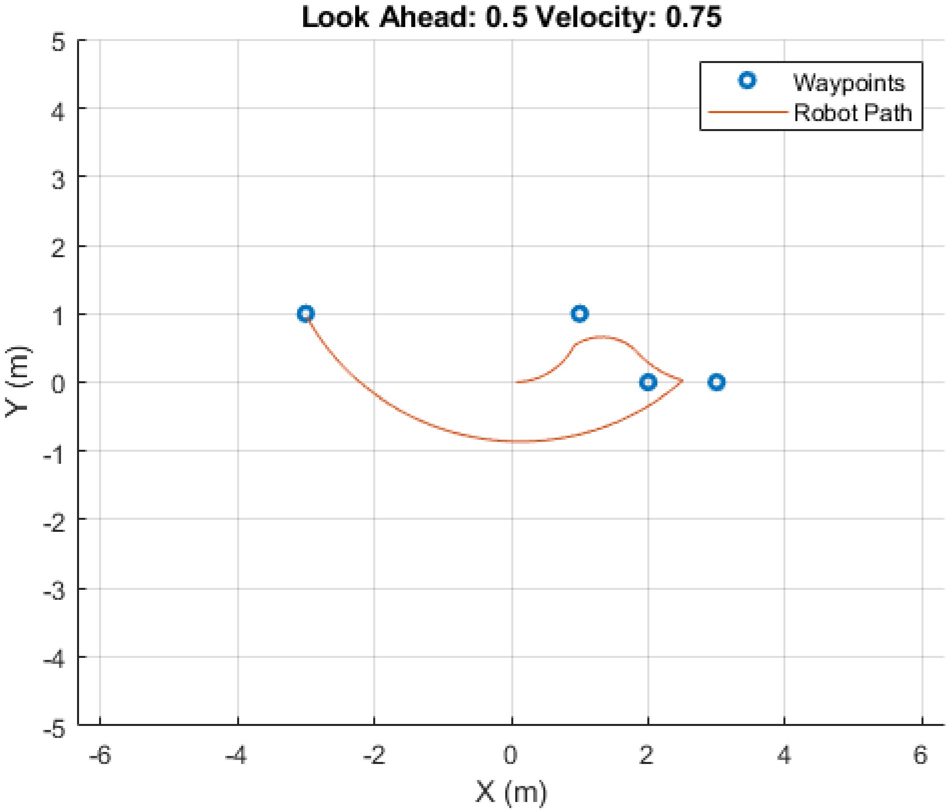
```
% Pure Pursuit Implementation
% Template for Students
clear all
clc
close all

dt = 0.1; % simulation step in sec

waypointlist = [

1 1
2 0
3 0
-3 1

];

velocities = [.5, .5, .75, .75, 1, 1];
velocities = [velocities , 1 3 5]; % auto ones
lookaheadDists =  [.5 1 .5 1 .5 1];

for j = 1:length(velocities)

    clearvars -except robot j dt waypointlist velocities lookaheadDists
    pause(1)


    % robot Global Pose
    robot.X = 0; % X Global robot position in m
    robot.Y = 0; % Y Global robot position in m
    robot.Phi = 0;% Global orientation in radians, measured with respect to
X axis

    robot.angVel = 0;

    % Initialize a pure pursuit structure
    purepursuit.closestWayPointIndex = 1;
    purepursuit.proximityTh = 0.1; % a Threshold to determine if robot is at
the waypoint

    purepursuit.goalPointIndex = 1;

    purepursuit.lookahead = 0.3;
    initialVel = velocities(j);
    robot.vel = initialVel;



    % Draw robot at the initial pose
    robot.wheelR = (13.5/2)/100; % wheel radius in m
    robot.halfWidth = 0.15; % half width in m
    robot.length = 30/100; % length in m

    % Robot Geometry for Display
    robot.bodyWidth = 18.5/100; % in m
    robot.wheelWidth = 6.0/100; % in m; this includes the shaft
```

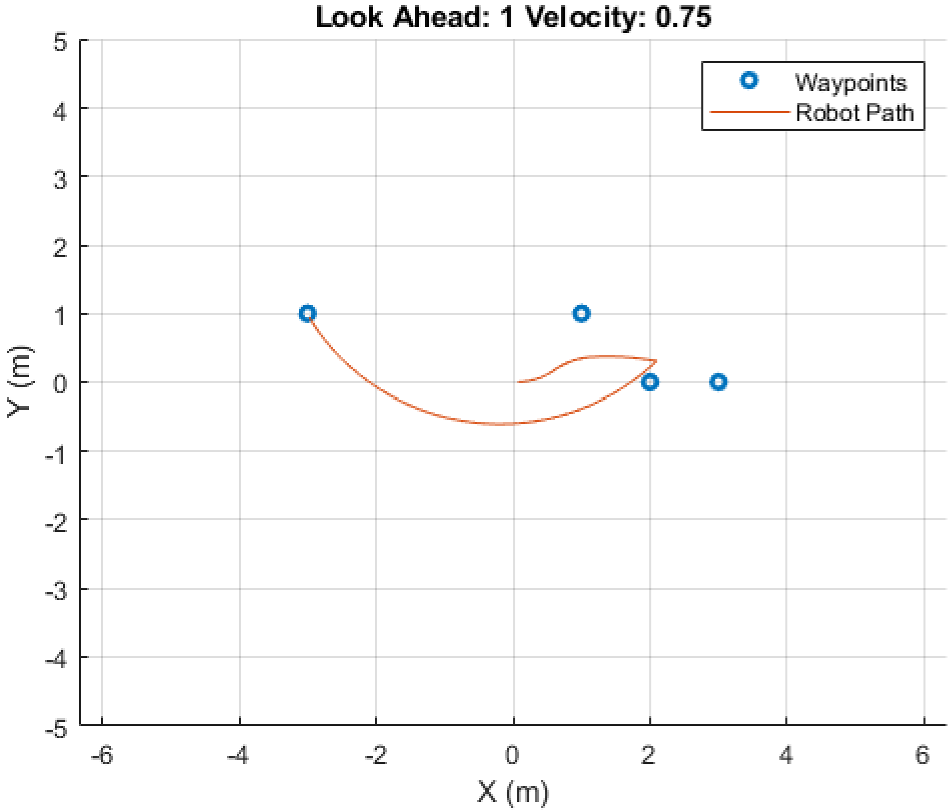
Draw Robot at initial pose

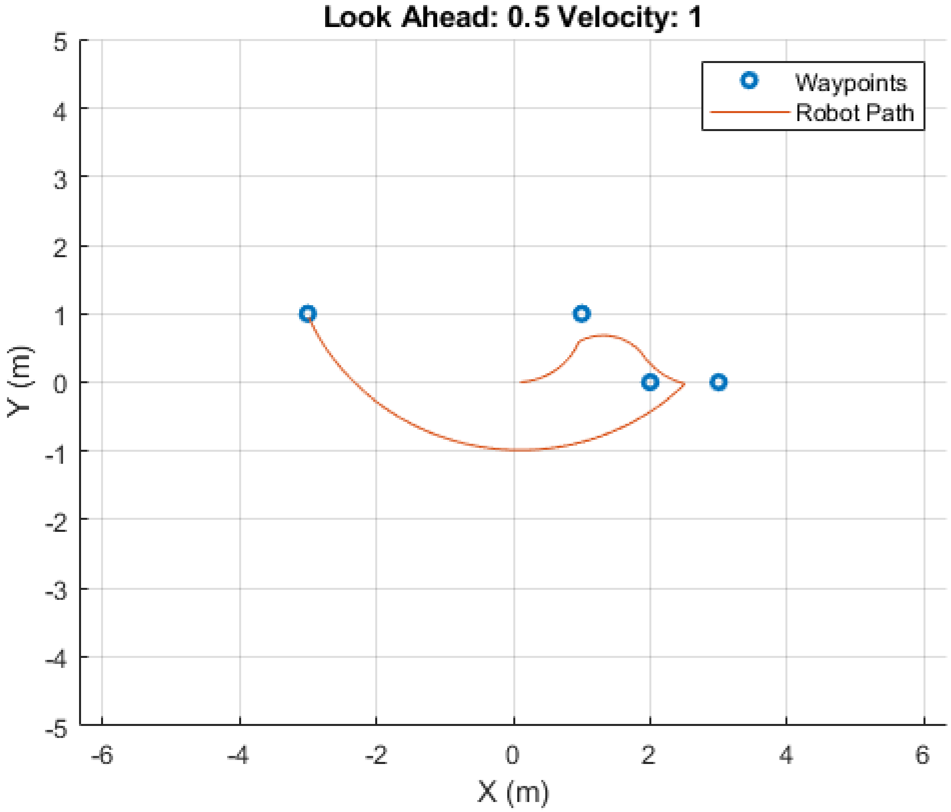
```
    leftCornerBodyX = -(robot.length/2);
    leftCornerBodyY = -(robot.bodyWidth/2);

    % Parent a rectangle to an hgtransform
    hgTX = hgtransform;
    robotBody = rectangle('Parent',hgTX,'Position',[leftCornerBodyX
leftCornerBodyY robot.length robot.bodyWidth],'FaceColor',[1 0 0]);

    shaftLength = 3.5/100;
    leftCornerRightWheelX = leftCornerBodyX - (3/100);
    leftCornerRightWheelY = leftCornerBodyY - shaftLength - robot.wheelWidth;
    leftCornerLeftWheelX = leftCornerBodyX - (3/100);
    leftCornerLeftWheelY = leftCornerBodyY + robot.bodyWidth + shaftLength;
    wheelRight = rectangle('Parent',hgTX,'Position',[leftCornerRightWheelX
leftCornerRightWheelY 2*robot.wheelR robot.wheelWidth],'FaceColor',[0 0 0]);

    wheelLeft = rectangle('Parent',hgTX,'Position',[leftCornerLeftWheelX
leftCornerLeftWheelY 2*robot.wheelR robot.wheelWidth],'FaceColor',[0 0 0]);

    figure(j)
    hold on
    axis([-5 5 -5 5])
    axis('equal')
    drawRobot(hgTX,robot)


    %waypoints = waypointlist(:, (2*j-1):(2*j));
    waypoints = waypointlist(:, 1:2);

    % Compute distance to last waypoint.
    [NwayPoints, dim] = size(waypoints);
    distanceRobotFinalPoint =
getEuclideanDistance(robot.X,robot.Y,waypoints(NwayPoints,1),waypoints(NwayPo
ints,2));

    %Main Loop
    i = 0;

    while((purepursuit.closestWayPointIndex <= NwayPoints) &&
(distanceRobotFinalPoint >= purepursuit.proximityTh))
        i = i + 1;

        %Find the waypoint to head towards.
        purepursuit = findGoalWayPoint(purepursuit,robot,waypoints);

        %Make an update to the closestWaypoint so that in the next iteration
the search is done starting from the latest goalPointIndex.
        purepursuit.closestWayPointIndex = purepursuit.goalPointIndex;

        % Compute the required turn radius
        [flag, turnRadius] = findTurnRadius(robot,purepursuit,waypoints);

        % Compute robot commands. The commands depend on the flag value
        % **** ADD your code here ***

        if (flag == -1)
            robot.angVel = 2;
            robot.vel = 0;
        elseif (flag == 1)
            robot.angVel = -2;
            robot.vel = 0;
        elseif(flag == 0)
            robot.vel = initialVel;
            robot.angVel = robot.vel/turnRadius;
        end


        % Simulate Forward.
        robot = fwdSim(robot,dt);

        % % Draw your robot at the new pose
        % drawRobot(hgTX,robot);
        pause(0.01);

        % Compute distance to Final point to determine if robot should stop
        distanceRobotFinalPoint =
getEuclideanDistance(robot.X,robot.Y,waypoints(NwayPoints,1),waypoints(NwayPo
ints,2));


        % Caclulate new lookahead for last three trials
        if j >6
            distanceWaypoint =
getEuclideanDistance(robot.X,robot.Y,waypoints(purepursuit.goalPointIndex,1),
waypoints(purepursuit.goalPointIndex,2));
            purepursuit.lookahead = .05 * robot.vel / distanceWaypoint;
        else
            purepursuit.lookahead = lookaheadDists(j);
        end
        y(i) = robot.Y;
        x(i) = robot.X;

    end
    % Draw your robot at the new pose
    plot(waypoints(:,1),waypoints(:,2),'o','MarkerSize',5,'Linewidth',2);
    plot(x, y);
    xlabel("X (m)")
    ylabel("Y (m)")
    legend('Waypoints' , 'Robot Path');
    grid on
    if j > 6
        title(strcat("Automatic Lookahead For: ", num2str(round(robot.vel,
3))));
    else
        title(strcat("Look Ahead: ", num2str(purepursuit.lookahead),
strcat(" Velocity: ", num2str(round((robot.vel), 3)))));
    end
    hold off
```

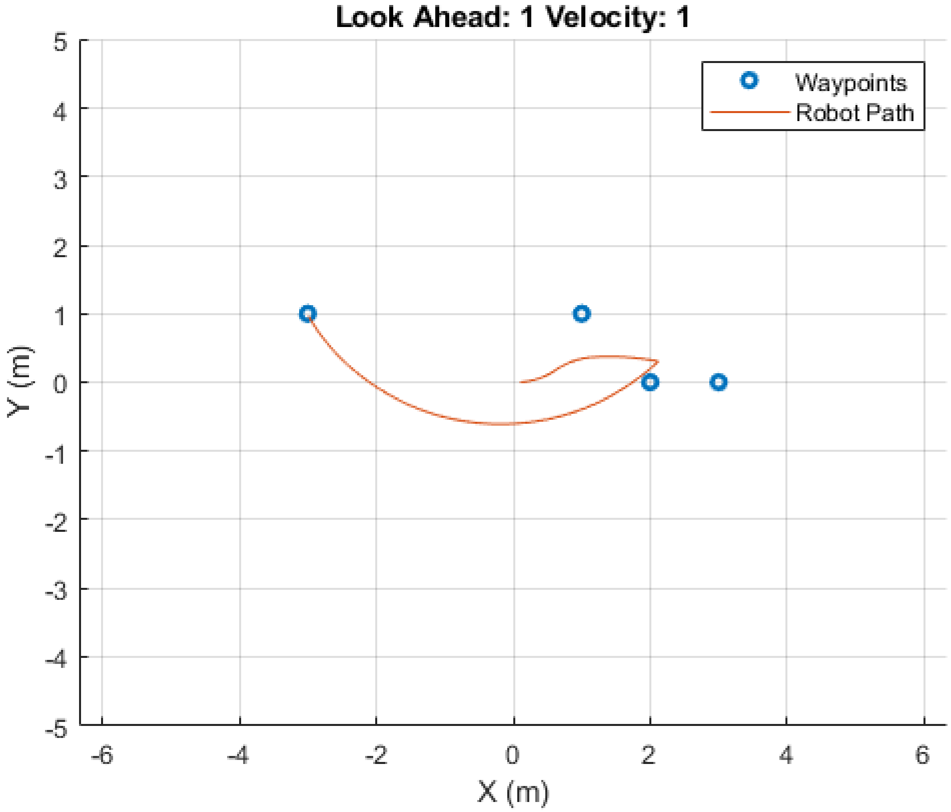


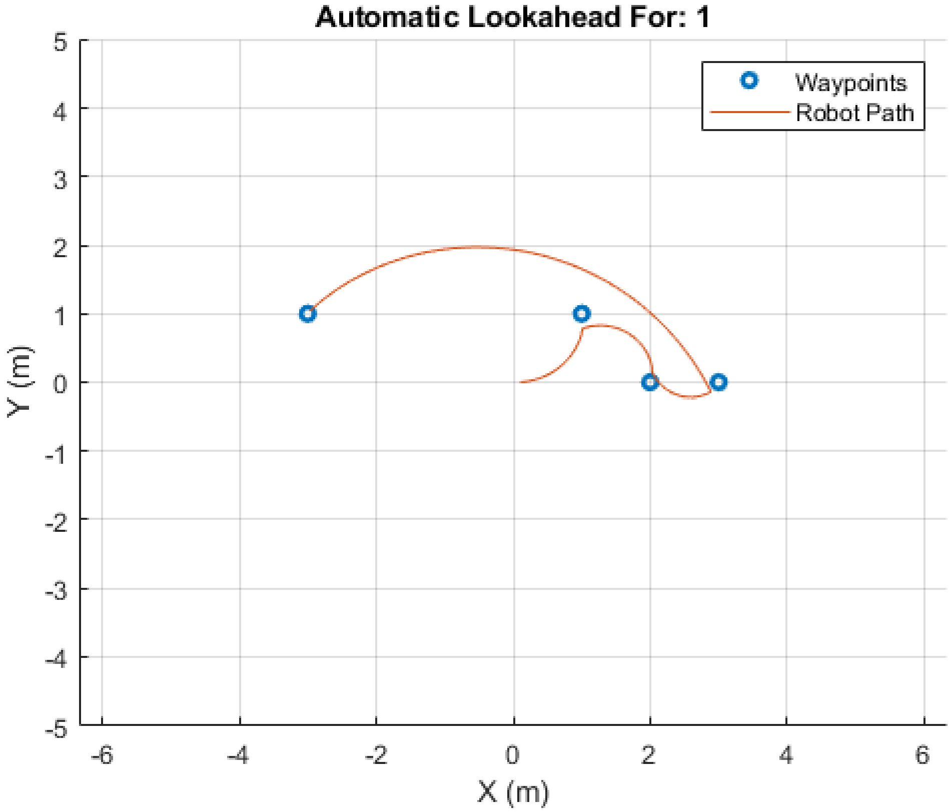

Look Ahead: 1 Velocity: 0.5
Waypoints
Robot Path
Y (m)
X (m)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6

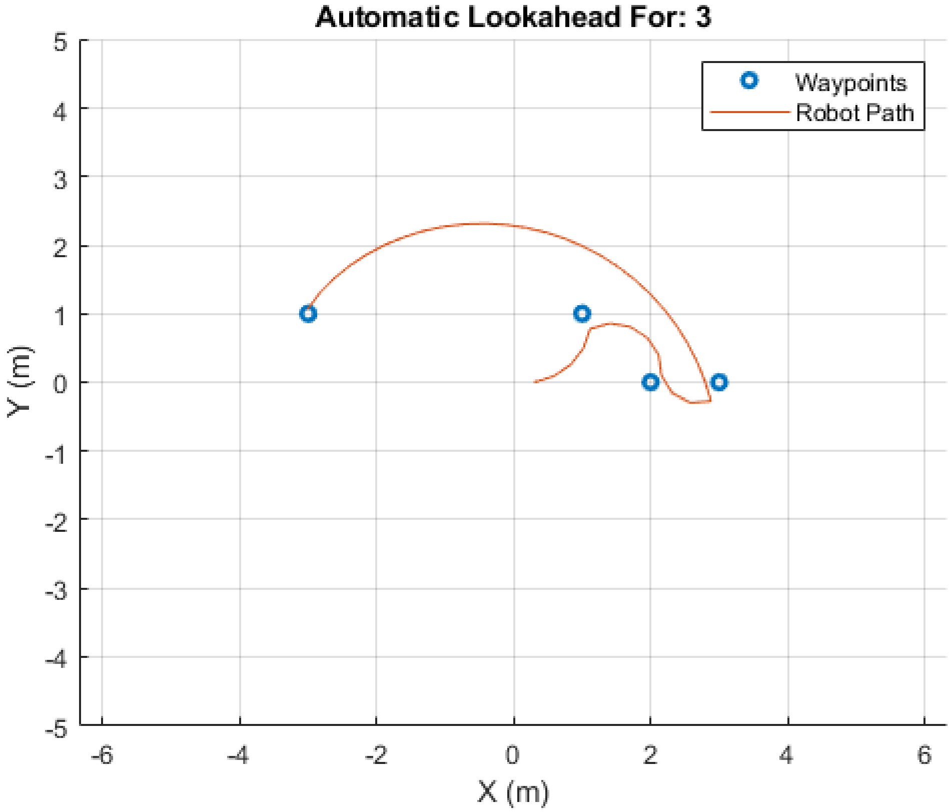

Look Ahead: 0.5 Velocity: 0.75
Waypoints
Robot Path
Y (m)
X (m)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6

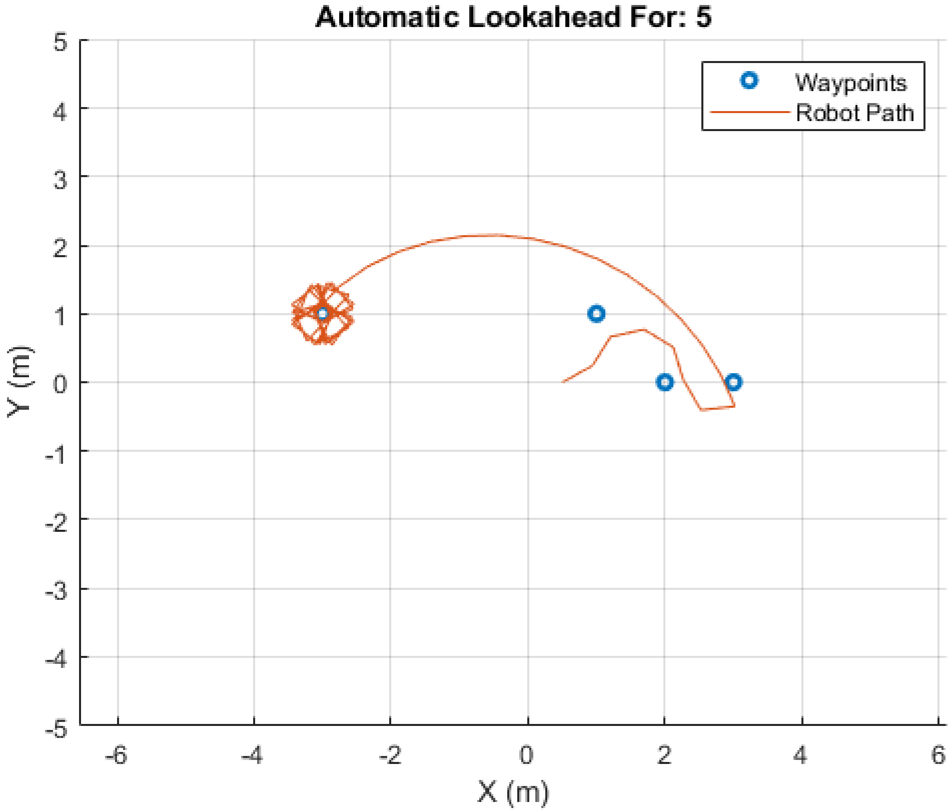

Look Ahead: 1 Velocity: 0.75
Waypoints
Robot Path
Y (m)
X (m)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6


Look Ahead: 0.5 Velocity: 1
Waypoints
Robot Path
Y (m)
X (m)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6

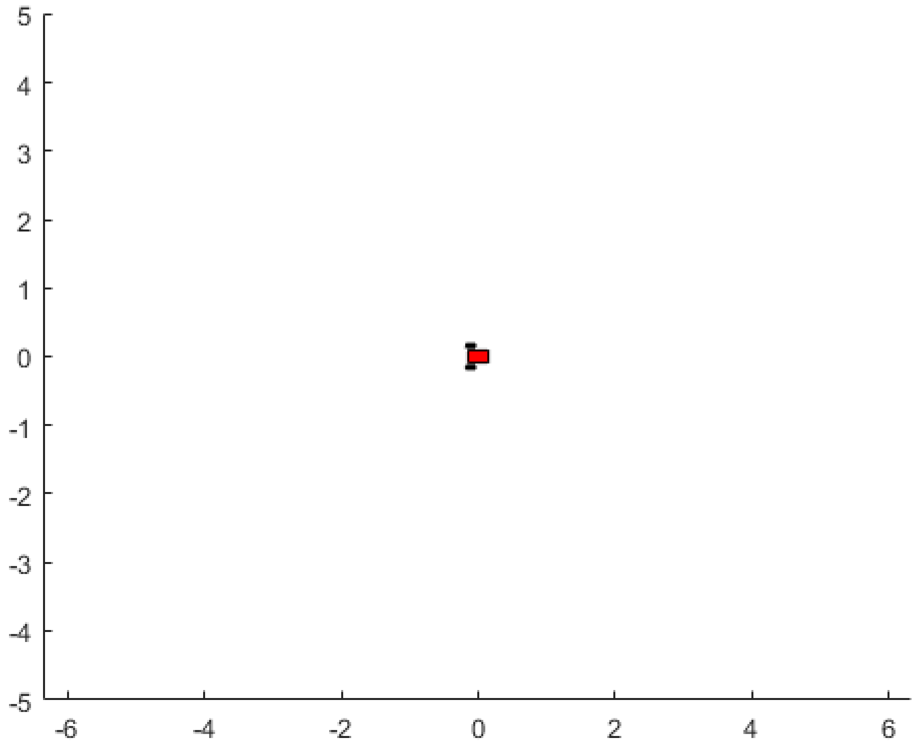

Look Ahead: 1 Velocity: 1
Waypoints
Robot Path
Y (m)
X (m)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6


Automatic Lookahead For: 1
Waypoints
Robot Path
Y (m)
X (m)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6


Automatic Lookahead For: 3
Waypoints
Robot Path
Y (m)
X (m)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6


Automatic Lookahead For: 5
Waypoints
Robot Path
Y (m)
X (m)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6

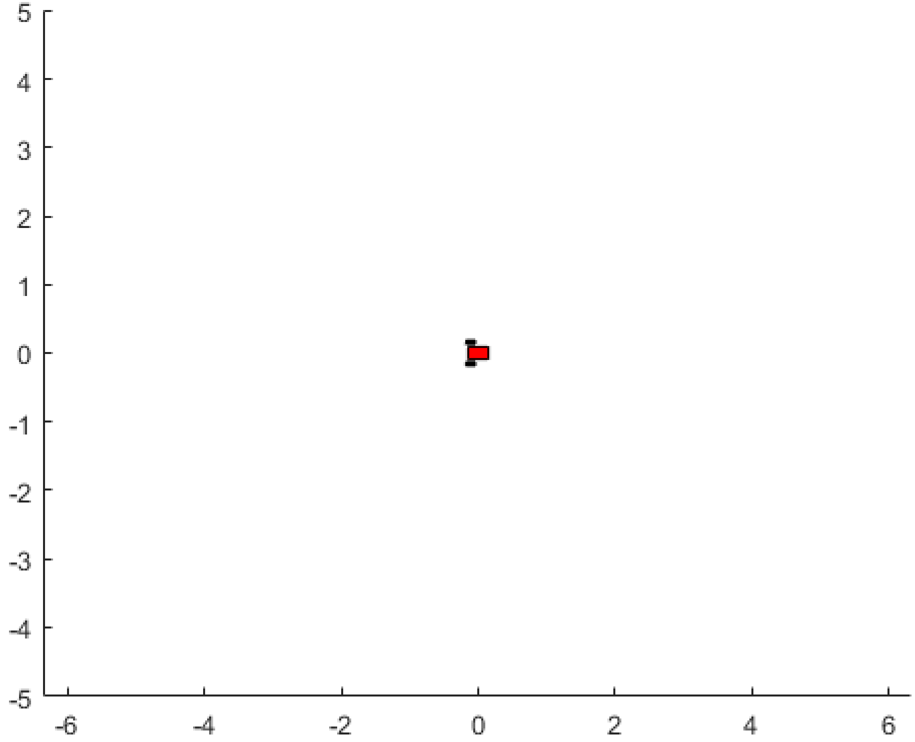
```
end
```

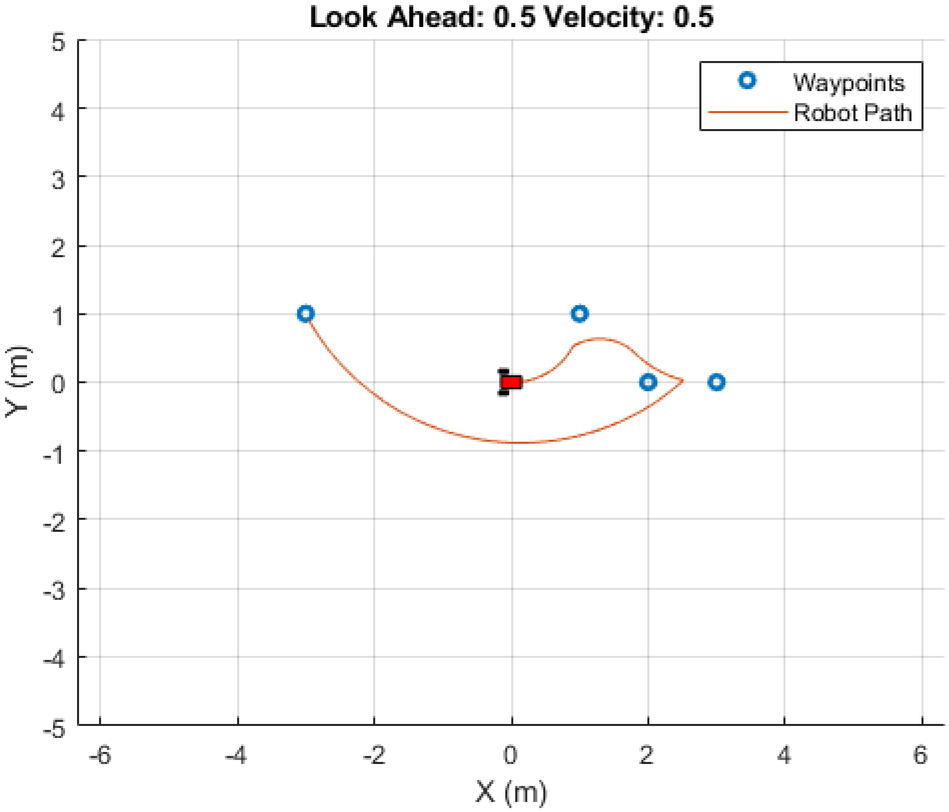


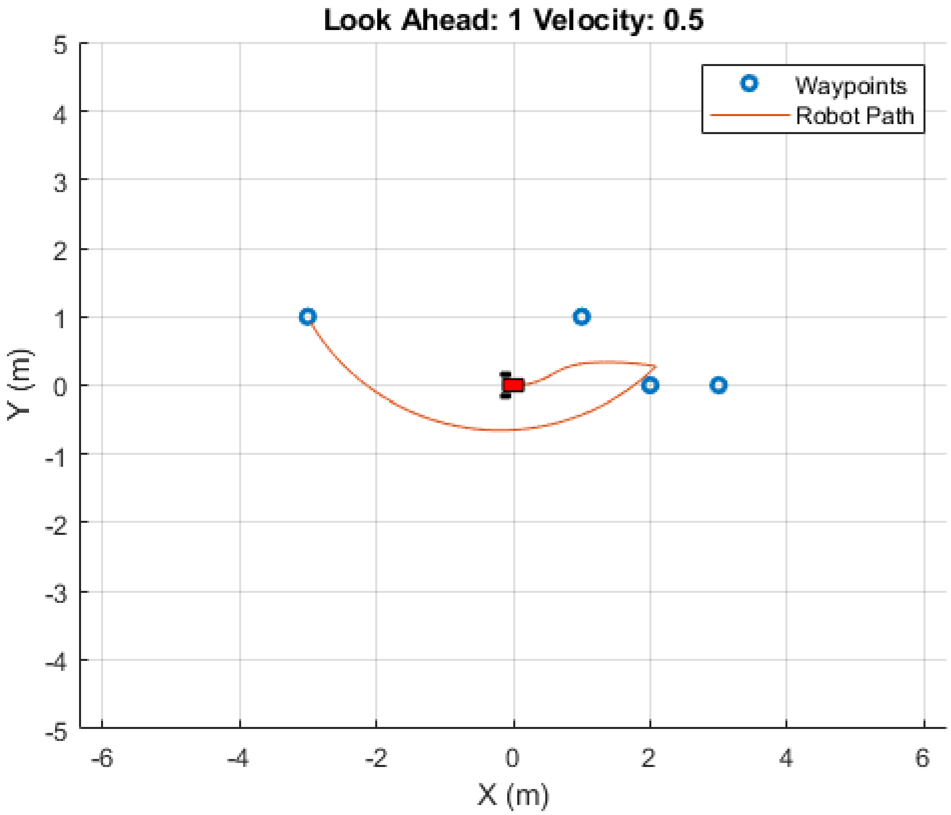
Functions

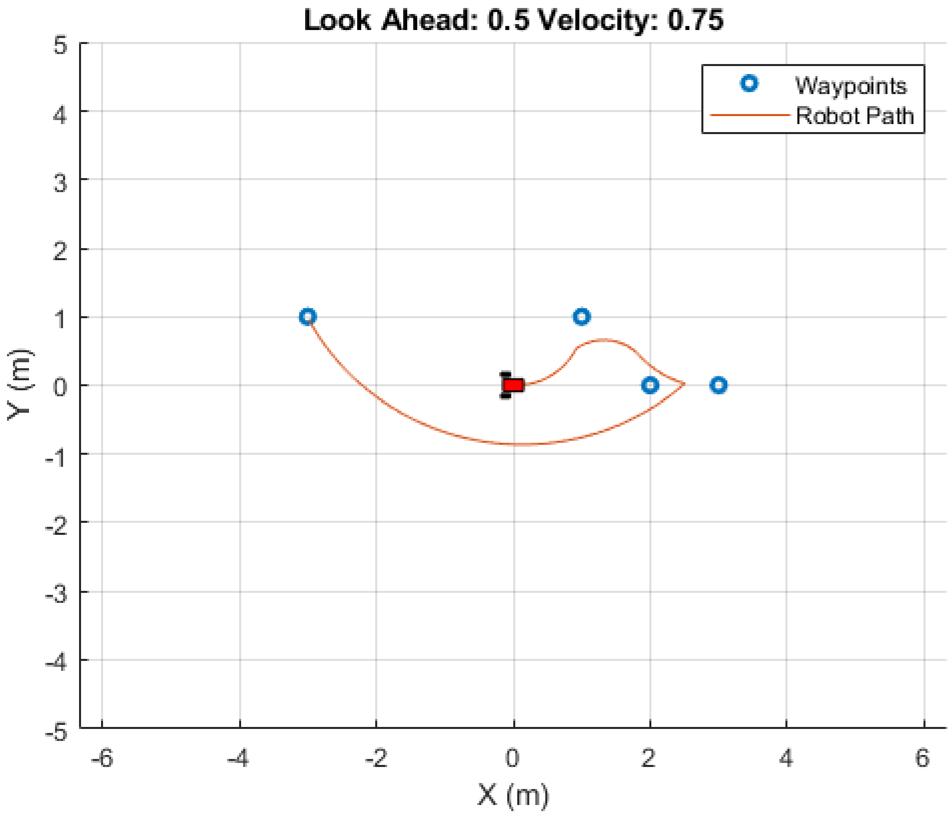
robot animation function

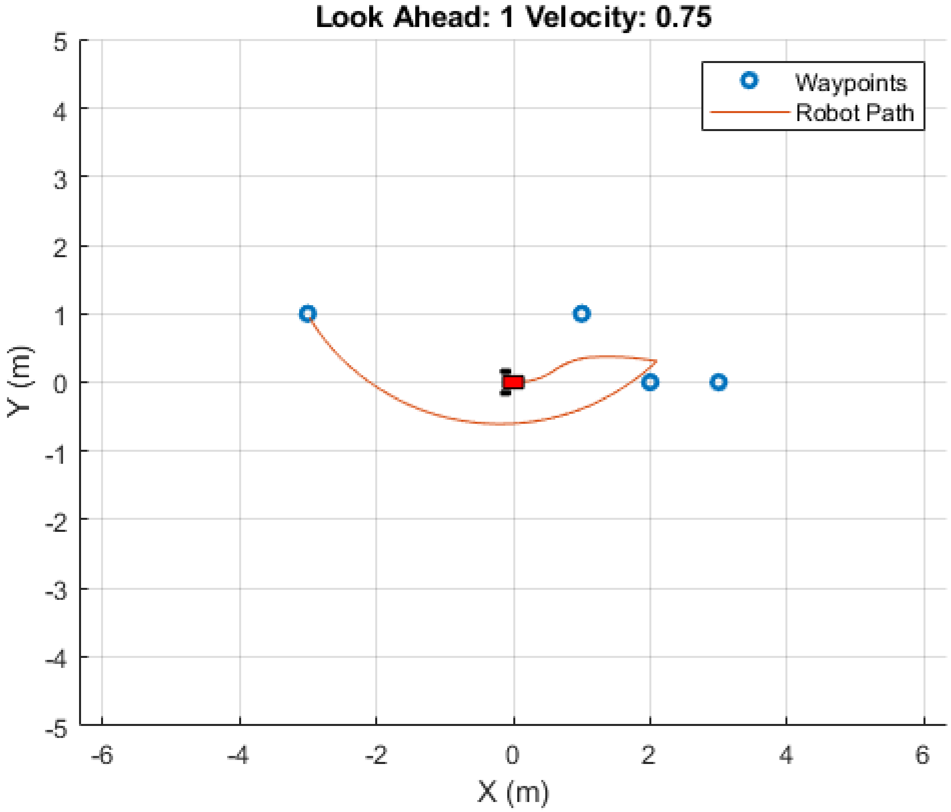
```
function drawRobot(hgTX,robot)

hgTX.Matrix = makehgtform('translate',[robot.X robot.Y
0],'zrotate',robot.Phi);

end

function robot = fwdSim(robot,dt)
robot.X = robot.X + robot.vel*cos(robot.Phi)*dt;
robot.Y = robot.Y + robot.vel*sin(robot.Phi)*dt;
robot.Phi = robot.Phi + robot.angVel*dt;
end
```

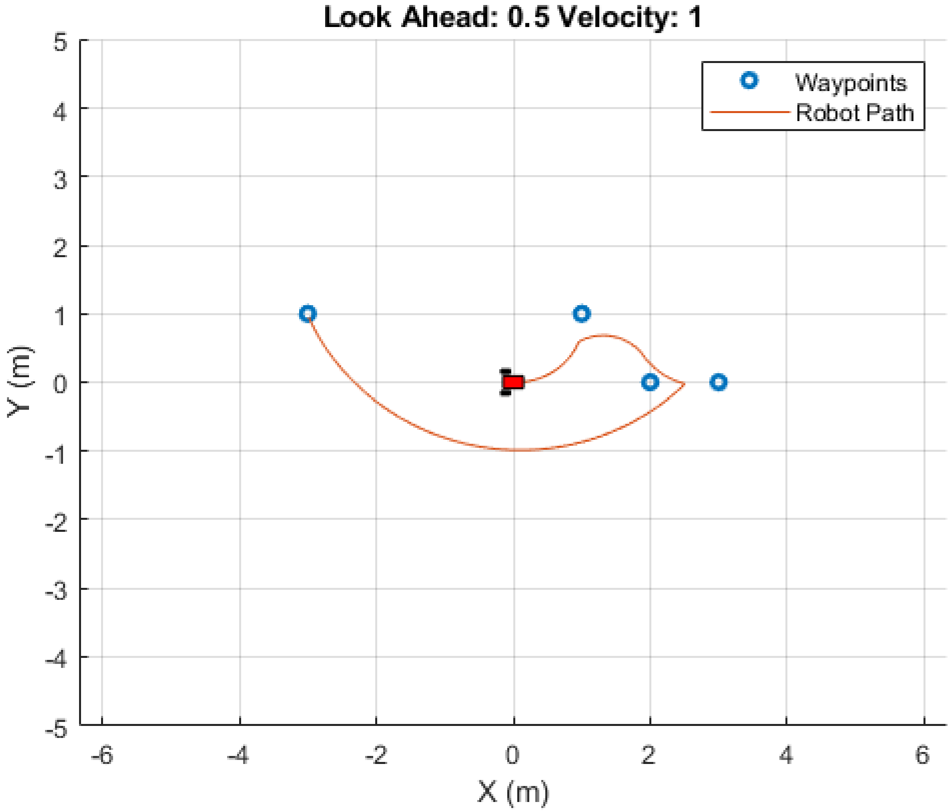

5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6

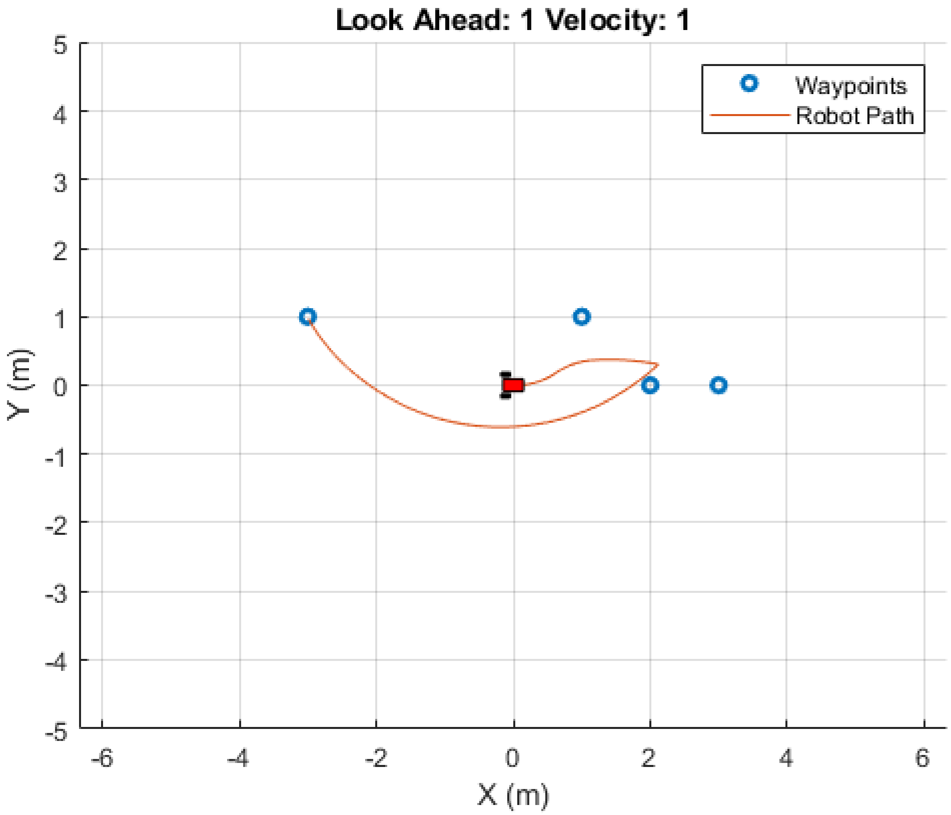

Look Ahead: 0.5 Velocity: 0.5
Waypoints
Robot Path
Y (m)
X (m)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6

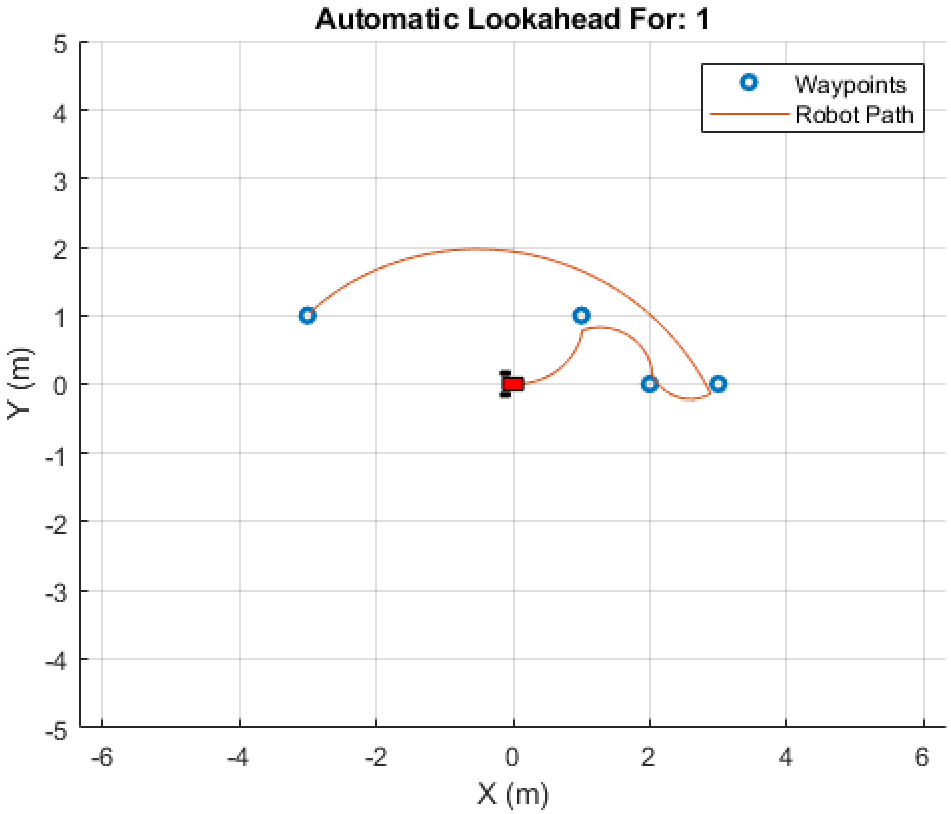

Look Ahead: 1 Velocity: 0.5
Waypoints
Robot Path
Y (m)
X (m)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6

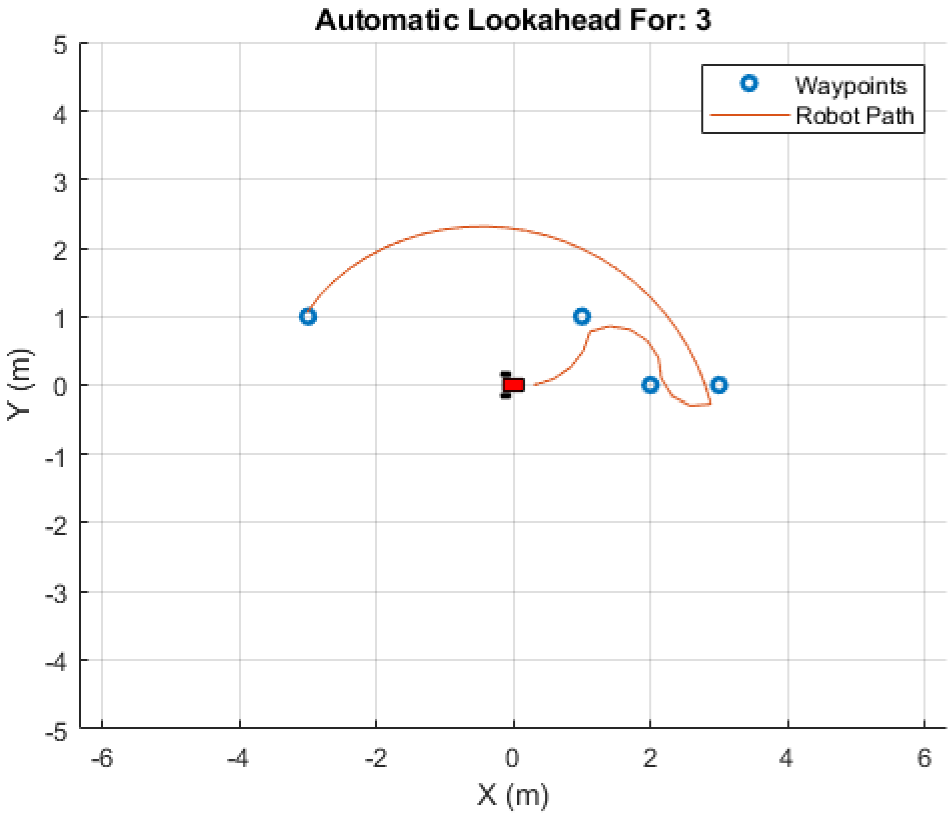

Look Ahead: 0.5 Velocity: 0.75
Waypoints
Robot Path
Y (m)
X (m)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6


Look Ahead: 1 Velocity: 0.75
Waypoints
Robot Path
Y (m)
X (m)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6


Look Ahead: 0.5 Velocity: 1
Waypoints
Robot Path
Y (m)
X (m)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6


Look Ahead: 1 Velocity: 1
Waypoints
Robot Path
Y (m)
X (m)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6


Automatic Lookahead For: 1
Waypoints
Robot Path
Y (m)
X (m)
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6



Functions that you need to implement

```
function distance = getEuclideanDistance(x1,y1,x2,y2)

distance = sqrt((x2-x1)^2 + (y2-y1)^2);

end

function purepursuit = findGoalWayPoint(purepursuit,robot,waypoints)
% Find the goal waypoint towards which the robot should drive. You
% must find the index of the goal waypoint that is at least one lookahead
% distance from the robot position. Note that purepursuit has a
% closestWayPointIndex that we use to always search forward.
% this function will update the field purepursuit.goalPointIndex

distance = getEuclideanDistance(robot.X,
robot.Y, waypoints(purepursuit.closestWayPointIndex, 1),
waypoints(purepursuit.closestWayPointIndex, 2));

if ((distance <= purepursuit.lookahead) && (purepursuit.closestWayPointIndex
< length(waypoints)))
    purepursuit.goalPointIndex = purepursuit.goalPointIndex + 1;

elseif (purepursuit.goalPointIndex == length(waypoints))
```

```
    robot.vel = 0;
    robot.angVel = 0;
end


end

function [flag, turnRadius] = findTurnRadius(robot,purepursuit,waypoints)
% This function computes the turn radius of the circle which
% will take the robot to the goal waypoint.
% The flag output is zero if the goal is in front of the robot.
% -1 : if a left point turn should be performed
%     1: if a right hand turn should be performed

currentWPIndex = purepursuit.goalPointIndex;

local_x = cos(robot.Phi) * (waypoints(currentWPIndex, 1) - robot.X) +
sin(robot.Phi) * (waypoints(currentWPIndex, 2) - robot.Y);
local_y = -sin(robot.Phi) * (waypoints(currentWPIndex, 1) - robot.X) +
cos(robot.Phi) * (waypoints(currentWPIndex, 2) - robot.Y);

local_phi = atan2(local_y, local_x);

if (local_phi > (pi/2 - .5))
    flag = -1;
elseif(local_phi < (-pi/2 + .5))
    flag = 1;
else
    flag = 0;
end

turnRadius = (local_x^2 + local_y^2)/(2*local_y);

end
```

Published with MATLAB® R2024a