COT 4930 COT 5930 EEL 4930 EEL 5661 Robotic Applications (Fall 2022)

Homework 2 (Version 3.0)

Problem 1: Trajectory Planning

1.1 A flying saucer has a homogeneous transformation A (shown below) with respect to its space base. A translation unit of 1 means "1 billion miles". This spaceship can move along any straight line, emanating from its body center, at a top speed of 10 million miles per hour and acceleration time of 10 hours. With the same acceleration time, it can also perform ZYZ Euler angles motion at a top angular velocity of 0.1 rad/hour. The spaceship needs to reach a transformation B (shown below). Estimate how many hours are needed for the trip. Then design (using the "mtraj" command and "lspb" one-dimensional trajectories) and plot the needed motions.

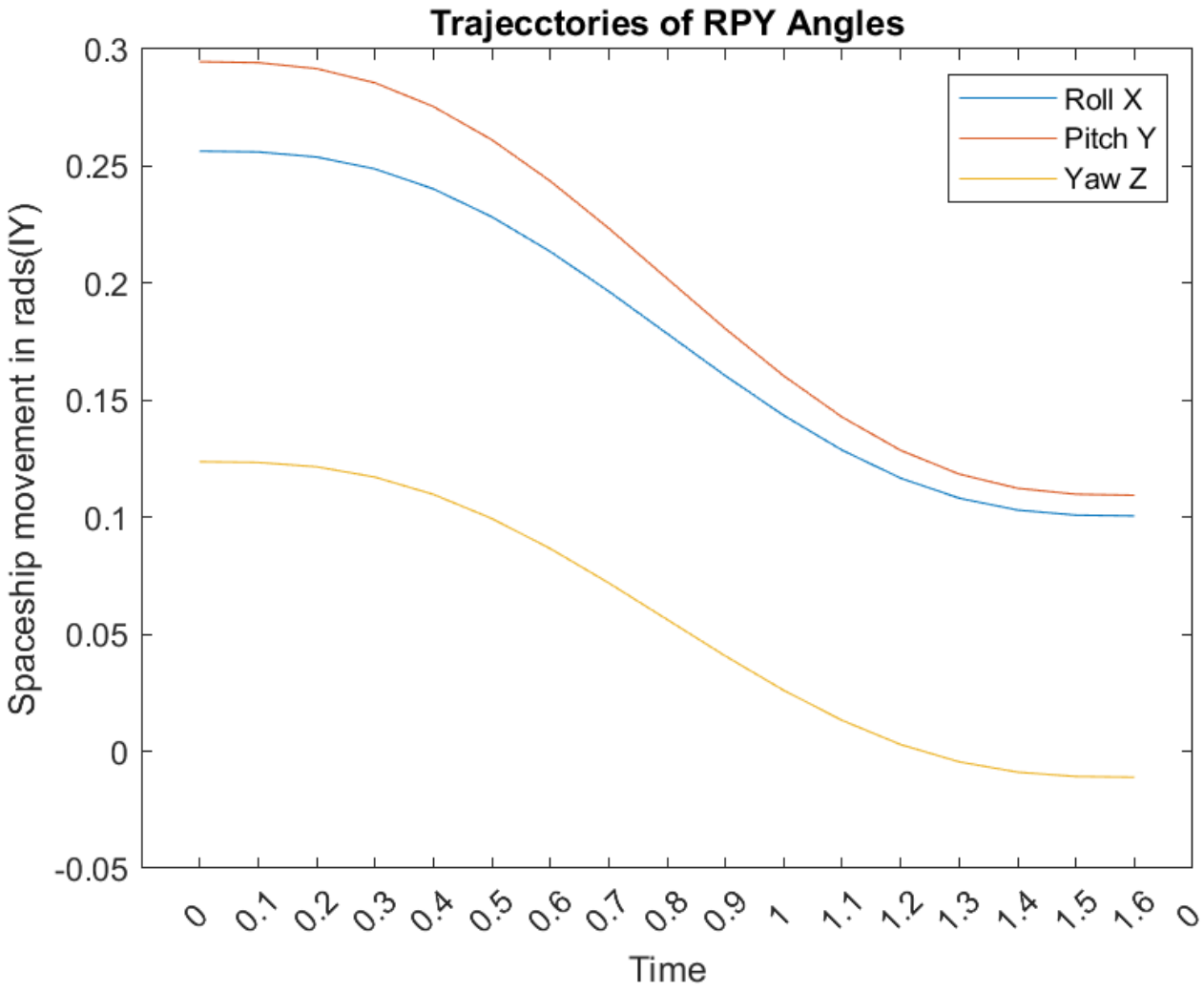
$$A = \begin{pmatrix} 0.9256 & -0.2427 & 0.2904 & 0.9797 \\ 0.2863 & 0.9508 & -0.1181 & -0.7888 \\ -0.2474 & 0.1925 & 0.9496 & 1.3115 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 0.9890 & -0.0998 & 0.1092 & 0.4004 \\ 0.0992 & 0.9950 & 0.0110 & 0.1555 \\ -0.1098 & 0 & 0.9940 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

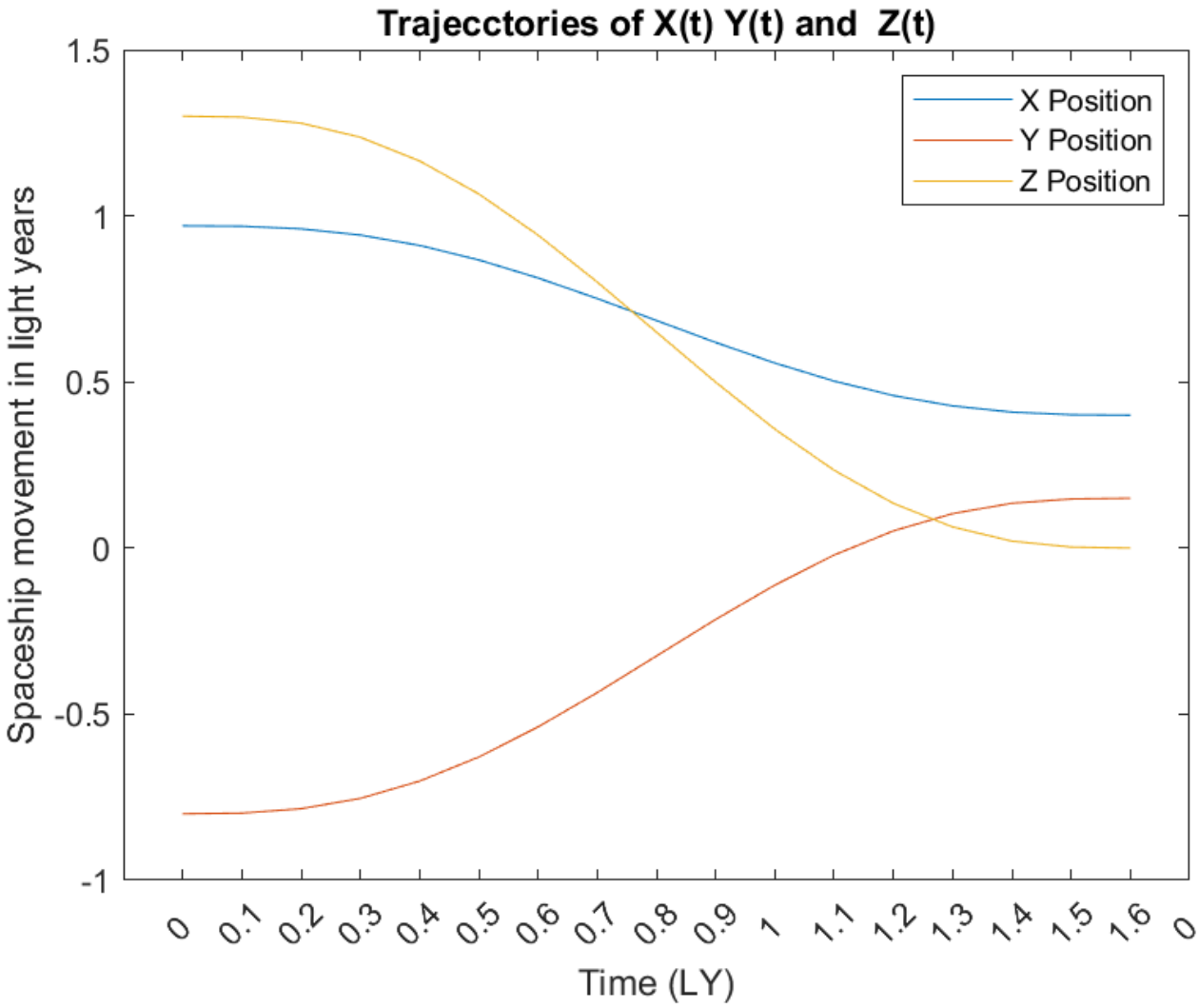
Part 1:

```
%1000 million = 1 billon
% Top speed of 10 million miles per hour and acceleration time of 10 hours

% (Rounding) Units of initial position
xA = 0.9
yA = -8
zA = 1.3

% (Rounding) Units to travel to from initial position
xB = 0.4
yB = 0.2
zB = 0

% Difference traveled per position in 10 hours
Xab = 0.5
Yab = -8.2
Zab = 1.3


Xab = 0.5 % 500 million miles  /  10 million mph = 0.05 hours in X
Yab = -8.2 %  8 billion miles / 10 million mph = -82 hours in Y
Zab = 1.3  % 1 billion and 300 million miles /10 million mph =  13 hours in Z
```

Part 2

```
close all;
% Shaun Pritchard
% Ell 5661
% Problem 1.1 HW 2

tt = 0:0.1:1.6; % Time as distance

A = [0.9256 -0.2427 0.2904 0.9797; 0.2863 0.9508 -0.1181 -0.7888; -0.2474 0.1925
0.9496 1.3115; 0 0 0 1];
B = [0.9890 -0.0998 0.1092 0.4004; 0.0992 0.9950 0.0110 0.1555; -0.1098 0 0.9940 0; 0
0 0 1];

PosA =  [0.97 -0.8 1.3 ];
PosB = [0.4 0.15 0 ];

qpos = mtraj(@tpoly, PosA, PosB, tt);
plot(qpos)
title("Trajecctories of X(t) Y(t) and  Z(t)")
xlabel('Time (LY)')
ylabel('Spaceship movement in light years')
legend( {'X Position', 'Y Position', 'Z Position' }, 'Location', ' northeast')
set(gca, 'XTick', 1:21)
set(gca, 'XTickLabel', tt)

figure
RPYA = tr2rpy(A, 'xyz'); %Get roll, pitch yaw with angles of transformation for A
RPYB = tr2rpy(B, 'xyz'); %Get roll, pitch yaw with angles of transformation for B

qrpy = mtraj(@tpoly, RPYA,RPYB, tt);
plot(qrpy)
title("Trajecctories of RPY Angles")
xlabel('Time')
ylabel('Spaceship movement in rads(lY)')
legend({'Roll X', 'Pitch Y', 'Yaw Z' }, 'Location', ' northeast')
set(gca, 'XTick', 1:21)
set(gca, 'XTickLabel', tt)
```


Trajecctories of RPY Angles
Roll X
Pitch Y
Yaw Z
Spaceship movement in rads(IY)
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1.1
1.2
1.3
1.4
1.5
1.6
0
Time



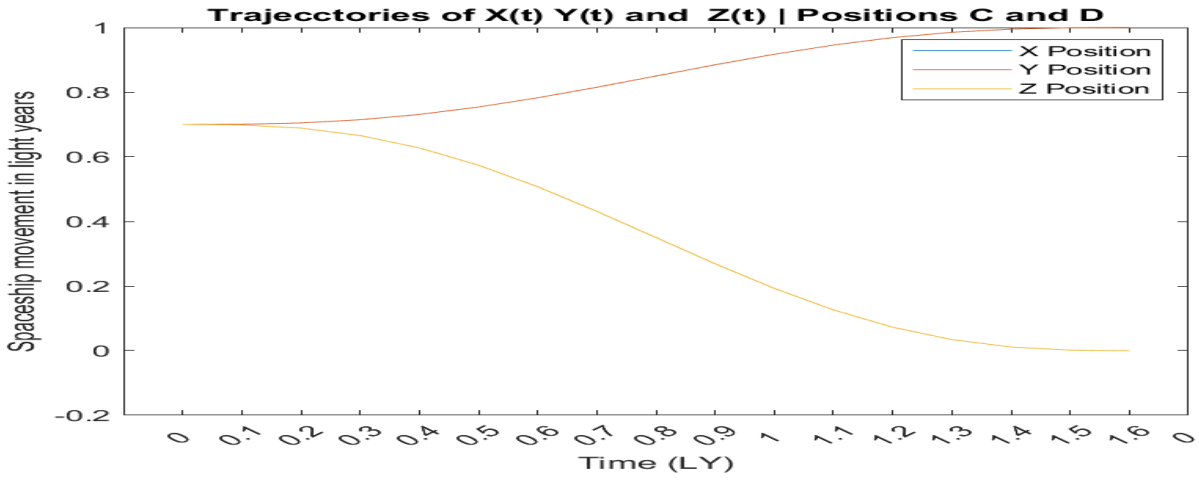
In figures 1 and 2, we can see that each trajectory crosses from Transformation A to Transformation B in 1.6 rad/hours at a total of 1.6 billon miles

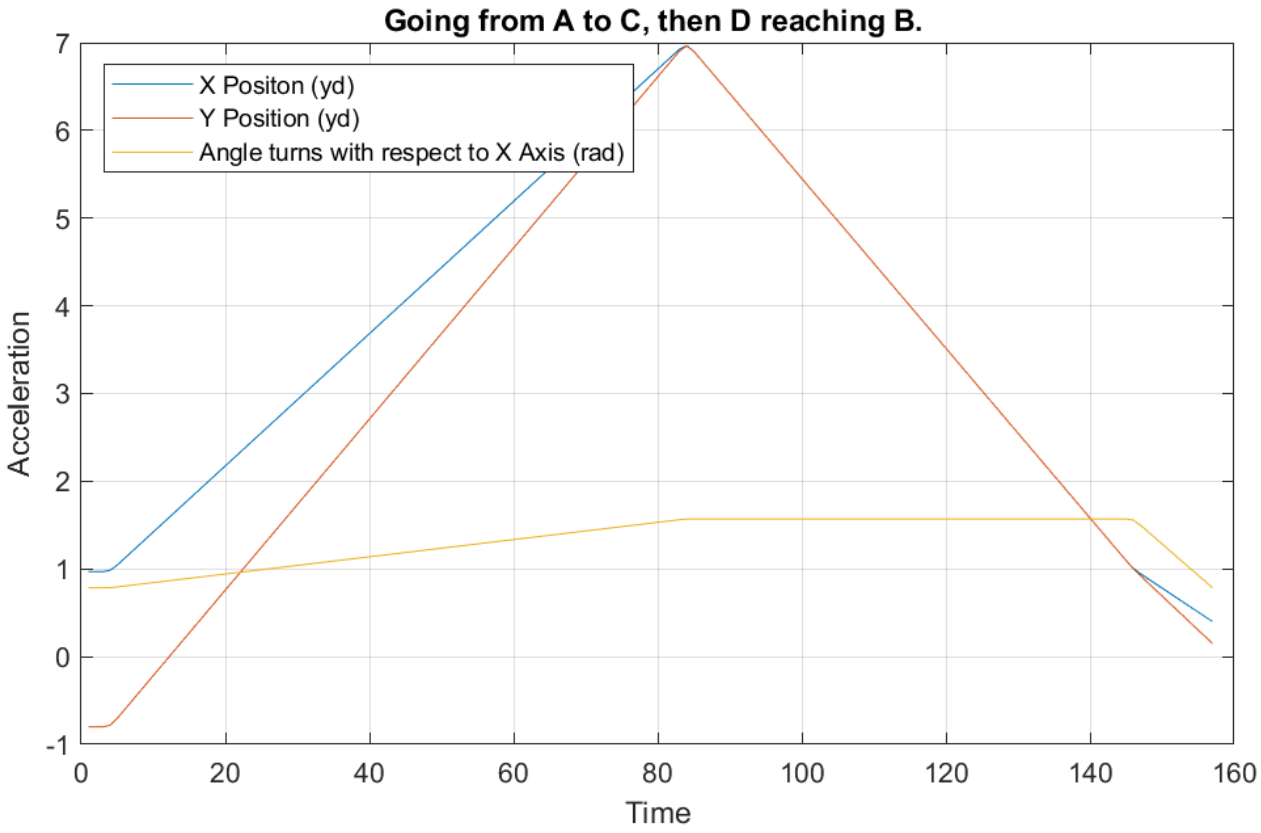
1.2 As in 1.1, except that the spaceship must go near two via points characterized by transformations C and D. It should take more or less the same amount of time for each segment, going from A to C, then D reaching B. Use “mstraj” with “tpoly” to plan and plot the flying saucer’s motion.

$$C = \begin{pmatrix} 1.0000 & 0 & 0 & 0.7000 \\ 0 & 1.0000 & 0 & 0.7000 \\ 0 & 0 & 1.0000 & 0.7000 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad D = \begin{pmatrix} 1.0000 & 0 & 0 & 1.0000 \\ 0 & 1.0000 & 0 & 1.0000 \\ 0 & 0 & 1.0000 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
%PART II via points
% Acceleration time 1.6
via =[0.97 -0.8 pi/4; 7 7 pi/2; 1 1 pi/2;  0.4 0.15 pi/4]';
q1 = mstraj(via(:,[1 2 3 4])',[0.1 0.1 0.1],[],via(:,1)',1,2);
figure
plot(q1)
grid on
title('Going from A to C, then D reaching B. ')
xlabel("Time")
ylabel("Acceleration")
legend({'X Position (yd)', 'Y Position (yd)', 'Angle turns with respect to X Axis
(rad)'}, 'Location', 'NW')
```

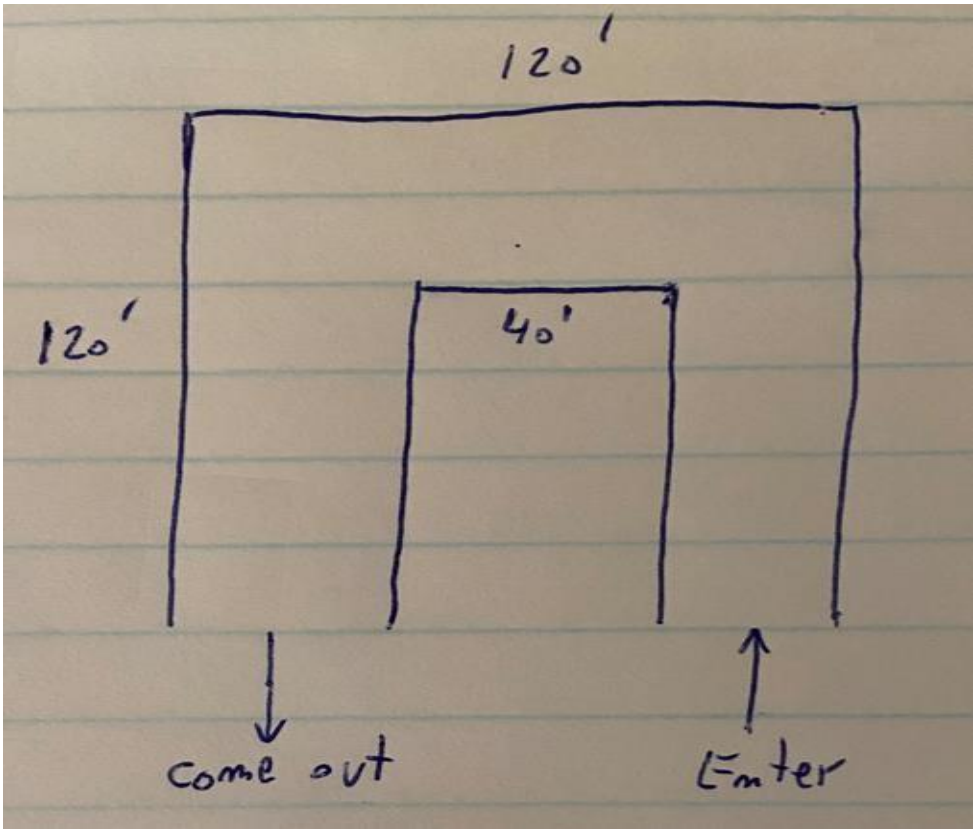
Trajectories of C to D





Problem 2: Wheeled Robotic Vehicles

Consider the following top view of a racetrack for mobile robots. Two robots, a tricycle robot and a differential drive robot compete (one at a time) who can traverse the racetrack faster. Both robots are rectangular shaped having dimensions of 5' by 3' each. So, no problem navigating inside the 40' wide corridors. Both vehicles have a top linear speed of 10' per second. The tricycle has a top steering angular velocity of 0.1 rad/s. The two vehicles have acceleration time of 1s from 0 to any top speed. Both vehicles have the same distance between the center of the front wheel and the center point between the two back wheels. You may make more reasonable assumptions, if needed. There is no feedback control. All motions are pre-planned and are executed in **open loop**. Whoever wins the race depends entirely on the motion that is pre-planned for each robot.



Your job is to plan the best path that you can for each vehicle and execute its motion in Simulink.

Use simple trapezoidal velocity profiles everywhere that velocity profiles need to be defined.

Which robot wins? The tricycle would win because the differential would have different speeds for each when given the 0 angular velocity slowing it down and creates a smother plot overall.

$v = (v,0)$

$q = (x,y,0)$

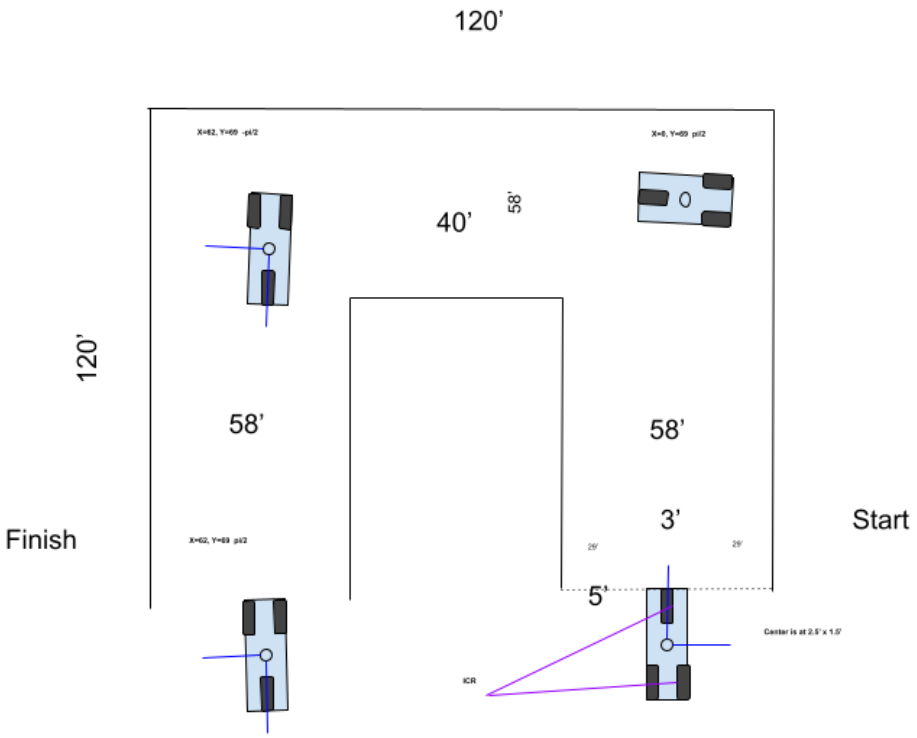
$D0/dt = v/Rb$

$Rb = L / Tan\gamma$

Robot 1: Tricycle robot 5' by 3' size, top linear speed of 10' per second, The tricycle has a top steering angular velocity of 0.1 rad/s per second, and acceleration time of 1s from 0

Robot 2: Differential drive robot 5' by 3' size, top linear speed of 10' per second, no cap on the top steering angular velocity rad/s per second, and acceleration time of 1s from 0.

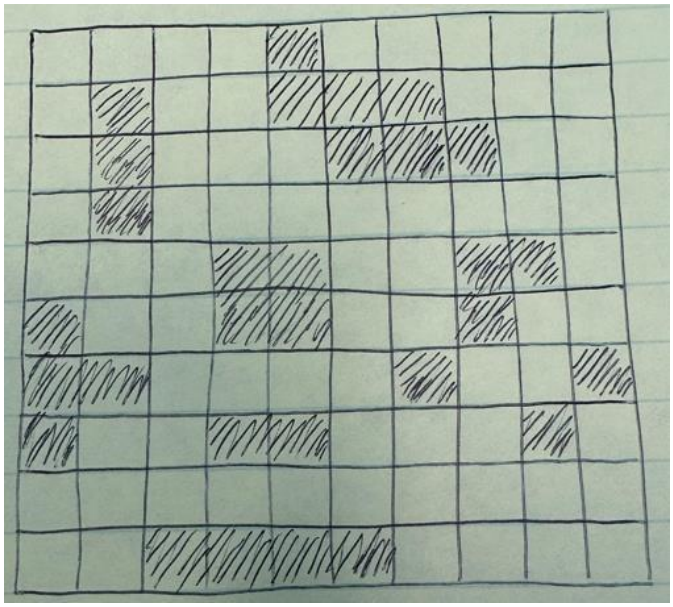
Each vehicle has a pose of q = (x,y,0) The mobile robot “joint variables” are its linear velocity v and its steering angle γ. The vehicles are driving in an open loop were v(t) and γ(t) are user defined signals.



Based on the map

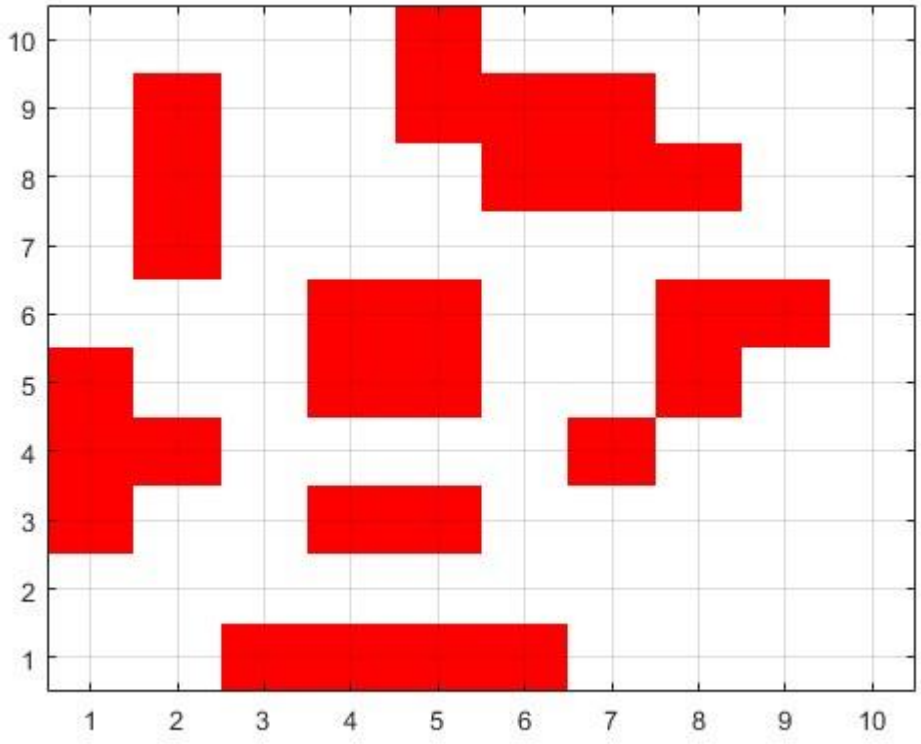
Problem 3: Mobile Robot Navigation

The following 10x10 maze is used for four different mobile robot navigation tasks (3.1-3.4).



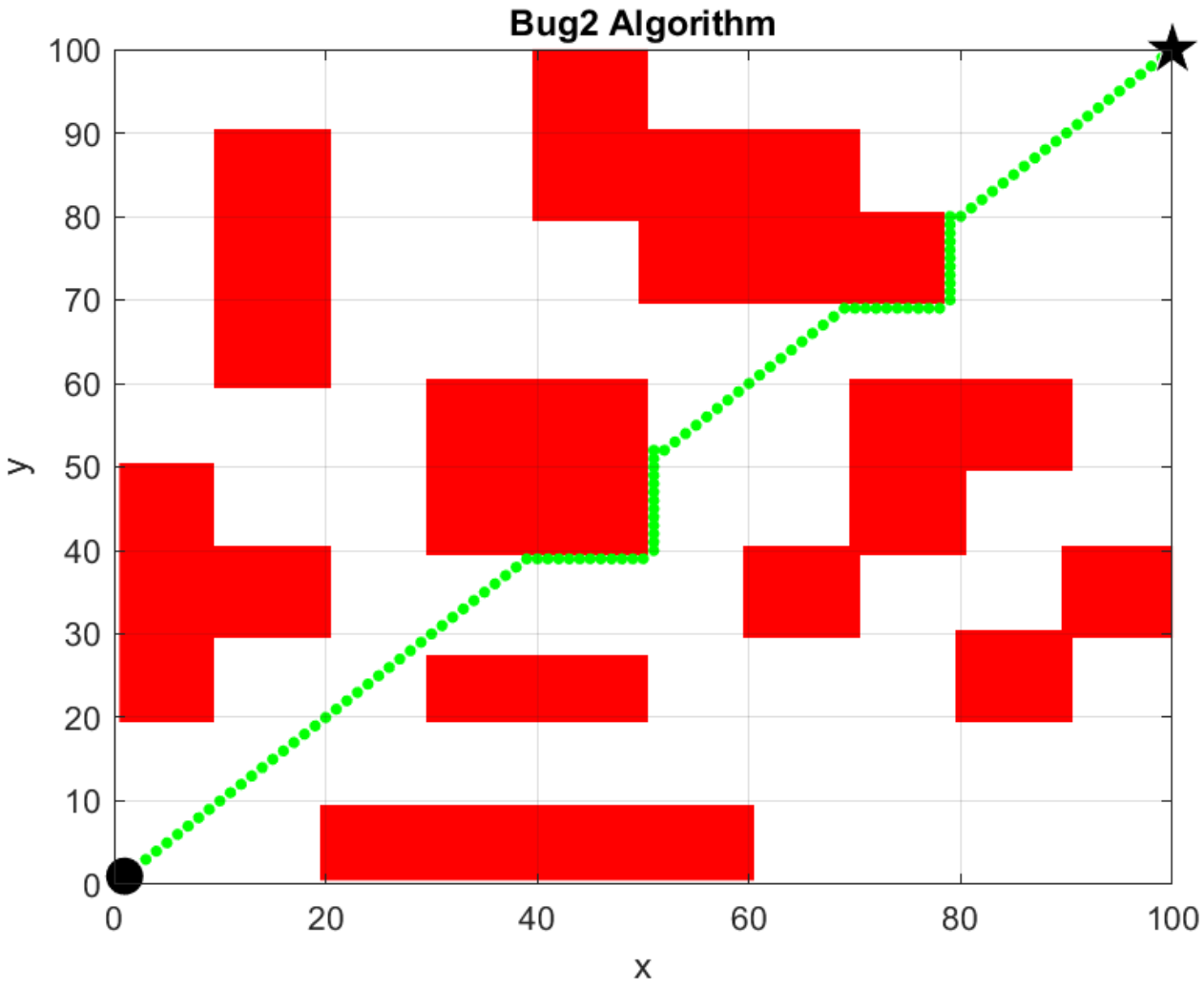
3.1 Choose a starting point somewhere near the SW corner and a target cell near the NE corner. Show manually, and with MATLAB, how to navigate using the Bug2 algorithm.

Manual Implementation



From block 1,1 SW corner to 10,10 in the NW corner

1. NORTH 1,1 TO 1,2
2. EAST FROM 1,2 TO 8,2
3. NORTH FROM 8,2 TO 8,4
4. EAST FROM 8,4 TO 9,4
5. NORTH FROM 9,4 TO 10,5
6. NORTH FROM 10,5 TO 10,10

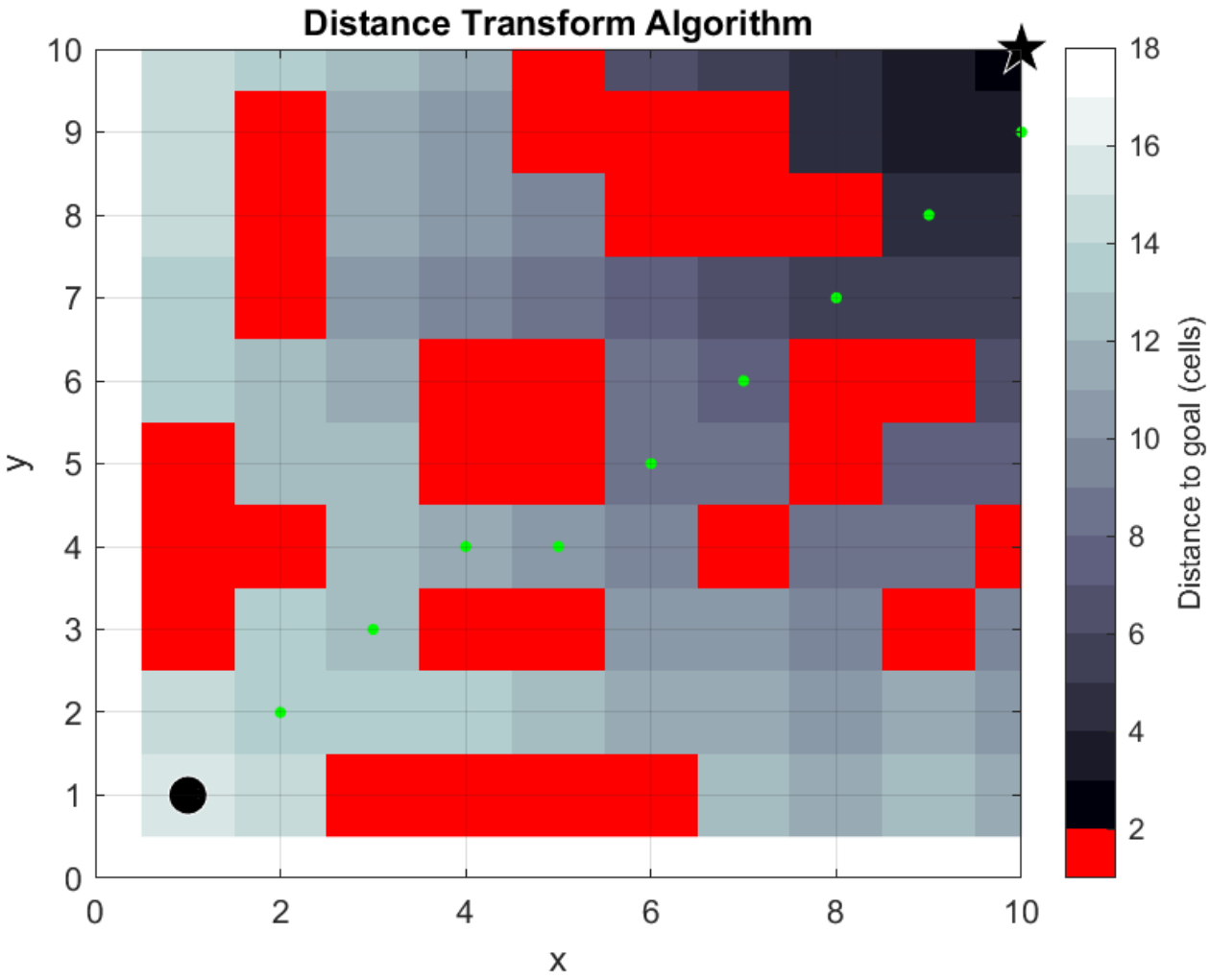


By cutting through center, Bug2 algorithm selected the m path directly to section 10,10, whereas my manual attempt dissipated to the north east direction.

```
map = [
 0,0,1,1,1,1,0,0,0,0;
 0,0,0,0,0,0,0,0,0,0;
 1,0,0,1,1,0,0,0,1,0;
 1,1,0,0,0,0,1,0,0,1;
 1,0,0,1,1,0,0,1,0,0;
 0,0,0,1,1,0,0,1,1,0;
 0,1,0,0,0,0,0,0,0,0;
 0,1,0,0,0,1,1,1,0,0;
 0,1,0,0,1,1,1,0,0,0;
 0,0,0,0,5,0,0,0,0,0;
];
start = [1 1];
goal = [10 10];


goal3_1 = [100 100]; %define the goal
```

```
map3_1 = zeros(100); % create an array with only zeros

map3_1(20:50,1:9) = 1; % create obstacles on map
map3_1(30:40,10:20) = 1; % create obstacles on map
map3_1(60:90,10:20) = 1; % create obstacles on map
map3_1(1:9,20:60) = 1; % create obstacles on map
map3_1(20:27,30:50) = 1; % create obstacles on map
map3_1(40:60,30:50) = 1; % create obstacles on map
map3_1(80:100,40:50) = 1; % create obstacles on map
map3_1(70:90,50:70) = 1; % create obstacles on map
map3_1(70:80,70:78) = 1; % create obstacles on map
map3_1(30:40,60:70) = 1; % create obstacles on map
map3_1(20:30,80:90) = 1; % create obstacles on map
map3_1(30:40,90:100) = 1; % create obstacles on map
map3_1(50:60,80:90) = 1; % create obstacles on map
map3_1(40:60,70:80) = 1; % create obstacles on map

bug = Bug2(map3_1); % create navigation object
bug.plot(); % Plot ma
p = bug.query(start, goal3_1,'animate'); % animate path
bug.plot(p); % Plot animation
title('Bug2 Algorithm')
xlim([0 100]);
ylim([0 100]);
```

3.2 For the same target cell chosen in 3.1 perform a Distance Transform navigation. Do it manually and with MATLAB. Compare the performance of 3.1 and 3.2.



The distance transform derived a similar path to the manual path without being contrived.

```
%3.2
figure % new figure
dx=DXform(map); % constructs distance transform using map
dx.plan(goal); % plan a path to the goal
p = dx.query(start,'animate') %animate path from start
dx.plot(p) % plor animation on map
xlim([0 10]);
ylim([0 10]);
title('Distance Transform Algorithm
```

3.3 For the sake of referring to specific cells, let us arbitrarily denote the SW corner as {X=1, Y=1}. The SE corner is {X=10, Y=1} and the NE corner is {X=10, Y=10}. Let the NE corner be the target point for the navigation of the robot. Assuming that initially the road conditions were excellent everywhere, a tentative navigation plan had been prepared using the D* algorithm.

During the night a severe storm hit the whole grid, left several bad road conditions, and there is a need to modify the navigation plan. Here is an advisory list of all the troublesome locations:

{1,6} AND {1,7}: 1ft deep puddles

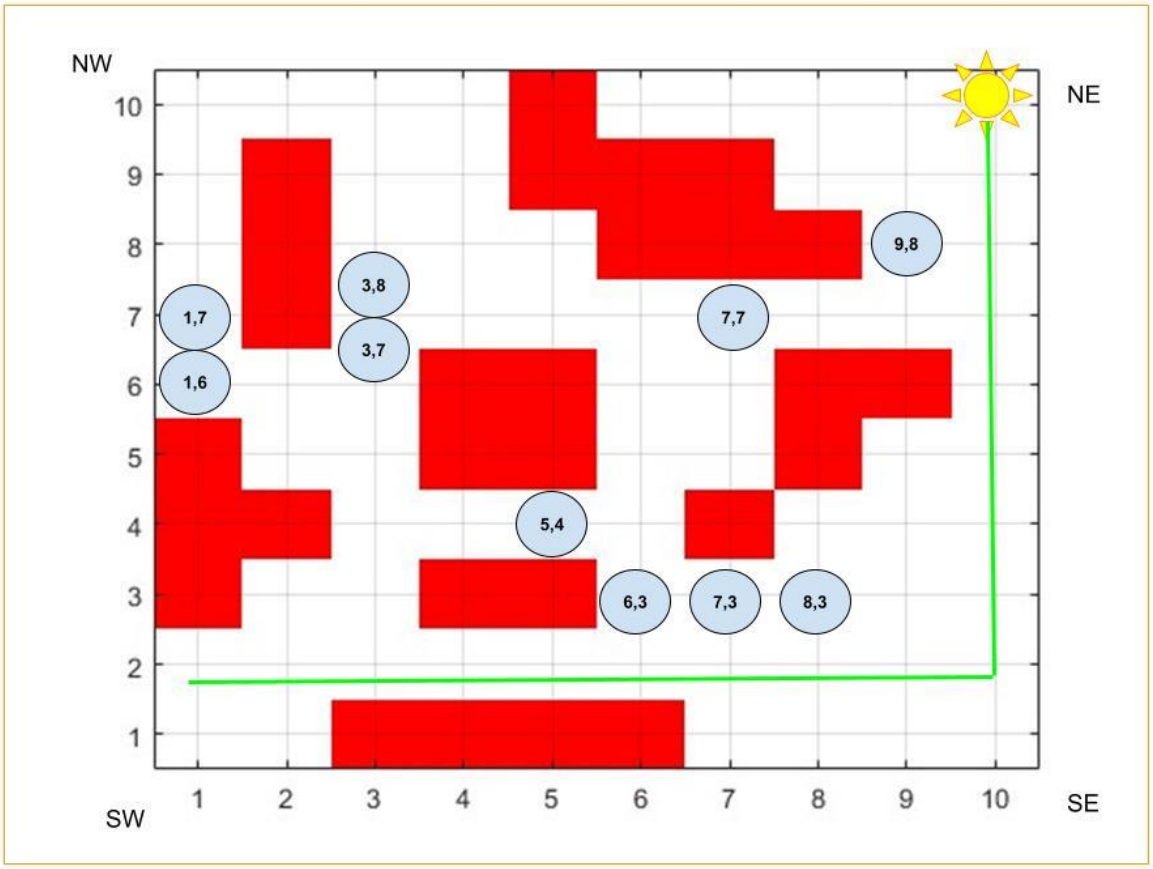
{3,7} AND {3,8} AND {4,7}: Fallen trees blocking the road

{5,4}: 3ft deep puddles

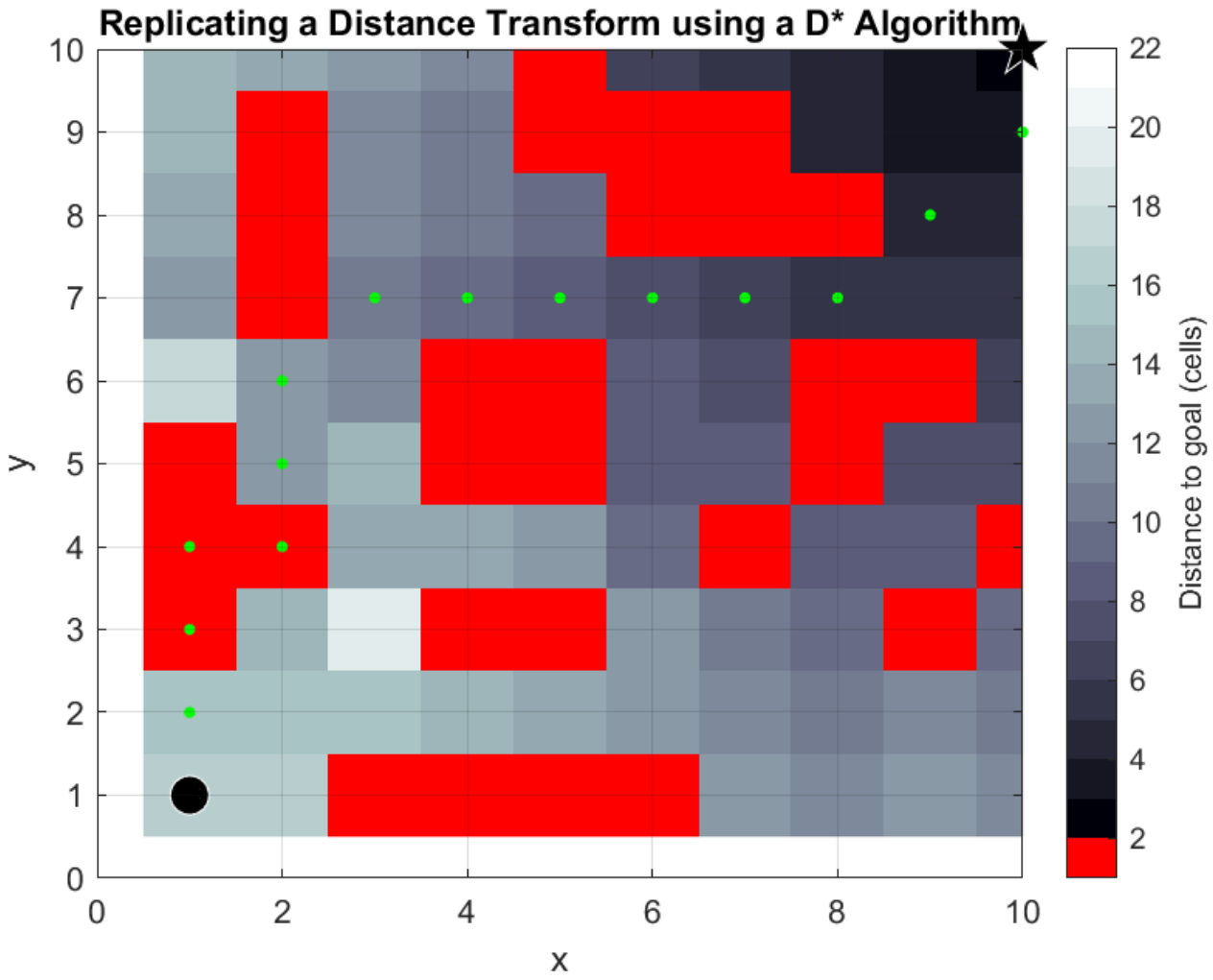
{6,3} AND {7,3} AND {8,3}: Collapsed buildings blocking the road

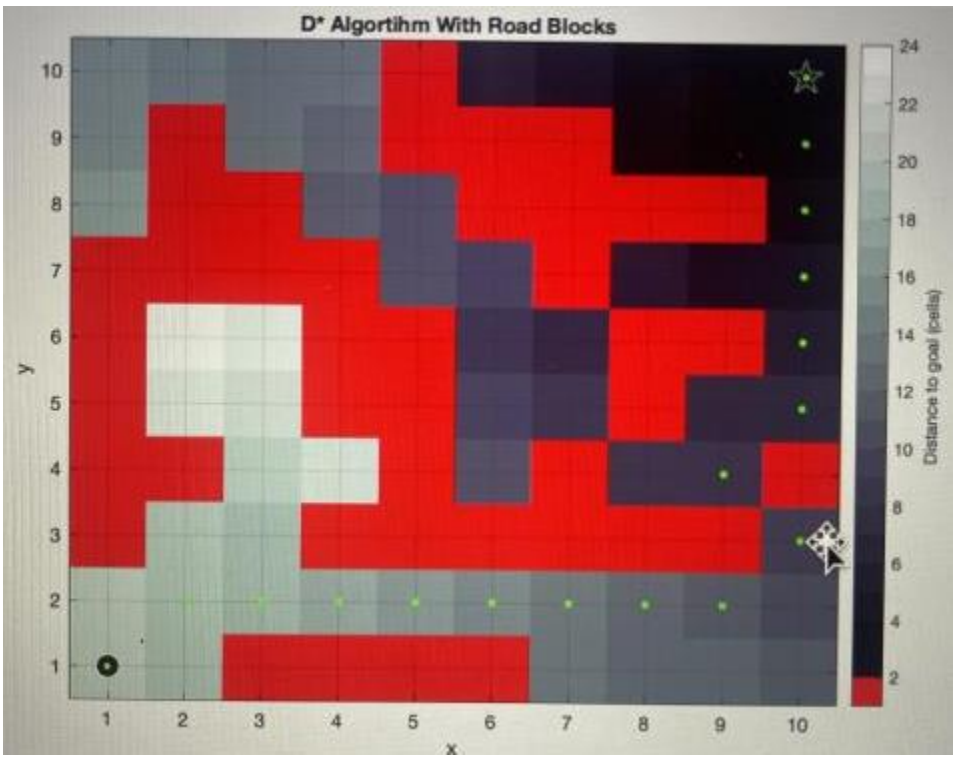
{7,7}: Some debris on the road

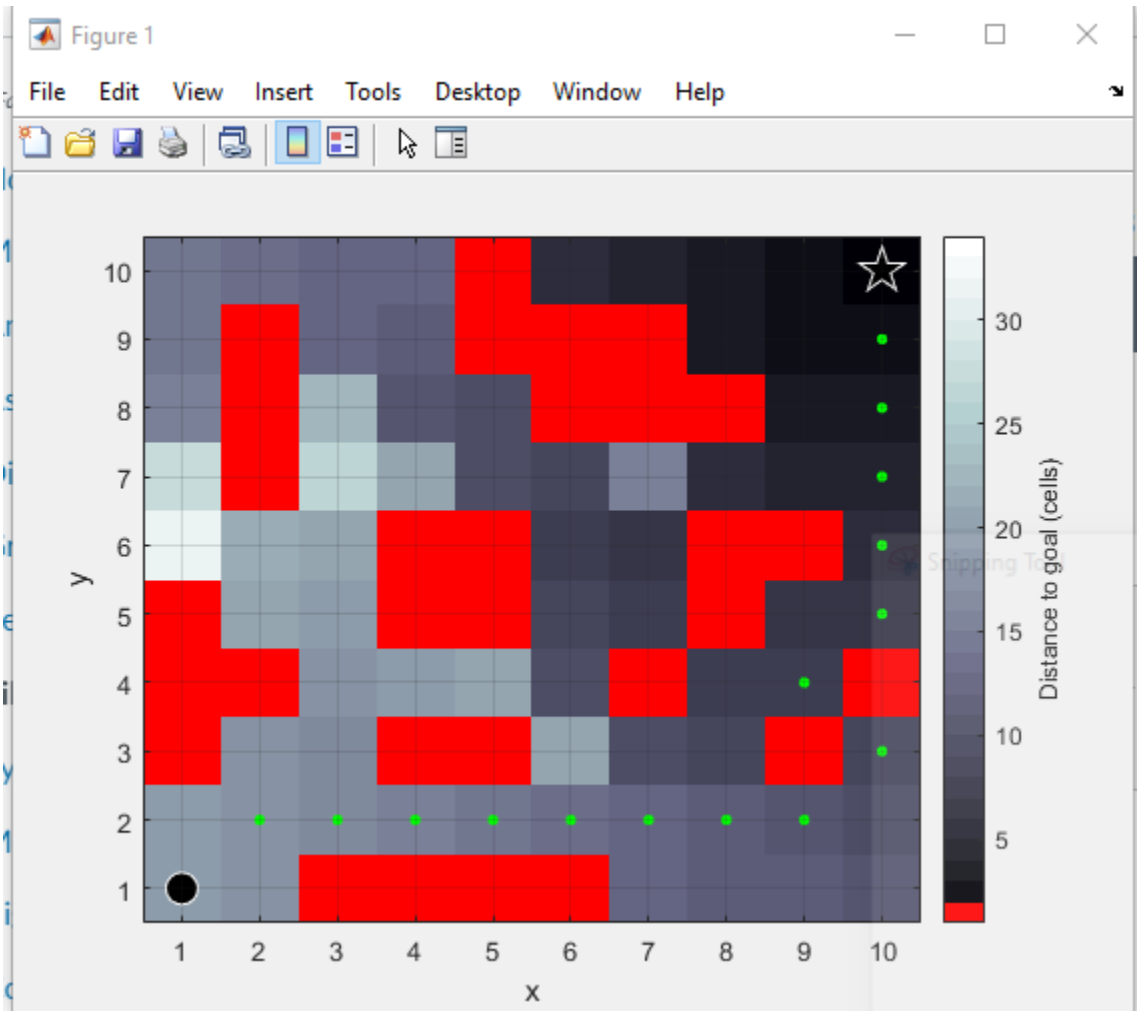
{9,8}: 1ft deep puddles.



Show manually the original and the modified navigation paths, and then do it with MATLAB. Compare the performance of the two plans.


Replicating a Distance Transform using a D* Algorithm
y
x
Distance to goal (cells)
0
2
4
6
8
10
22
20
18
16
14
12
10
8
6
4
2


D* Algortihm With Road Blocks
y
x
Distance to goal (cells)
1
2
3
4
5
6
7
8
9
10
2
4
6
8
10
12
14
16
18
20
22
24



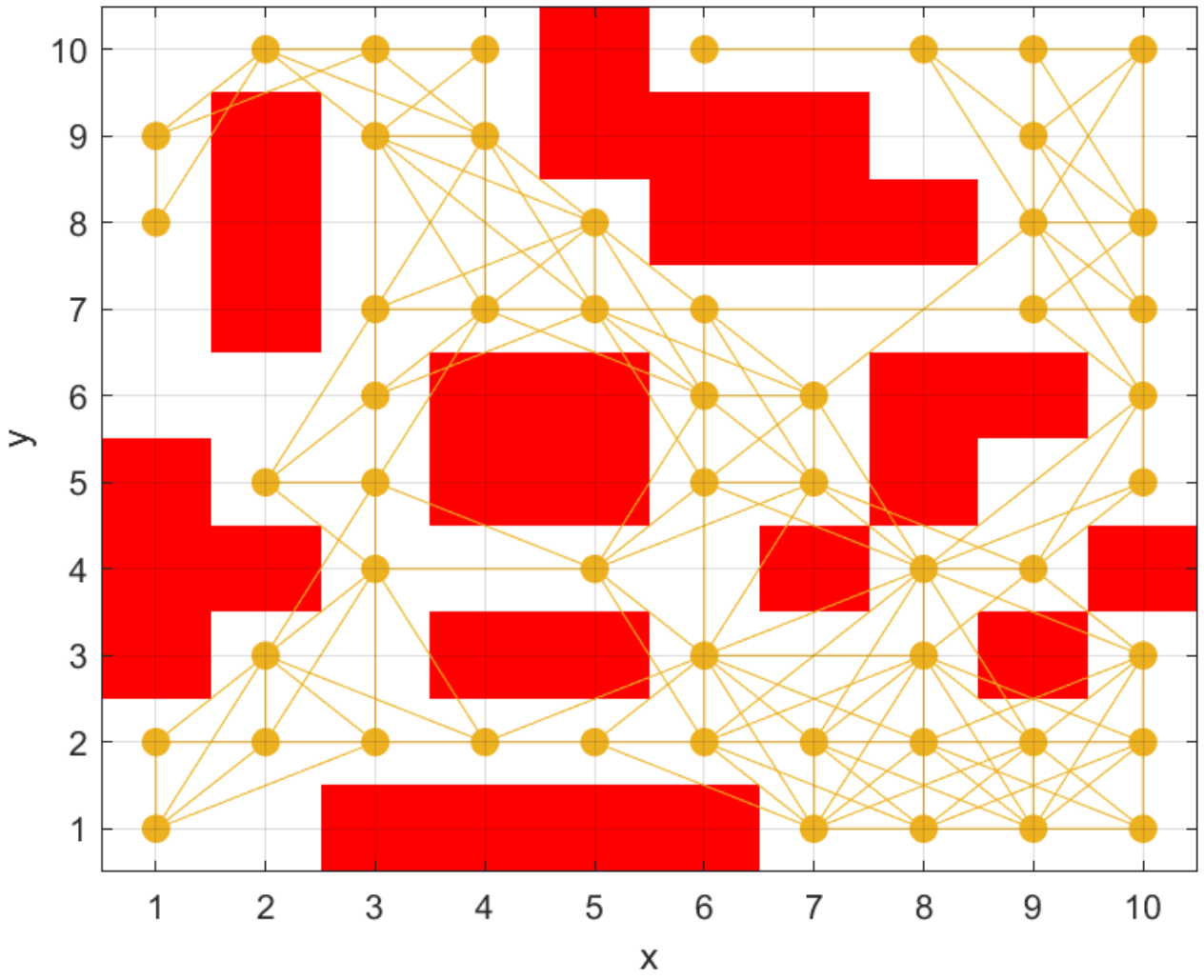
I implement a few different courses

```
%3.3 Alternative
%this method is used to meet the no diagonal motion requirement
ds=Dstar(map,'cityblock'); % create navigation object
c=ds.costmap(); % load costmap
ds.plan(goal); % plan path to goal
figure % new figure
ds.query(start,'animate') % animate path from start to goal
xlim([0 10]);
ylim([0 10]);

%Modify cost of each cell on map to create the desired distance transform
%map

cell12=[1;2] ;
cell13=[1;3] ;
cell14=[1;4] ;
cell16=[1;6] ;
```

```
cell18=[1;8] ;
cell1=[1;10] ;

ds.modify_cost(cell12,2);
ds.modify_cost(cell13,1);
ds.modify_cost(cell14,0);
ds.modify_cost(cell16,10);

cell21=[2;1] ;
cell22=[2;2] ;
cell23=[2;3] ;
cell24=[2;4] ;
cell25=[2;5] ;
cell26=[2;6] ;

ds.modify_cost(cell21,0);
ds.modify_cost(cell22,2);
ds.modify_cost(cell23,2);
ds.modify_cost(cell24,0);
ds.modify_cost(cell25,1);
ds.modify_cost(cell26,0);

cell32=[3;2]
cell33=[3;3]
cell35=[3;5]
cell36=[3;6]

ds.modify_cost(cell32,1);
ds.modify_cost(cell33,10);
ds.modify_cost(cell35,3);
ds.modify_cost(cell36,3);


cell42=[4;2]
cell43=[4;3]
cell45=[4;5]
cell46=[4;6]

ds.modify_cost(cell42,1);
ds.modify_cost(cell43,10);
ds.modify_cost(cell45,3);
ds.modify_cost(cell46,4);


cell51=[5;1] ;
cell52=[5;2] ;
cell56=[5;6] ;

ds.modify_cost(cell51,10);
ds.modify_cost(cell52,1);
ds.modify_cost(cell56,5);


cell61=[6;1] ;
cell62=[6;2] ;
```

```
cell63=[6;3] ;
cell64=[6;4] ;
cell65=[6;5] ;

ds.modify_cost(cell61,10);
ds.modify_cost(cell62,2);
ds.modify_cost(cell63,3);
ds.modify_cost(cell64,4);
ds.modify_cost(cell65,5);

ds.plan(); % update plan
p=ds.query(start); % animate start to goal
ds.plot(p) % plot animation
xlim([0 10]);
ylim([0 10]);
title('Replicating a Distance Transform using a D* Algorithm')
```

3.4 Assuming perfect driving conditions, use MATLAB to create a Probabilistic Road Map for the grid, with as few as you can take number of "train stations" (or, more correct mathematically, joints along the Voronoi diagram). Then let MATLAB find the path from the SW corner to the NW corner, and also the path from the NE corner to the SE corner.



3.5 (1% optional bonus programming activity): Create a grid map that describes a section of a city (like New York). This city segment has some 8-10 avenues, and some 30-40 streets.

Like in NYC, avenues have bi-directional traffic, going north-south. The hardest part of this assignment is to figure out how to implement one-way streets.

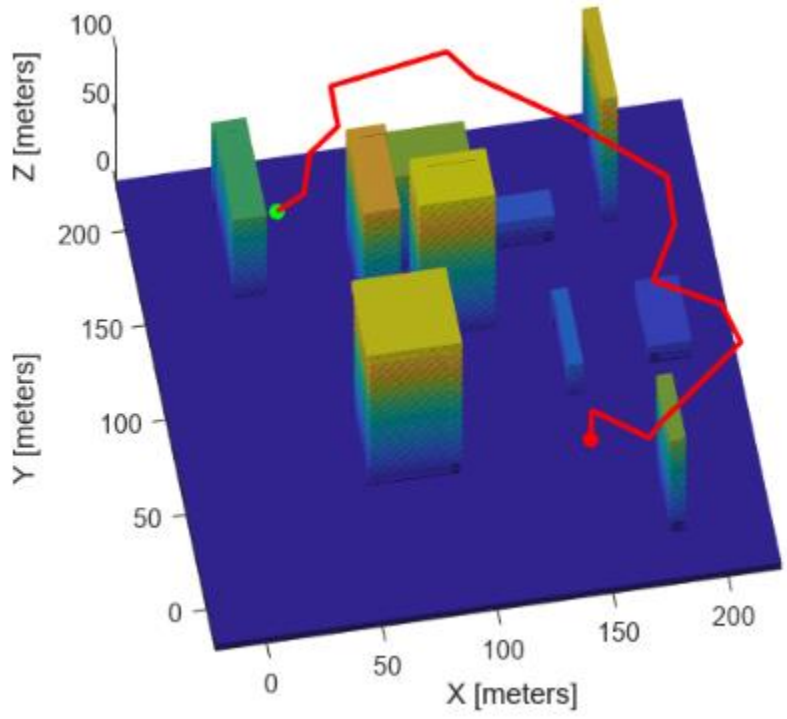
Every 6th street is considered a major street, and it intersects each of the avenues at a traffic light. Drivers who drive on a street that is not a major street must turn right whenever reaching one of the avenues. The traffic light intersections tend to be very busy all day long, and most of the time only one car (per traffic light cycle) can make a left turn from an avenue to a major street. It may be faster to drive forward on the avenue one more street, turn right into it, then another right at the

next avenue, and finally one more right turn brings the driver to the original intersection, but at a more favorable status: Instead of making a left turn, they now go straight through the intersection.

Much of the traffic arrives at the grid from the SE direction, heading towards the NW part of the grid. Also be sure to include several recreation parks (in which there is no cars traffic). Each park occupies a block of two avenues and three streets area. One of the parks lies near the SW side of the grid, and another is near the grid's center.

Create a D* navigation plan from the SE corner to the NW corner. Then assume that a car accident happens at one of the intersections along your planned path. Show how to modify the driving plan in real time.

Nothing in this project should be done manually. Write an annotated MATLAB code. Plot your grid, the cost map, the original path, the modified cost map and the modified path.



Submission Deadline: F 10/21/2022 by 11:59 PM.