FLORIDA ATLANTIC UNIVERSITY

Jancarlos Gomez

EEL5661

Homework 2

12 October 2021

Dr. Roth

Problem 1: Trajectory Planning

1.1 A 3P3R industrial robot has XYZ prismatic joints to determine the position of its wrist center. The wrist itself has three revolute joints built to create roll-pitch-yaw (RPY) angles. We need to plan a trajectory for the robot to go from homogeneous transformation A to homogeneous transformation B, given below, in no more than 2 seconds. The units of the X, Y, and Z motions are [m]. The units of the roll, pitch, and yaw angles are [rad]. Use a MATLAB RTB mtraj command, featuring 6 parallel and synchronized tpoly one-dimensional motion planners.

$$A = \begin{bmatrix} 0.9256 & -0.2427 & 0.2904 & 0.8000 \\ 0.2863 & 0.9508 & -0.1181 & -0.7000 \\ -0.2474 & 0.1925 & 0.9496 & 0.5000 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 0.9890 & -0.0998 & 0.1092 & 0.5000 \\ 0.0992 & 0.9950 & 0.0110 & 0.1000 \\ -0.1098 & 0 & 0.9940 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Plot the six trajectories: X(t), Y(t) and Z(t) in one plot and the RPY angles in a second plot.

Below is the MATLAB code to plot the six trajectories:

```
clear all;clc; close all;
%Jancarlos Gomez
%Z23521760
%Problem 1
%1.1
tt = 0:0.1:2  % define the time range; robot must go from A to B in 2 seconds
A = [0.9256 -0.2427 0.2904 0.8000; 0.2863 0.9508 -0.1181 -0.700; -0.2474 0.1925 0.9496 0.5000; 0 0 0 1]
B = [0.9890 -0.0998 0.1092 0.5000; 0.0992 0.9950 0.0110 0.1000; -0.1098 0 0.9940 0; 0 0 0 1]

PosA = [0.8 -0.7 0.5] % X Y Z positions of transformation A
PosB = [0.5 0.1 0] % X Y Z positions of transformation B
qpos = mtraj(@tpoly,PosA,PosB,tt) % generate the trajectories of X(t), Y(t), and Z(t)
plot(qpos)
title("Trajectories of X(t), Y(t), and Z(t)")
xlabel("Time")
ylabel("Joint Movement in Meters")
legend({'X Postion ','Y Position','Z Position'},'Location','northeast')
set(gca,'XTick', [1:21]) % remap the current x-axis on the plot to match 0 to 2 seconds
set(gca,'XTickLabel', [tt])

figure
RPYA = tr2rpy(A,'xyz') % obtain the roll, pitch, and yaw angles of transformation A
RPYB = tr2rpy(B,'xyz') % obtain the roll, pitch, and yaw angles of transformation B
qrpy = mtraj(@tpoly,RPYA,RPYB,tt) % generate the trajectories of the RPY angles
plot(qrpy)
title("Trajectories of the RPY angles")
xlabel("Time")
ylabel("Joint Movement in Rad")
legend({'Roll (X)','Pitch (Y)','Yaw (Z)'},'Location','northeast')
set(gca,'XTick', [1:21]) % remap the current x-axis on the plot to match 0 to 2 seconds
set(gca,'XTickLabel', [tt])
pause
```

Figure 1 MATLAB Code for 1.1

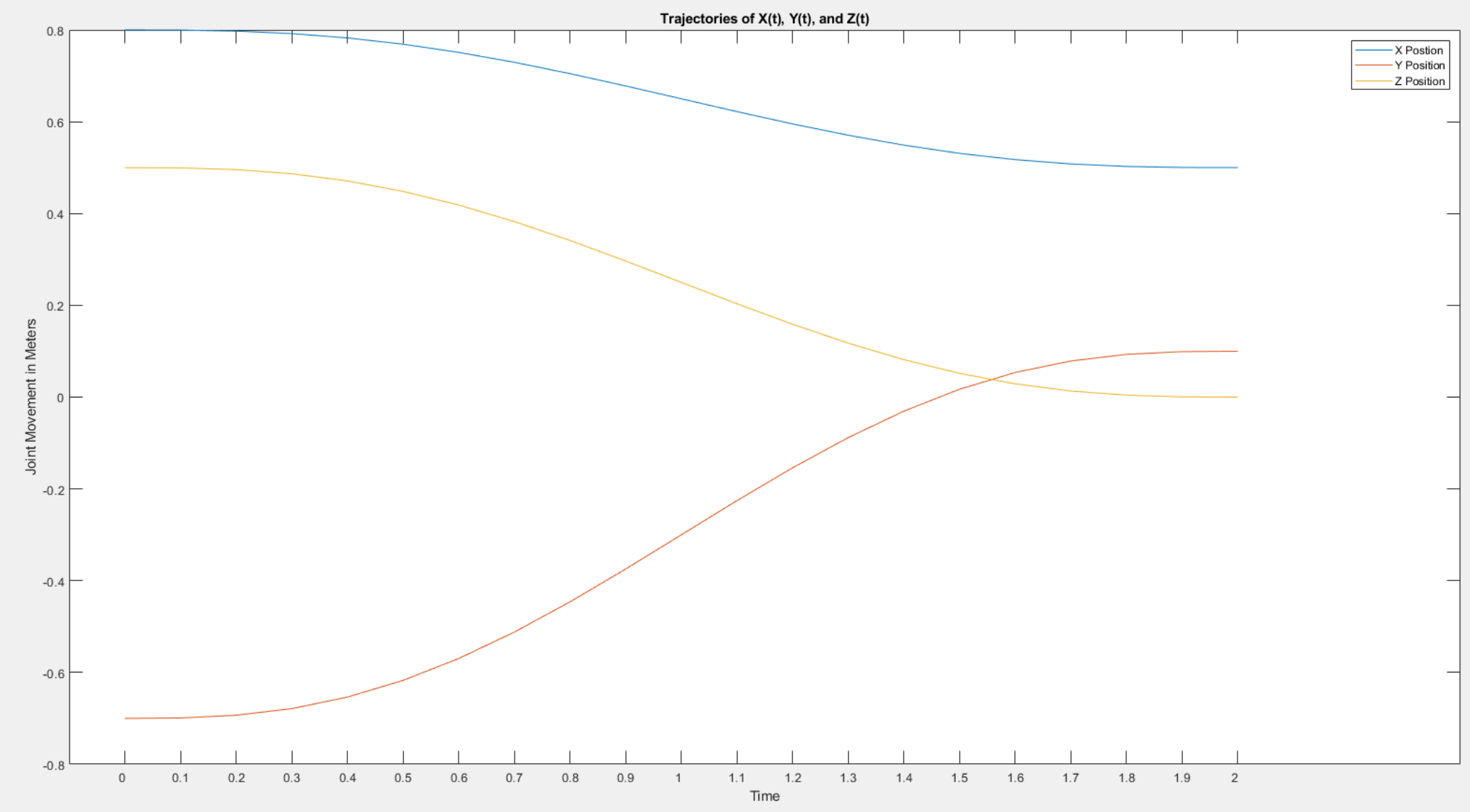


Figure 2 Plot of Trajectories for X(t), Y(t), Z(t)

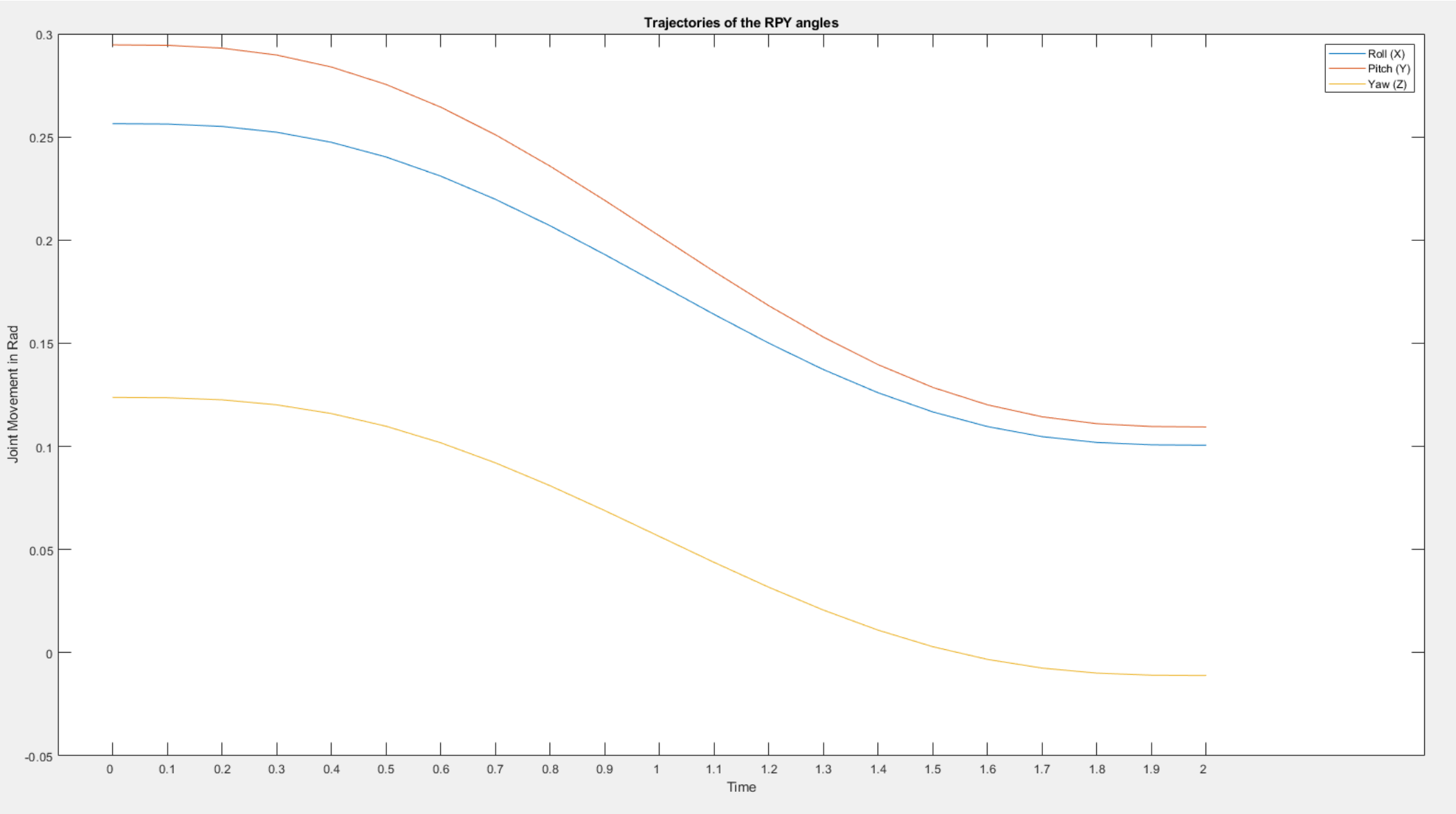


Figure 3 Plot of Trajectories for RPY Angles

As we can observe in figures 2 and 3, each trajectory goes from its starting point in Transformation A to its endpoint in Transformation B in 2 seconds.

1.2 The pose of a mobile robot (in world coordinates) is (x, y, θ) where (x,y) is the position of the origin of the vehicle's local coordinate frame, with respect to the world coordinate frame. The angle θ is the turning angle of the vehicle's local X axis with respect to the world's X axis. The units of x and y are 1 unit = [1 hundred yards] and the units of the θ are [rad]. Use MATLAB RTB mstraj command to plan and plot x(t), y(t) and θ(t), going from a stopping endpoint A = (0, 0, 0) to a via point B = (2, 4, π/4) and then to another via point C = (7, 3, π/2) and finally to an endpoint D = (10, 3.5, π). The vehicle's maximum velocity is Vmax = 10 yards/s and the acceleration time from zero to maximum speed is tacc = 2 seconds.

Below is the MATLAB code for the trajectories of x(t), y(t), and θ(t).

```
%1.2
via = [0 0 0; 2 4 pi/4; 7 3 pi/2; 10 3.5 pi]'; % use variable via to
%describe the coordinates of each point
ql= mstraj(via(:,[1 2 3 4])',[0.1,0.1,0.1],[],via(:,1)',1,2); %define acceleration time,
%maximum velocity, and the order of the via points
figure
plot(ql)
grid on
title("Trajectories of x(t), y(t), θ(t)")
xlabel("Time")
ylabel("Movement")
legend({'X Position (yd)','Y Position (yd)','Turning Angle with Respect to X Axis (rad)'},'Location','northwest')
```

Figure 4 MATLAB Code for 1.2

MATLAB File

Problem1.m

In figure 5, we can observe that x(t), y(t), and θ(t) go to the four via points successfully; however, the reason the time it takes to get to each via point is longer than I expected is due to the acceleration time and changing speed. This is not a constant speed and acceleration problem; thus we can expect more time is needed to reach each via point as shown in figure 5. When y(t) changes direction we can see a plateau occur momentarily as it accelerates to change direction, which adds to the time needed to get to the next via point. We can see that the movement of x(t), y(t), or θ(t) can bottleneck how long it takes to get from one via point to another since they are not all moving at the same speed.

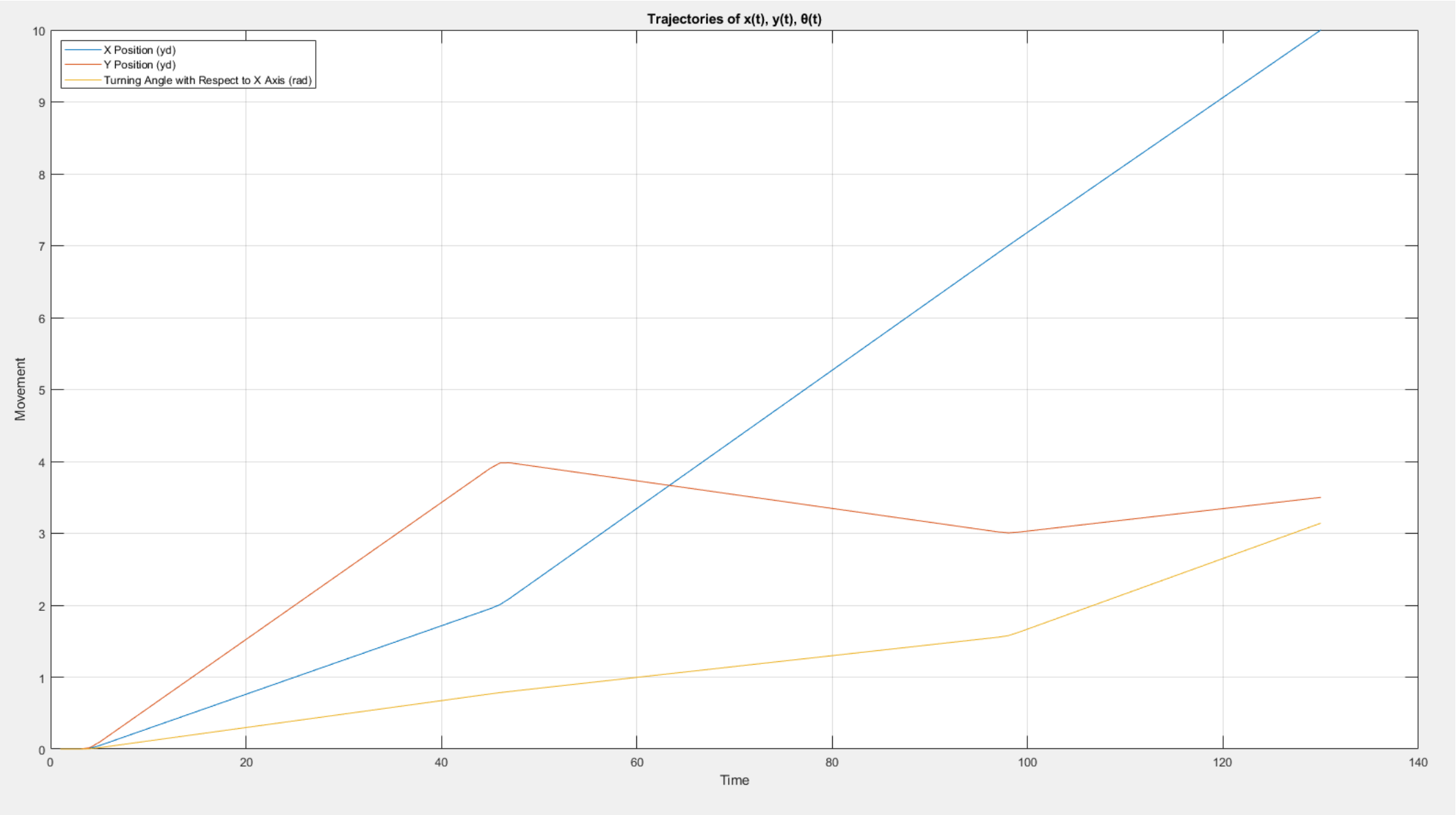
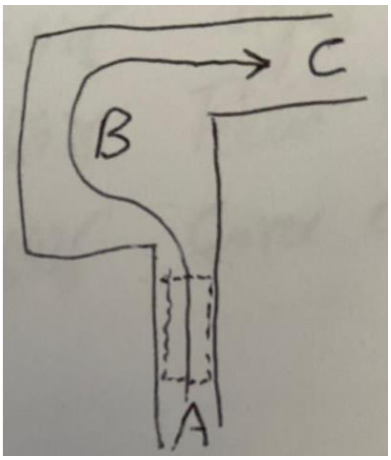


Figure 5 Plot of x(t), y(t), and θ(t) from Starting Point to End Point

Problem 2: Wheeled Robotic Vehicles

A wheeled mobile robot travels along a narrow corridor A, intending to turn right into another narrow corridor C. There is a wide hall B connecting corridor A to corridor C (see a sketch, below). The vehicle's width and length dimensions are 2'W by 4'L. Corridors A and B are 3' wide, each. Hall B is 6' wide and 6' long.



2.1 The shown trajectory (turning to the left at a circle, in order for the vehicle to align itself with corridor C on the right) seems suitable for bicycle-type or tricycle-type wheeled vehicles. Pick up either one. Create a Simulink model for the vehicle and demonstrate its motion. Assume that the distance between the back wheels axel and the front wheel center is 3'.

For this problem, I decided to model the corridor and vehicle in SolidWorks in order to find the suitable path the vehicle should take and make sure that the vehicle does not hit the walls when turning. The corridor is quite tight making the vehicle have a very limited path to take. Using trial and error I was able to find a few suitable points that the vehicle must pass by to successfully go from A to C without crashing into a wall. I used the measure tool in SolidWorks to determine the x and y coordinates for the vehicle. The origin of this trajectory is the front center face of the vehicle. I assumed that the vehicle is 1 foot behind the B area. The x-axis ranges from -4.5' (left wall of B area) to 5.5' (enough room for the vehicle to be in the C area) and the y-axis ranges from 0' to 7' (top wall of B/C area). The front and back wheels are separated by a distance of 3'. To be symmetrical I assumed that the wheels are 0.5' inwards from the front and back faces. For 2.1, I decided to show two methods of obtaining the result. The first method is the one shown in class using a bicycle model with simple input signals. The only way I could form a similar shape as the one I found in SolidWorks was by messing with the gamma input and putting random values until I get a similar result. I added an additional pulse generator in order to make the second turn. On the other hand, method 2 which I prefer much more and is more accurate and does not require guesswork was more successful. This method uses the Robotics System Toolbox and Mobile Robotics Training Toolbox. For problem 2.2 I decided to use the toolboxes aforementioned as well since the method shown in class is confusing and less efficient. The general trajectory taken is described below in SolidWorks:

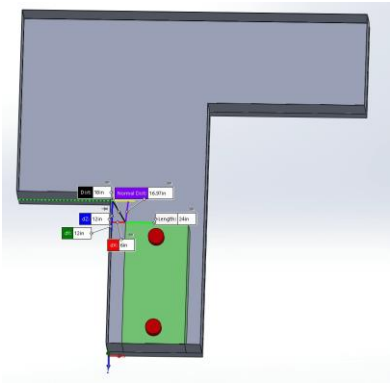


Figure 6 Starting Point of Vehicle (0, 0)

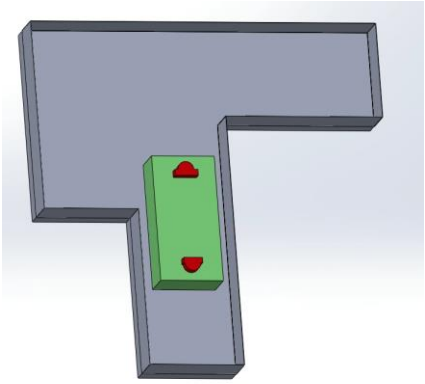


Figure 7 Vehicle Moves to (0, 2) to Prepare for Turning

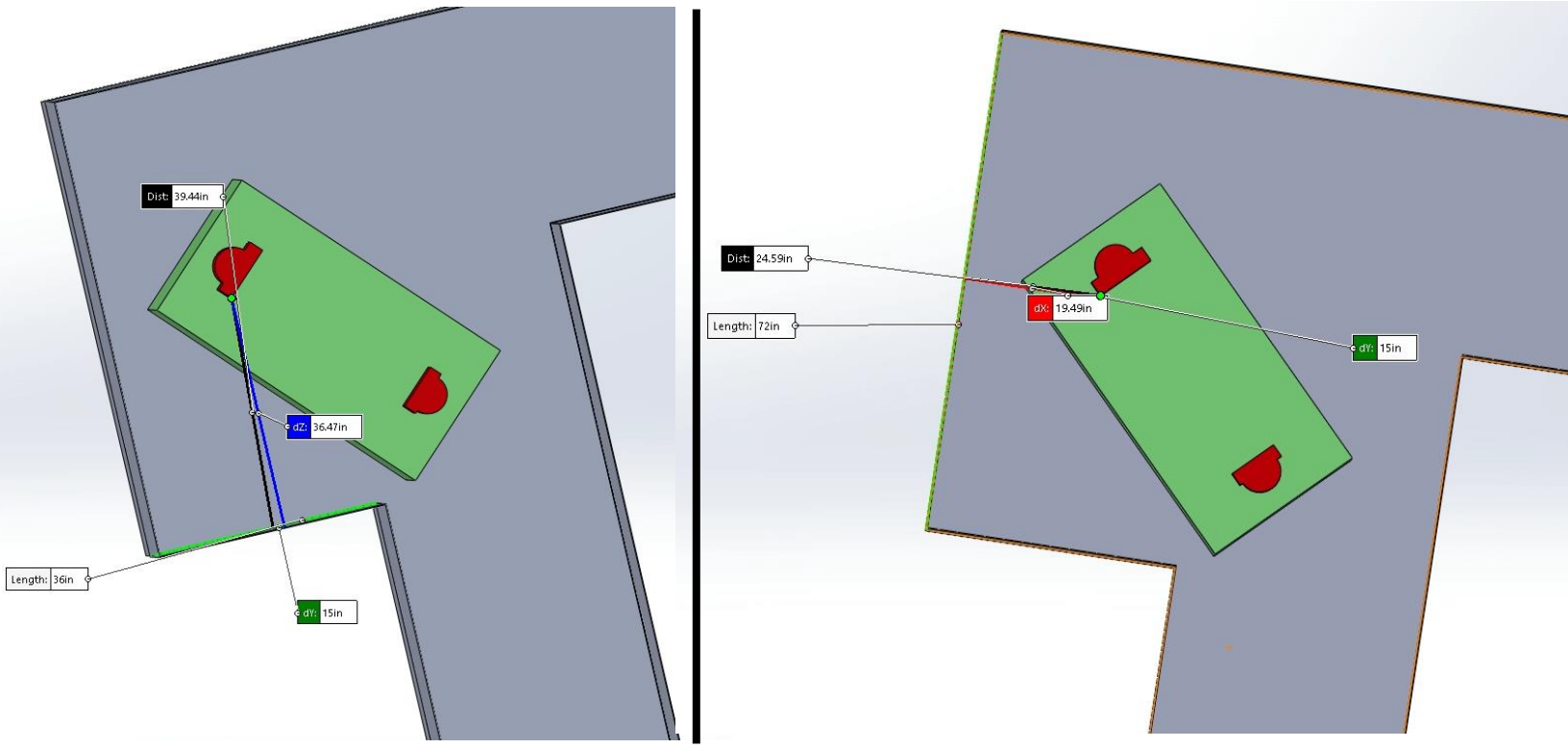


Figure 8 Vehicle Front Wheel Axel Center Moves to (-2.875, 4)

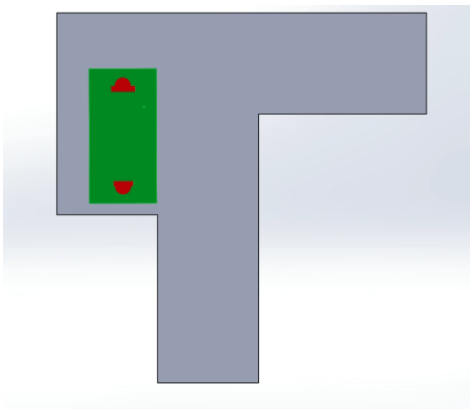


Figure 9 Back Axel Aligns with Front Axel Preparing for Next Turn

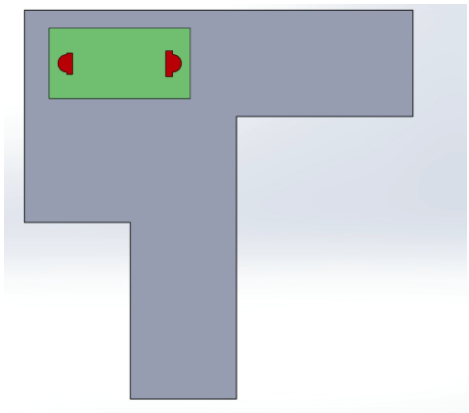


Figure 10 Vehicle Move Towards (-2.375, 5.5)

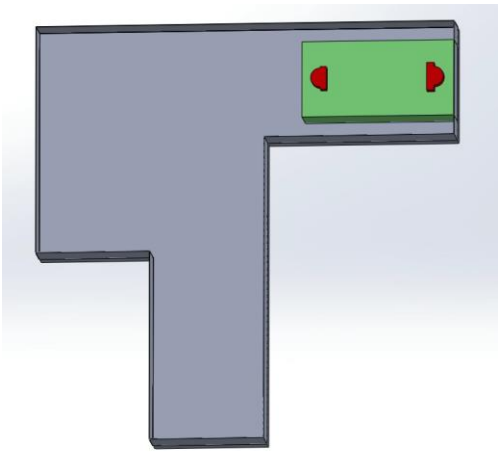


Figure 11 Ending Point of Vehicle (6.5, 5.5)

In the figures above we can observe the general trajectory, I want the vehicle to take. If the Simulink models, follow this trajectory they will not crash into the walls and they will get to the final destination successfully. In figure 12, we can observe my modification of the Bicycle with simple inputs. This method obtained an XY plot similar to what we are looking for but not quite. Method 2 is much better and easier to tweak.

Method 1 (Shown in Class):

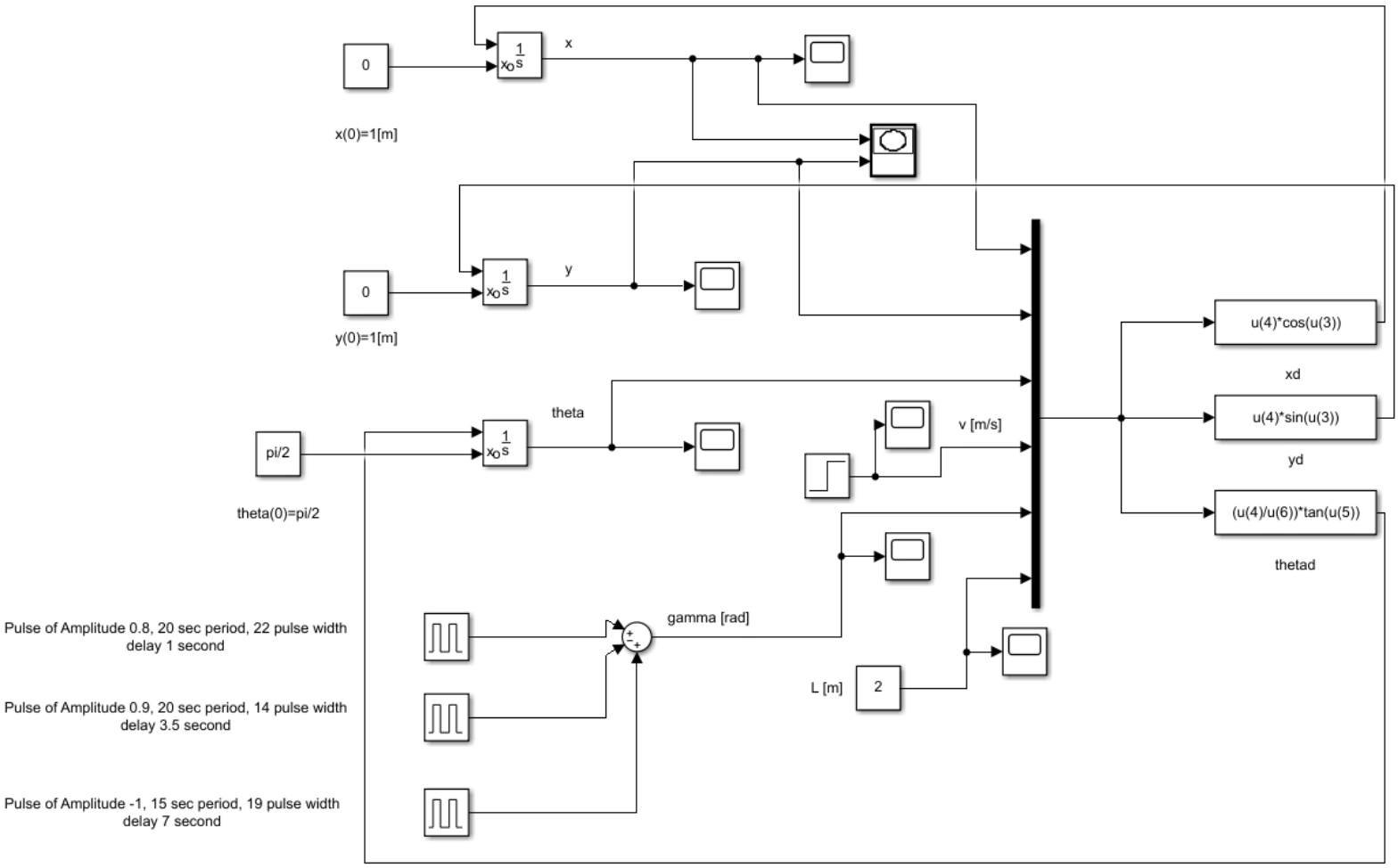
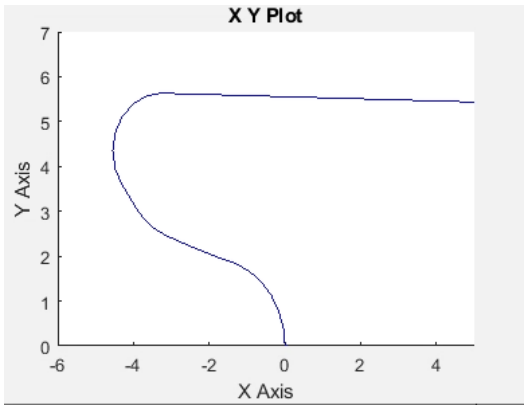
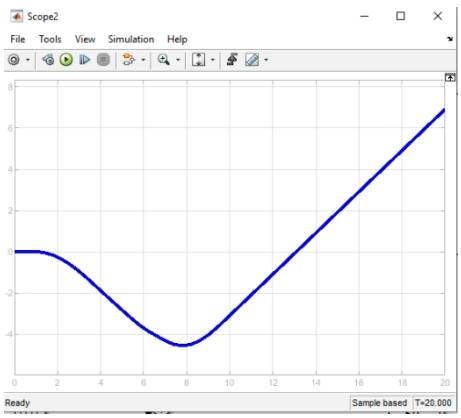


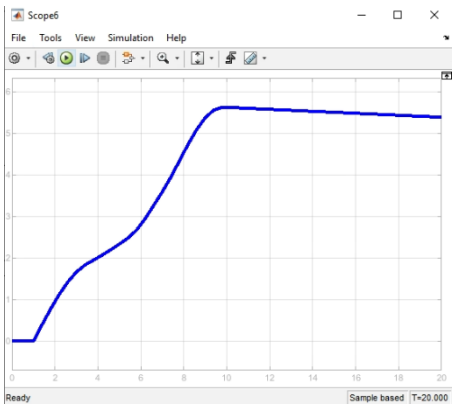
Figure 12 Bicycle with simple input signals



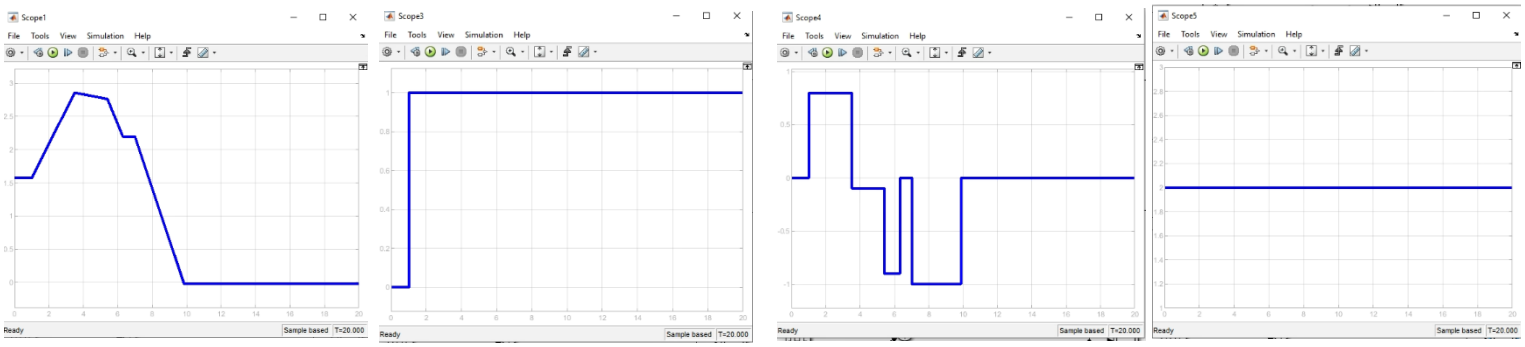
X Y Plot



x(t) Plot



y(t) Plot



Theta Plot **V [m/s] Plot** **Gamma Plot** **L [m] Plot**

Method 2 (Using Robotics System Toolbox and Mobile Robotics Training Toolbox):

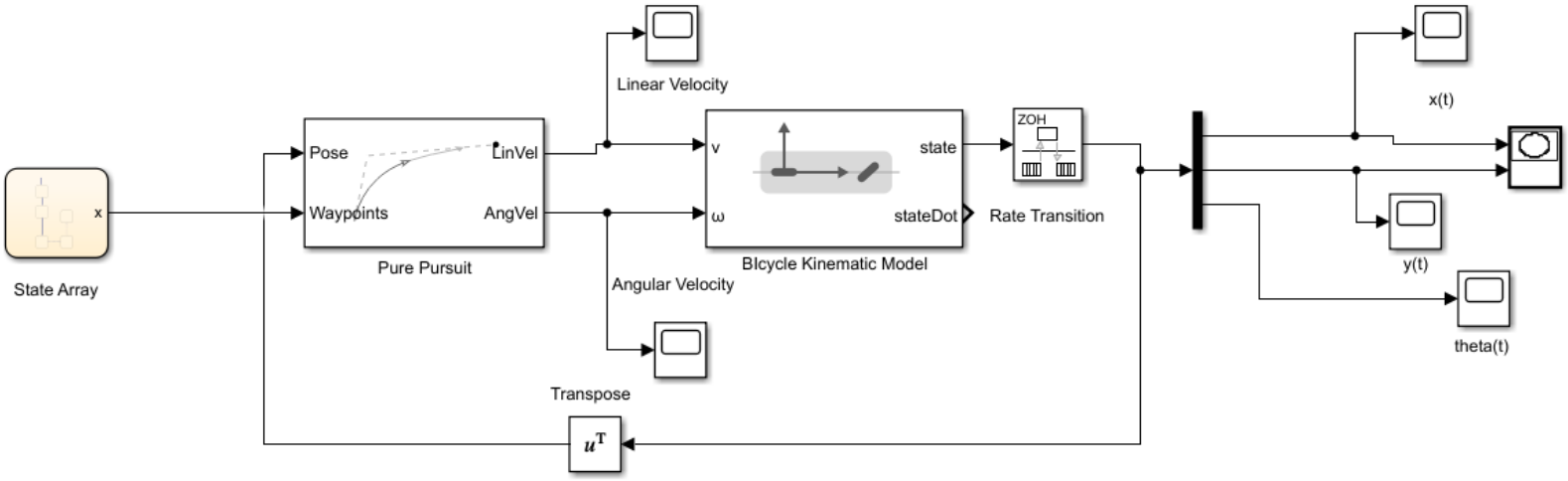


Figure 13 Simulink Model for Vehicle Using Bicycle Kinematic Model

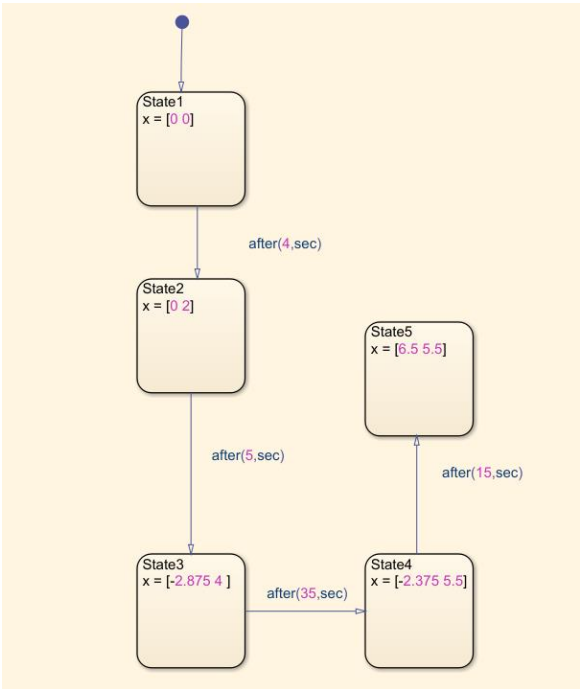
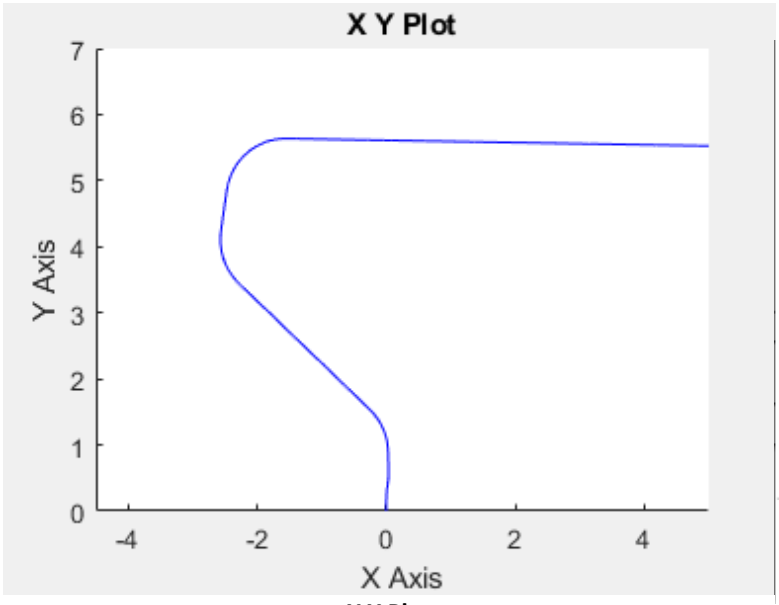
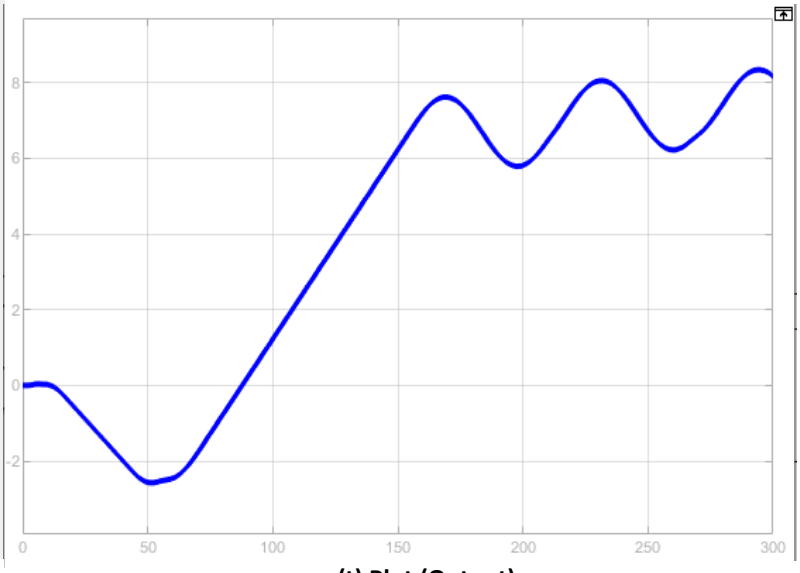


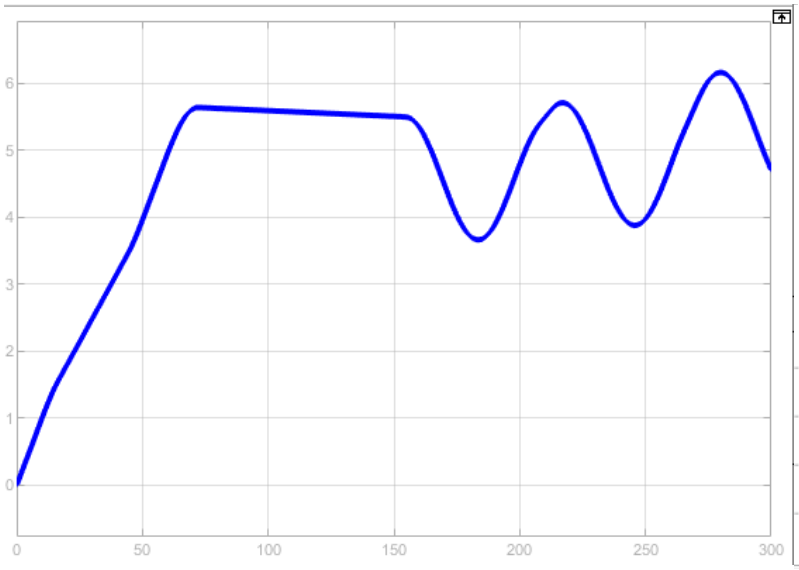
Figure 14 Inside the State Array



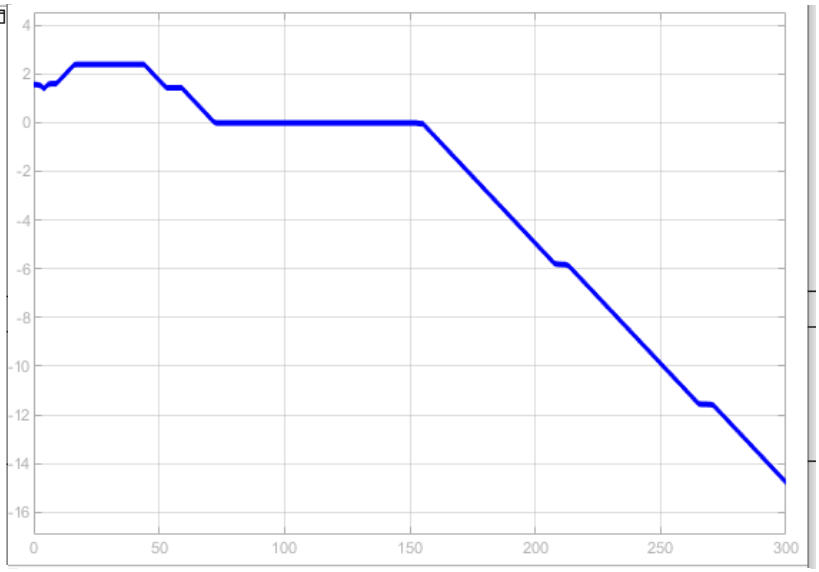
X Y Plot



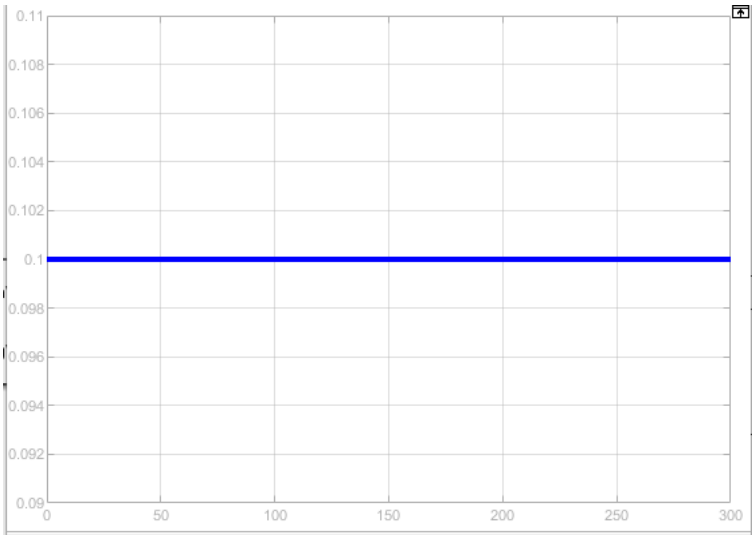
x(t) Plot (Output)



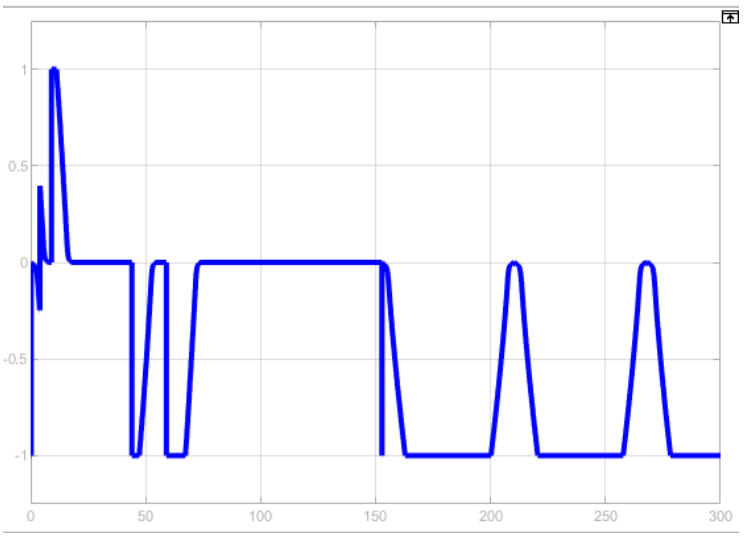
y(t) Plot (Output)



θ(t) Plot (Output)



Linear Velocity Plot (Input)



Angular Velocity Plot (Input)

We can observe in the plots above that the vehicle goes from A to C without crashing into a wall. The theta plot also illustrates the two turns taken in order to get to the final destination.

Simulink File

problem2_1.slx

2.2 Assume now that the vehicle is of a differential-drive-type. Create a Simulink model for the vehicle. Plan and demonstrate its motion, which can be different than that used for the robot of (2.1).

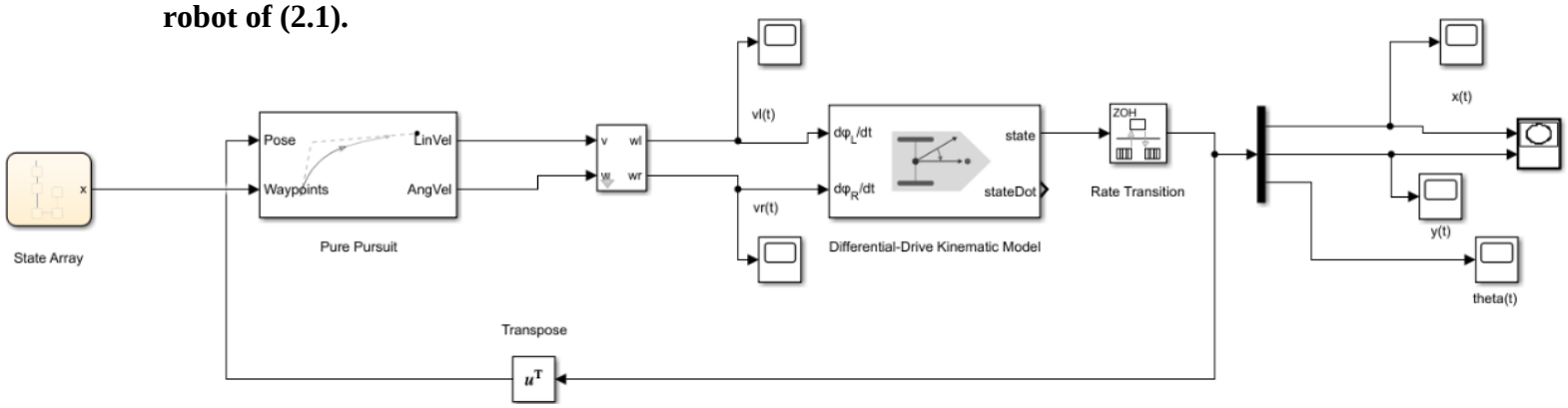


Figure 15 Simulink Model for Vehicle Using Differential-Drive Kinematic Model

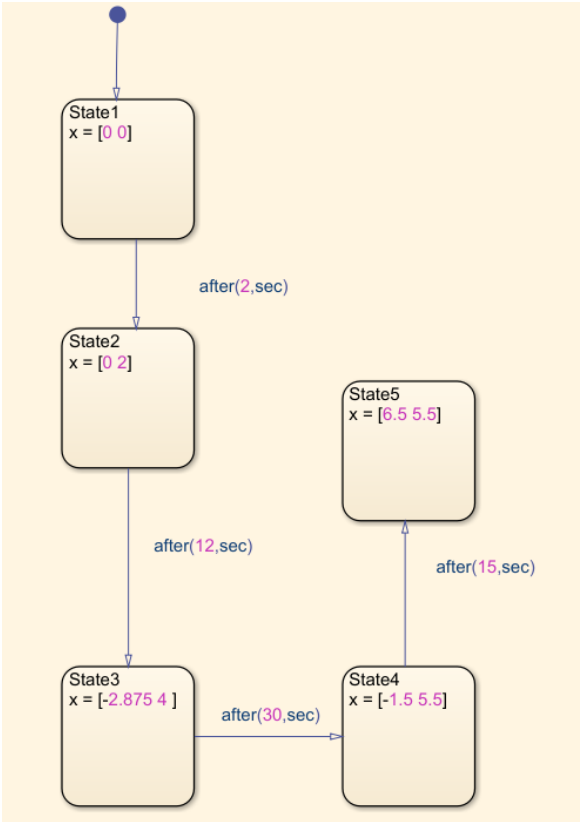
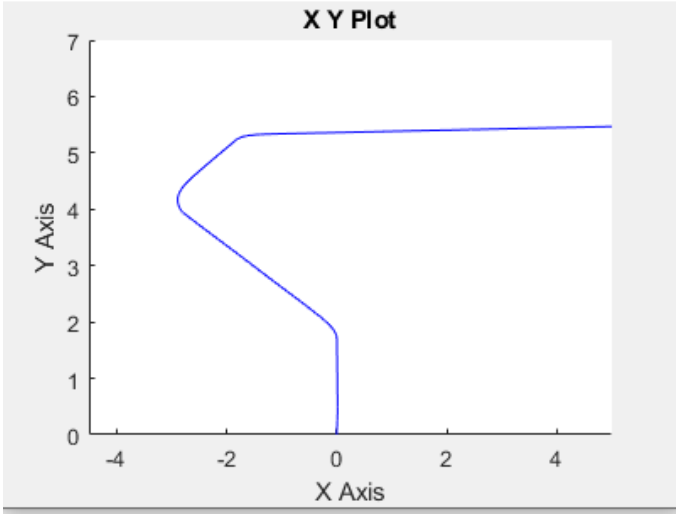
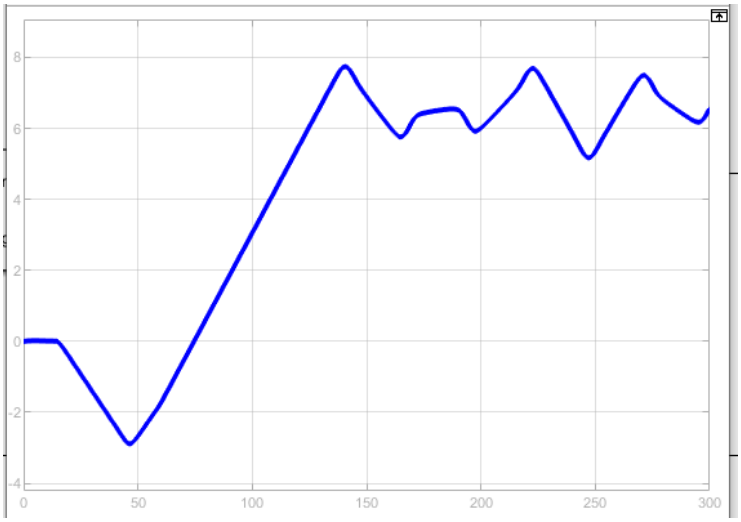


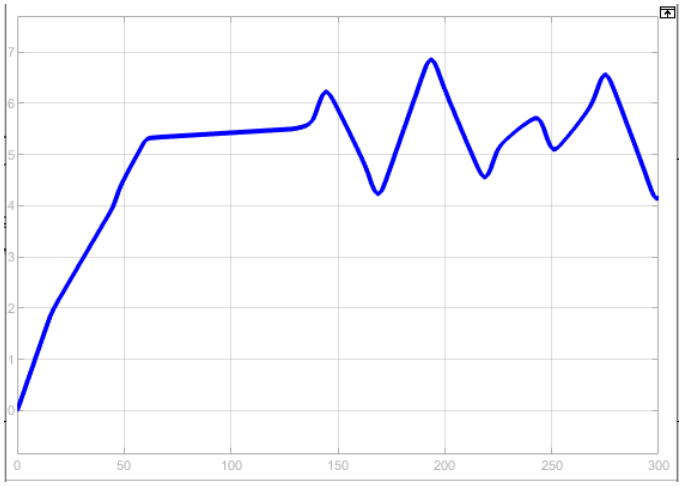
Figure 16 Inside the State Array



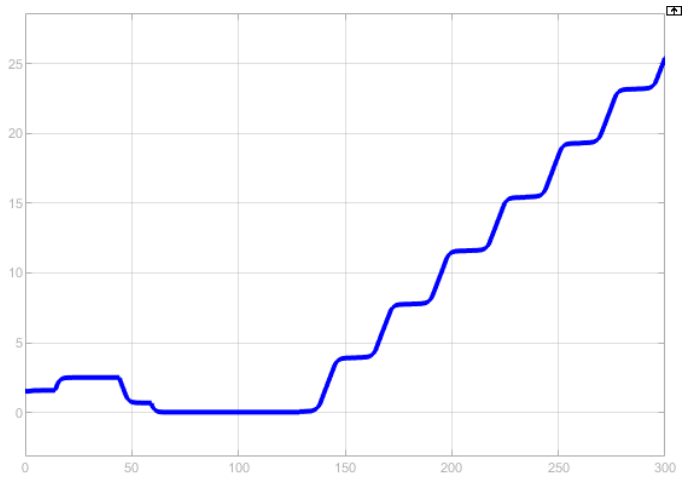
X Y Plot



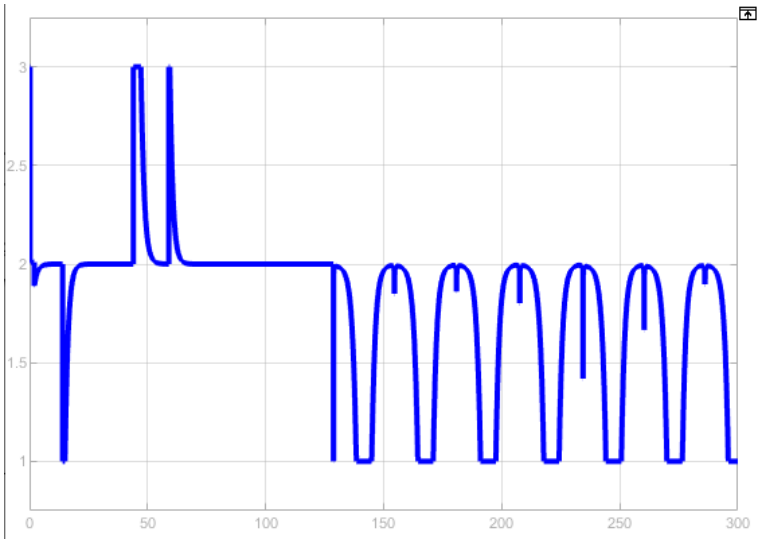
x(t) Plot (Output)



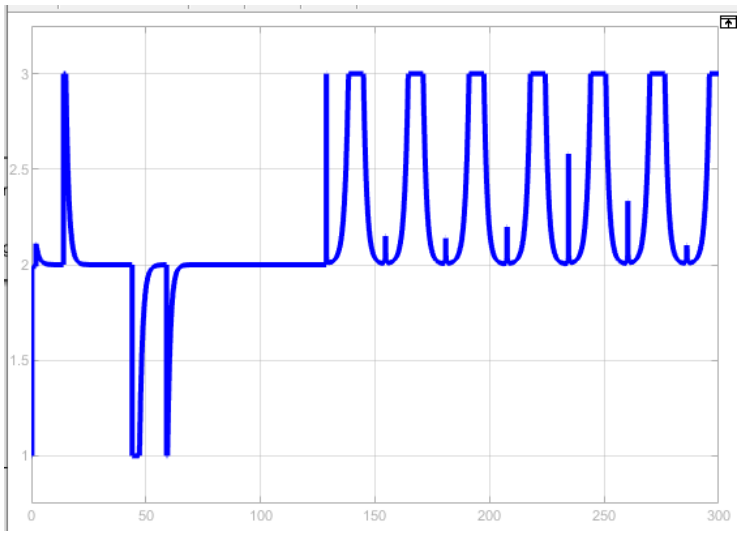
y(t) Plot (Output)



θ(t) Plot (Output)



vl(t) Plot (Input)



vr(t) Plot (Input)

Problem 2.2 uses a similar approach as 2.1 for the Simulink model; however, the inputs needed to be changed because the Bicycle Model generates a smoother XY plot compared to the Differential-Drive Model. In the XY plot of the Differential-Drive Model, we can observe that the turns are much sharper when compared to the XY plot of the Bicycle Model. This could be attributed to the number of wheels turning at once. Having multiple wheels turn instead of one can generate a sharper turn. For example, making a sharp turn on a bicycle is not recommend as the user will most likely lose stability and fall but in a 4-wheeled vehicle sharp turns are easier to make and they don't destabilize the vehicle as easily. Using the state arrays shown in figures 14 and 16, I was able to map the points found in SolidWorks and create the trajectory of the vehicle. I found the times shown in the state diagram through trial and error. We can observe that in both 2.1 and 2.2 the vehicle makes the C-shape turn without crashing into a wall. The plots accompanied with each Simulink model illustrate the outputs and inputs. These plots provide useful insight into the components that make the trajectory of the robot possible. All in all, this was a difficult problem but by using SolidWorks, the Robotics System Toolbox, and Mobile Robotics Training Toolbox I was able to solve this problem and comprehend what was occurring in the vehicle motion.

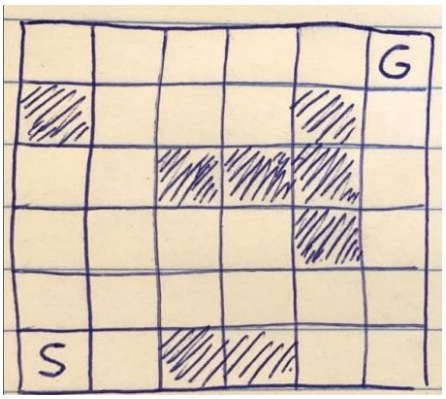
Simulink File

problem2_2slx.slx

Problem 3: Mobile Robot Navigation

In the map grid (below) the robot's start point is in the SW corner cell and the goal point is in the NE corner cell. Add any assumptions as you may need.

3.1 Use a Bug2 algorithm (of some kind) to navigate the robot. Diagonal crossing from cell to cell is allowed. The bug may be either left-turning only or right-turning only.



Solve manually and then verify the solution with MATLAB RTB.

We are tasked with finding a bug2 algorithm that is only left-turning or right-turning and diagonal crossing is allowed. In the following figure, I drew by hand the two trajectories this algorithm could possibly take:

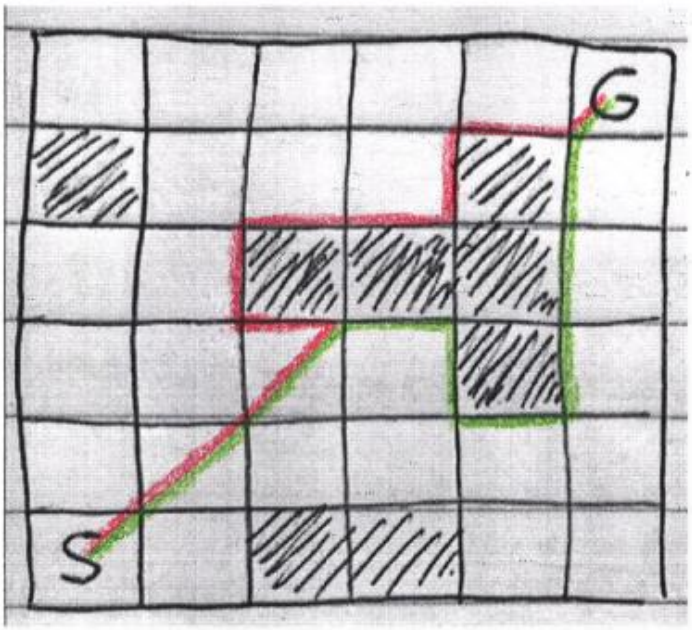


Figure 17 Possibe Paths of Bug2 Algorithm

The green line illustrates the right-turn-only bug2 algorithm and the red line illustrate the left-turn-only bug2 algorithm. After creating the MATLAB code, I obtained the following plot (MATLAB Code is shown at the end of problem 3):

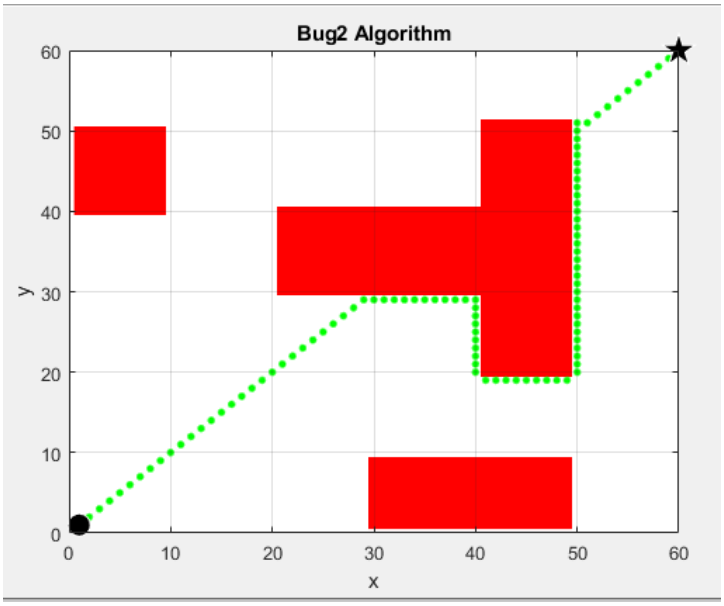
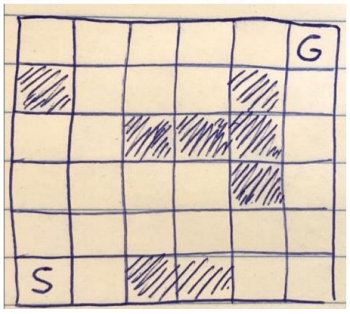


Figure 18 Bug2 Algorithm

The plot shown in figure 18 matches the green line shown in figure 17. For this problem, I had to modify the size of the map because the bug2 algorithm was doing an endless loop around the center obstacle. The reason for this is that if a cell is partially occupied then the program considers the whole cell occupied. Thus, I had to slightly modify the map to accommodate this issue.

3.2 Create a Distance Transform map and navigate the robot. Assume that the distance is measured using city blocks. No diagonal motion from cell to cell is allowed. Assume equal distance to cover a street or an avenue.



Solve manually, and then write a MATLAB code to verify your solution.

In this problem we are tasked with using the Distance Transform algorithm using city blocks and no diagonal motion; however, for this problem, I experienced an issue, where the Distance Transform algorithm did not allow me to edit the distance map. The Robotics Toolbox Manual

says the distance map can only be viewed but not edited for the dx functions. Thus, below I generated two codes where one uses the Distance Transform algorithm and the no diagonal motion requirement is neglected and the other uses the D* algorithm to modify the cost of the map to generate the desired distance map. I manually solved a few paths that the Distance Transform algorithm and the modified D* algorithm may take:

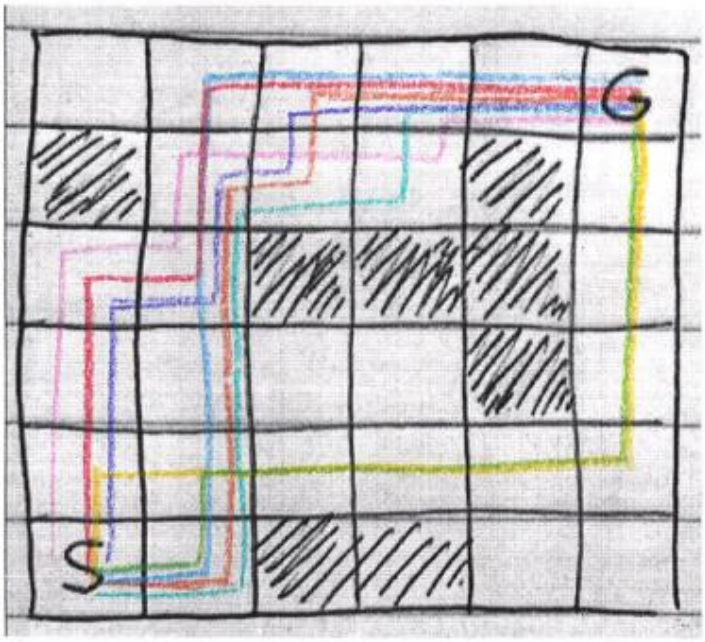


Figure 19 Some Possible Paths of a Distance Transform Map with no Diagonal Motion Allowed

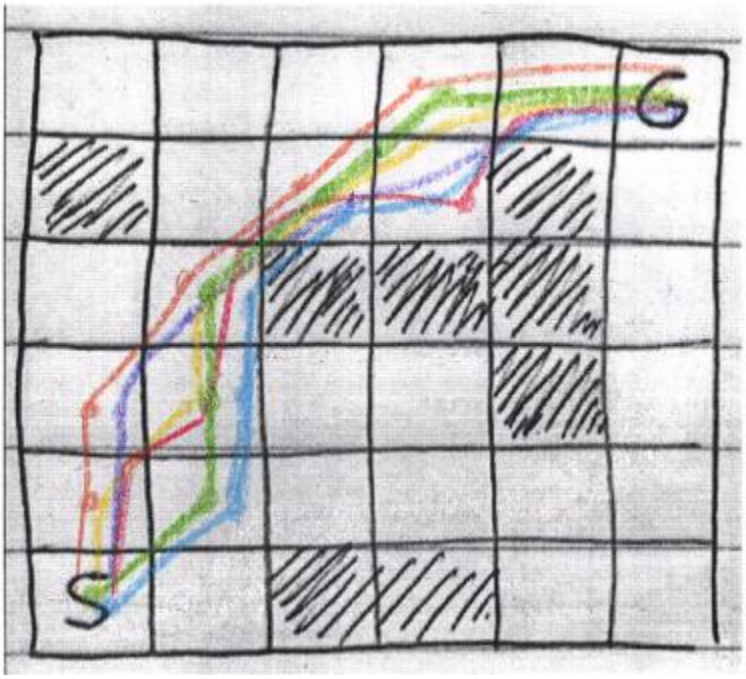


Figure 20 Possible Paths of dx Algorithm

After creating the MATLAB code, I obtained the following plot (MATLAB Code is shown at the end of problem 3):

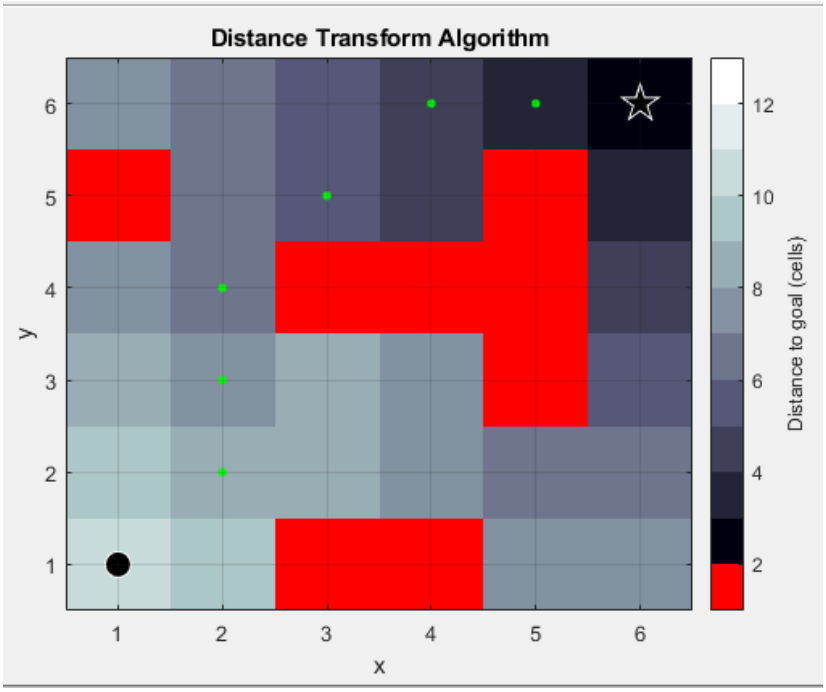


Figure 21 Distance Transform Algorithm

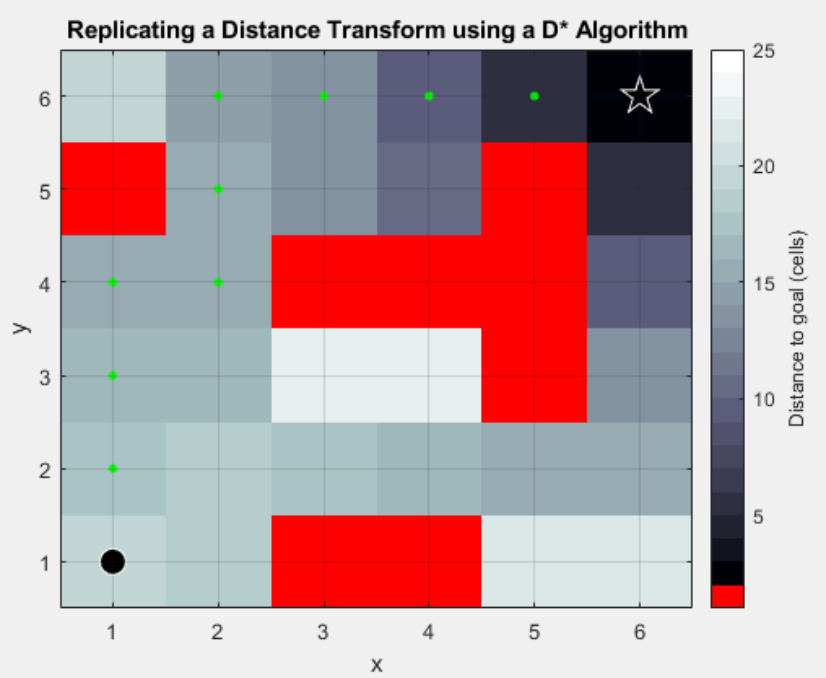
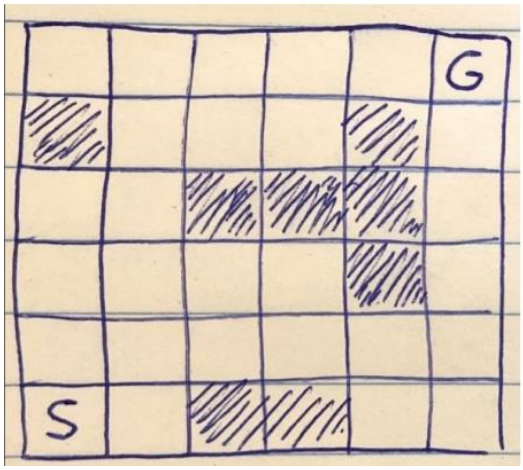


Figure 22 Attempt to Make Distance Transform Map with No Diagonal Motion Using D*

As we can observe in figures 21 and 22, the plots match some of the paths predicted by hand in figures 19 and 20.

3.3 The Cost Map in a D* navigation algorithm of this problem should be based on the road roughness. The SE region has very rough terrain. The NW region is mostly flooded with 1 [ft] deep puddles. Create a cost map and show the robot navigation, both manually and using MATLAB RTB.



In this problem, we are tasked with finding the D* algorithm for a map with environmental hazards. There is very rough terrain in the Southeast region and a 1-foot flood in the Northwest region. For this problem I assumed that the quadrant origin begins at the center of the map and that the regions are divided into sections of 9 blocks as shown below:

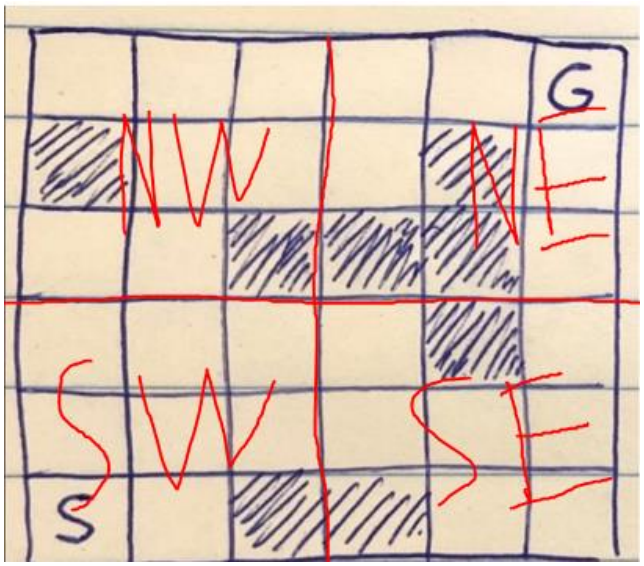


Figure 23 Quadrant Map of problem 3.3

By logic/common sense I assumed that it is easier for a grounded robot to navigate through very rough terrain than through a flood. Thus, for the cost map, I decided to give the vacant squares in the NW region a higher cost than the vacant squares in the SE region. Below I manually showed the two possible trajectories that the algorithm could take with the adjusted cost map:

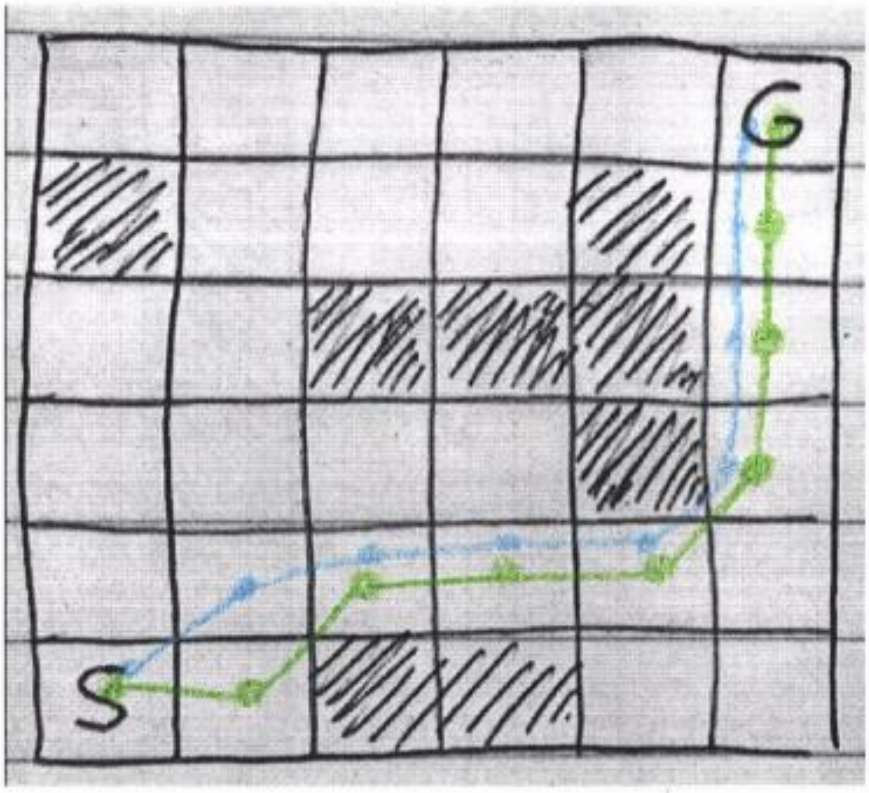


Figure 24 Possible Paths of D* Algorithm

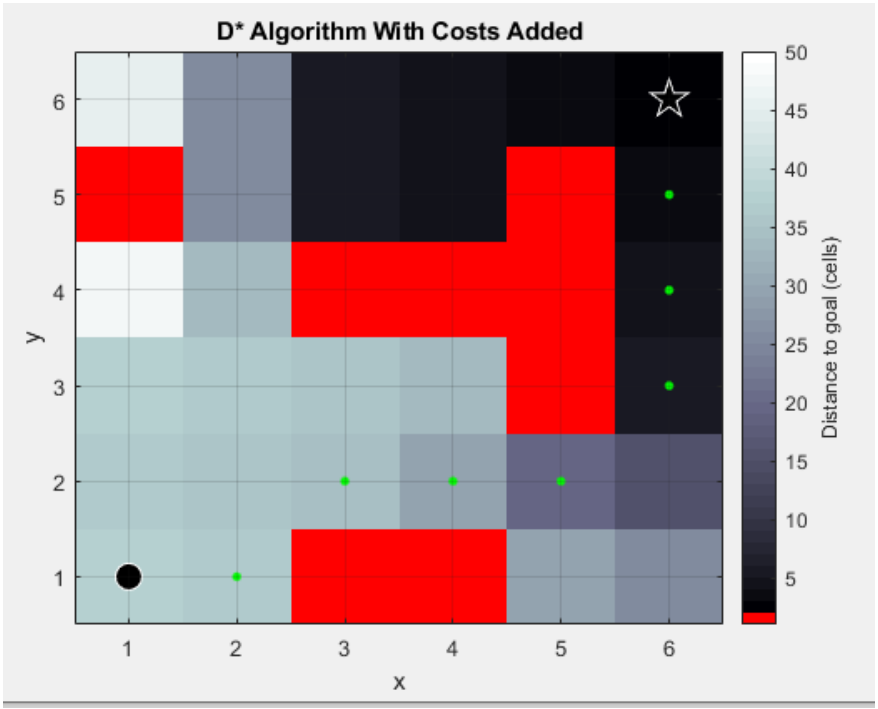


Figure 25 D* Algorithm with Costs Added

As we can observe in figure 25, the plot match one of the paths predicted by hand in figure 24.

MATLAB Code for Problem 3:

```
clear all;clc; close all;
%Jancarlos Gomez
%Z23521760
%Problem 3

map = [0,0,1,1,0,0; 0,0,0,0,0,0; 0,0,0,0,1,0; 0,0,1,1,1,0; 1,0,0,0,1,0;
0,0,0,0,0,0] % map used for 3.2 and 3.3
start = [1 1]; % starting point for 3.1-3.3
goal = [6 6]; % ending point for 3.2 and 3.3

%3.1
goal3_1 = [60 60]; %define the goal
map3_1 = zeros(60) % create an array with only zeros 60x60
map3_1(40:50,1:9) = 1 % create obstacles on map
map3_1(40:51,41:49) = 1 % create obstacles on map
map3_1(30:40,21:49) = 1 % create obstacles on map
map3_1(20:30,41:49) = 1 % create obstacles on map
map3_1(1:9,30:49) = 1 % create obstacles on map
bug = Bug2(map3_1); % create navigation object
bug.plot(); % plot map
p = bug.query(start, goal3_1,'animate'); % animate path
bug.plot(p); % plot animation
title('Bug2 Algorithm')
xlim([0 60]);
ylim([0 60]);
pause

%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 6]);
ylim([0 6]);
title('Distance Transform Algorithm')
pause

%3.2 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 6]);
ylim([0 6]);

%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]

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 6]);
ylim([0 6]);
title('Replicating a Distance Transform using a D* Algorithm')
pause

% %3.3
ds=Dstar(map); % create navigation object
c=ds.costmap(); % load costmap
ds.plan(goal); % plan path to goal
figure % summon new figure
ds.query(start,'animate') % animate sstart to goal
xlim([0 6]);
ylim([0 6]);
title('D* Algorithm Without Costs Added')
pause
%Modify cost to add obstacles
%Flooded Area NW
flood14=[1;4]
flood16=[1;6]
flood24=[2;4]
flood25=[2;5]
flood26=[2;6]
flood35=[3;5]
flood36=[3;6]

ds.modify_cost(flood14,20);
ds.modify_cost(flood16,20);
ds.modify_cost(flood24,20);
ds.modify_cost(flood25,20);
ds.modify_cost(flood26,20);
ds.modify_cost(flood35,20);
ds.modify_cost(flood36,20);

%Very Rough Terrain SE
rough42=[4;2]
rough43=[4;3]
rough51=[5;1]
rough52=[5;2]
rough61=[6;1]
rough62=[6;2]
rough63=[6;3]
```

```
ds.modify_cost(rough42,10);
ds.modify_cost(rough43,10);
ds.modify_cost(rough51,10);
ds.modify_cost(rough52,10);
ds.modify_cost(rough61,10);
ds.modify_cost(rough62,10);
ds.modify_cost(rough63,10);

ds.plan(); % update plan
p=ds.query(start) % animate start to goal
ds.plot(p) % plot it on the map
xlim([0 6]);
ylim([0 6]);
title('D* Algorithm With Costs Added')
```

MATLAB File

Problem3.m

Problem 4: Navigation of a User-Created Large Grid with Obstacles

4.1 Use MATLAB RTB "makemap" command and the RTB map editor to create some 80x80 (or something with the same order of magnitude) grid. Create some interesting obstacles inside the map.

First, I created the map and then saved the variable in a separate file to not have to keep recreating the map every time the program ran.

```
1 —    clear;clc; close all;
2      %Jancarlos Gomez
3      %Z23521760
4      %Problem 4
5
6 —    map = makemap(80)
7
8
```

Figure 26 MATLAB Code to make map

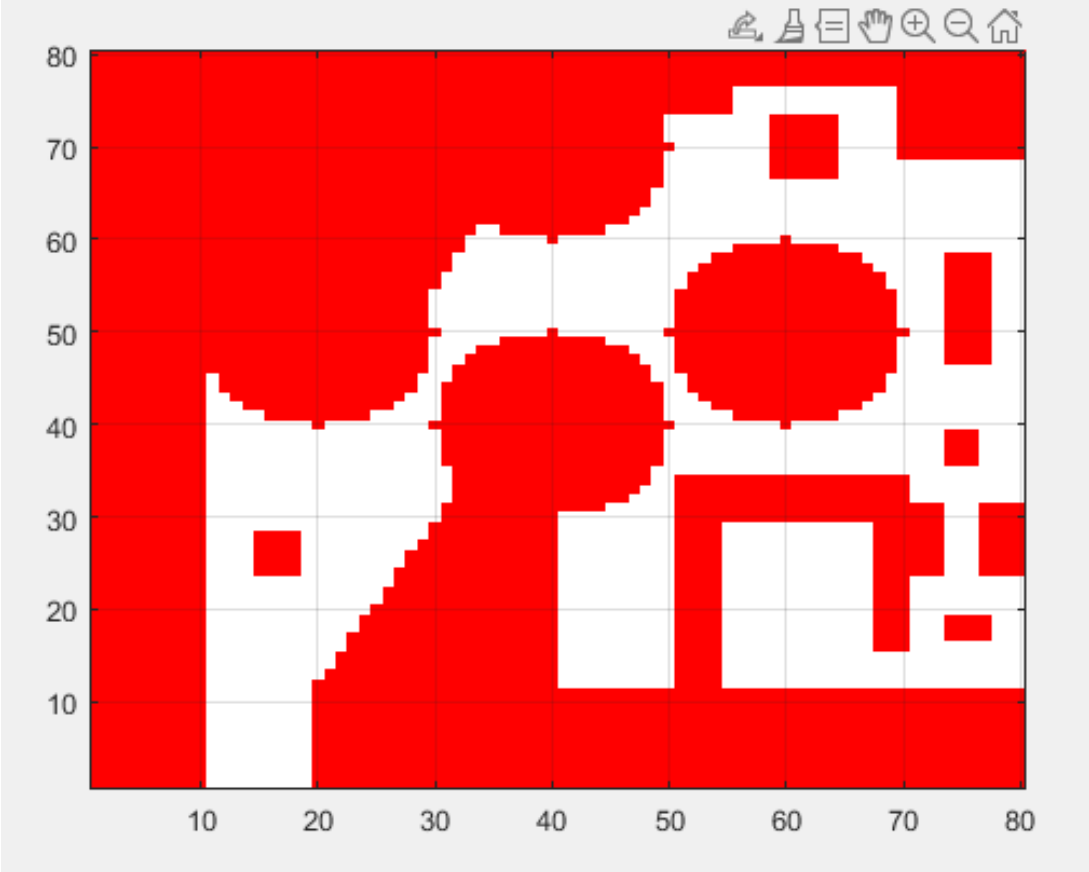


Figure 27 Map Created

4.2 Select a pair of start and goal cells and demonstrate (using MATLAB RTB navigation tools) a D* navigation algorithm, with some interesting cost of your choice.

The following start and goal cells were chosen:

```
start = [15 1]; % define the starting point
stop = [60 20]; % define the ending point
```

In the figure below we can observe D* algorithm before modifying the costs:

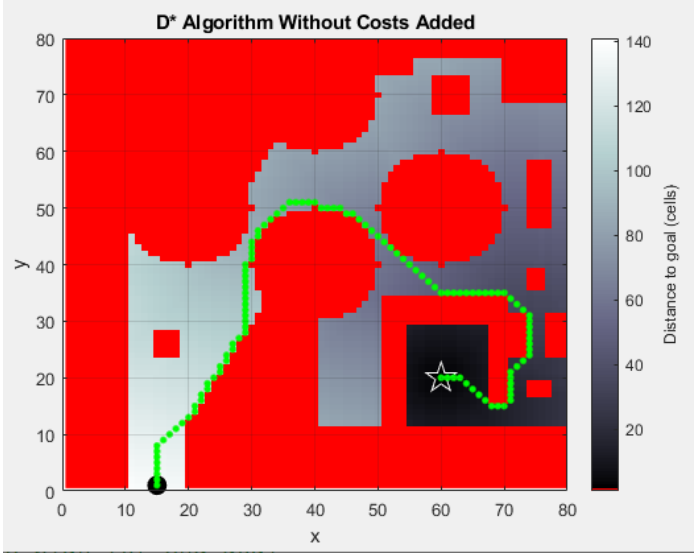


Figure 28 D* Algorithm Before Adding Costs

In the figure below we can observe D* algorithm after modifying the costs:

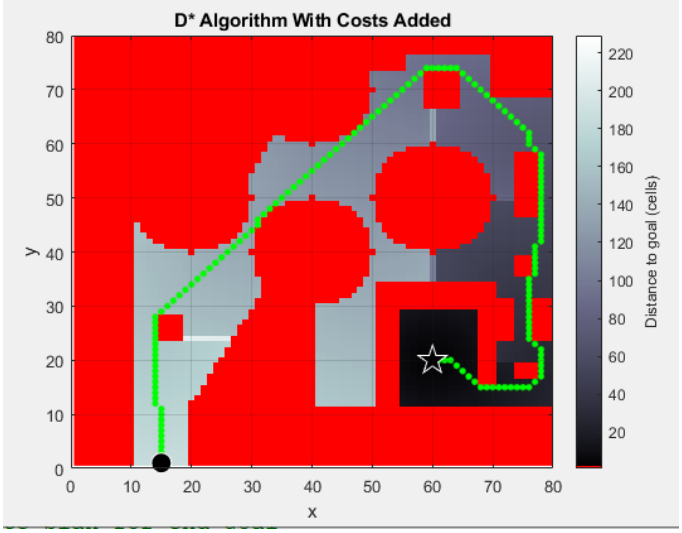


Figure 29 D* Algorithm After Adding Costs

In figure 29, we can observe that the path taken is longer than in figure 28 due to the high cost associated with taking shortcuts.

4.3 As a preparation for a Roadmap Navigation algorithm, create the Voronoi diagram for the free space of your grid.

In figure 30, we can observe Voronoi Diagram for the free space on the map:

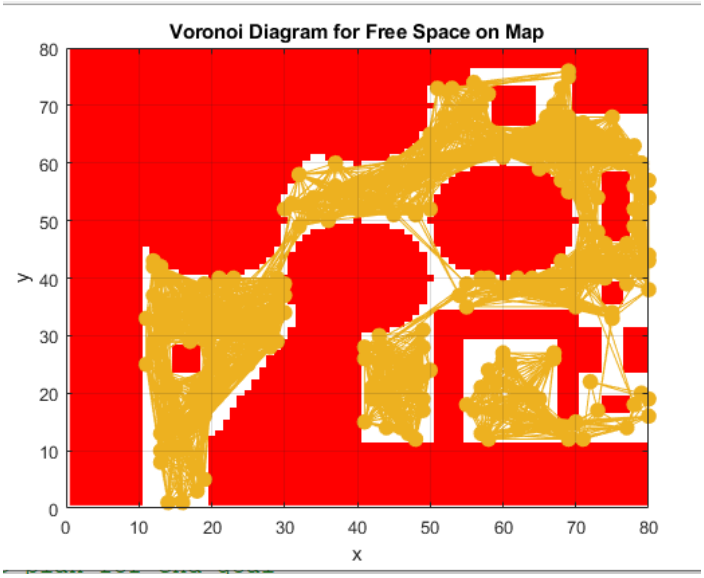


Figure 30 Voronoi Diagram for Free Space on the Map

4.4 Demonstrate a successful Probabilistic Roadmap Navigation from some start and end cells of your choice. Then change the start and end cells and run the navigation again, without changing the roadmap that you used for the previous set of start and end points.

Using the same start and goal cells as 4.1 and the same roadmap as in 4.3 the following Probabilistic Roadmap Navigation was generated:

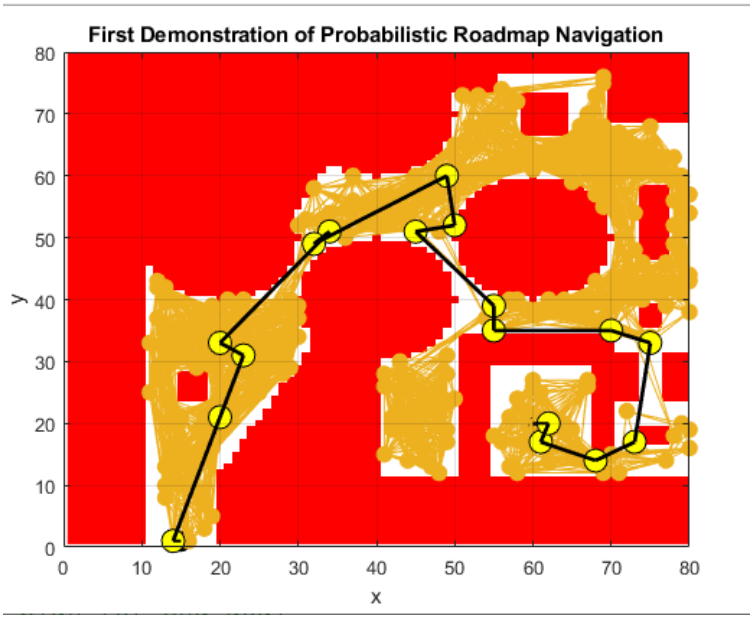


Figure 31 First Probabilistic Roadmap Navigation

Using the start and goal cells shown below and the roadmap found in 4.3 the following Probabilistic Roadmap Navigation was found:

```
start2 = [45 15];
stop2 = [62 75];
```

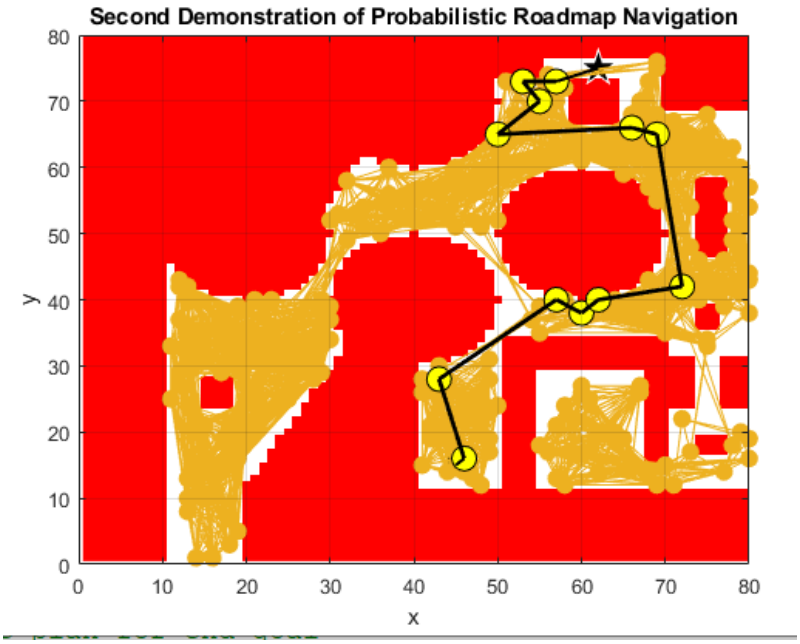


Figure 32 Second Probabilistic Roadmap Navigation

MATLAB Code for Problem 4:

```
clear all;clc; close all;
%Jancarlos Gomez
%Z23521760
%Problem 4
%4.1
%map = makemap(80)
%import map vector that was previously created to avoid having to remake
%the map
load map
%4.2
start = [15 1]; % define the starting point
stop = [60 20]; % define the ending point
ds=Dstar(map); % create navigation object
c=ds.costmap(); % load costmap
ds.plan(stop); % create plan for end goal
figure
ds.query(start,'animate') % animate path from starting location
xlim([0 80]);
ylim([0 80]);
title('D* Algorithm Without Costs Added')
pause
%Modify cost to add obstacles
%obstacle 0
```

```
roadblock0a=[19;24]
roadblock0b=[20;24]
roadblock0c=[21;24]
roadblock0d=[22;24]
roadblock0e=[23;24]
roadblock0f=[24;24]
roadblock0g=[25;24]
roadblock0h=[26;24]
roadblock0i=[27;24]
roadblock0j=[28;24]

ds.modify_cost(roadblock0a,100);
ds.modify_cost(roadblock0b,100);
ds.modify_cost(roadblock0c,100);
ds.modify_cost(roadblock0d,100);
ds.modify_cost(roadblock0e,100);
ds.modify_cost(roadblock0f,100);
ds.modify_cost(roadblock0g,100);
ds.modify_cost(roadblock0h,100);
ds.modify_cost(roadblock0i,100);
ds.modify_cost(roadblock0j,100);

%obstacle 1
roadblock1a=[60;34]
roadblock1b=[60;35]
roadblock1c=[60;36]
roadblock1d=[60;37]
roadblock1e=[60;38]
roadblock1f=[60;39]
roadblock1g=[60;40]

ds.modify_cost(roadblock1a,100);
ds.modify_cost(roadblock1b,100);
ds.modify_cost(roadblock1c,100);
ds.modify_cost(roadblock1d,100);
ds.modify_cost(roadblock1e,100);
ds.modify_cost(roadblock1f,100);
ds.modify_cost(roadblock1g,100);
%obstacle 2
roadblock2a=[60;60]
roadblock2b=[60;61]
roadblock2c=[60;62]
roadblock2d=[60;63]
roadblock2e=[60;64]
roadblock2f=[60;65]
roadblock2g=[60;66]

ds.modify_cost(roadblock2a,100);
ds.modify_cost(roadblock2b,100);
ds.modify_cost(roadblock2c,100);
ds.modify_cost(roadblock2d,100);
ds.modify_cost(roadblock2e,100);
ds.modify_cost(roadblock2f,100);
ds.modify_cost(roadblock2g,100);
%obstacle 3
```

```
roadblock3a=[70;50]
roadblock3b=[71;50]
roadblock3c=[72;50]
roadblock3d=[73;50]
roadblock3e=[74;50]

ds.modify_cost(roadblock3a,50);
ds.modify_cost(roadblock3b,50);
ds.modify_cost(roadblock3c,50);
ds.modify_cost(roadblock3d,50);
ds.modify_cost(roadblock3e,50);
%obstacle 4
roadblock4a=[70;19]
roadblock4b=[71;19]
roadblock4c=[72;19]
roadblock4d=[73;19]
roadblock4e=[74;19]

ds.modify_cost(roadblock4a,50);
ds.modify_cost(roadblock4b,50);
ds.modify_cost(roadblock4c,50);
ds.modify_cost(roadblock4d,50);
ds.modify_cost(roadblock4e,50);

ds.plan(); % update plan
p=ds.query(start) % generate path from start
ds.plot(p) % plot path
xlim([0 80]);
ylim([0 80]);
title('D* Algorithm With Costs Added')
pause

%4.3
%create Voronoi Diagram
figure
prm = PRM(map);
prm.plan('npoints',250);
prm.plot();
xlim([0 80]);
ylim([0 80]);
title('Voronoi Diagram for Free Space on Map')
pause

%4.4
%Demonstrate Probabilistic Roadmap Navigation Twice
%First Demonstration
prm.query(start,stop);
prm.plot();
xlim([0 80]);
ylim([0 80]);
title('First Demonstration of Probabilistic Roadmap Navigation')
pause
%Second Demonstration
start2 = [45 15];
stop2 = [62 75];
```

```
figure
prm.query(start2,stop2);
prm.plot();
xlim([0 80]);
ylim([0 80]);
title('Second Demonstration of Probabilistic Roadmap Navigation')
```

MATLAB Files

map.mat

Problem4.m

References

Peter Corke – Robotics Toolbox for MATLAB Release 10