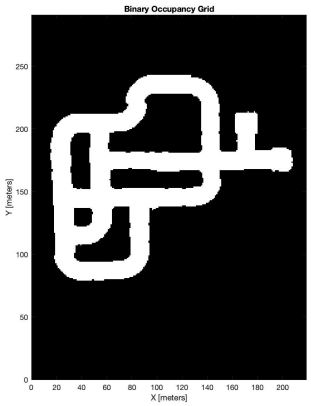
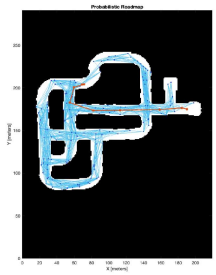
Lab 3 - Path Planning

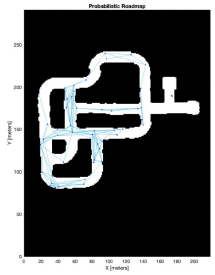
Part 2 - Where should I go next?

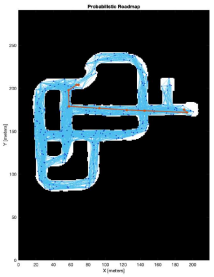
Similar to planning an adventure in an amusement park, the first item we will need for path planning is a map. In this activity, you will load a JPG image of the road into MATLAB to generate a 2D binary occupancy grid. Imagine covering the jpeg image with a transparency grip sheet, then coloring all the grids with **obstacles** (e.g. building, sidewalks, etc.) **in black** and leaving the **free space** (the road the robot allows to drive on) **in white**. That is the idea of a binary occupancy grid!



We then generate a probabilistic roadmap (PRM), which is a network graph of possible paths in a given map based on free and occupied spaces. We can tune the number of nodes within the network. Increasing the number of nodes will improve the resolution but will also increase the computational time. However, an insufficient number of nodes might result in no path being found. This can be seen in the figures below.







PRM is a network connecting all the nodes (the dark blue dots) in a map. The image on the left describes a PRM with a sufficient number of nodes, and therefore, a path between two locations is found and labeled in red. The image in the middle describes a PRM with an insufficient number of nodes, and therefore, results in a path not being found. The image on the right describes a PRM with a similar path as the image on the left. A map with an increased number of nodes is obtained, but computation time is increased.

The lab is modified from the following MATLAB examples:

https://www.mathworks.com/help/robotics/ug/probabilistic-roadmaps-prm.html, *https://www.mathworks.com/help/robotics/ug/path-following-for-differential-drive-robot.html*

Step 1: Set the scales

In this step, we define the correlation between pixels on an image to the real-world scale. Therefore, the question to be answered is how many pixels represent 1 cm?

Estimate the number of pixels that describes the width of the road from the image (Use any graphic editor, e.g. MS paint, mac Preview, etc.). Then, measure the physical dimension of the road at the same location. Input those values into the following code.

```
scalePixel = 3.435 % Replace XXX with your measurement from the image in Pixels.
```

```
scalePixel = 3.4350
```

```
scaleCm = 1 % Replace XXX with your measurement from the road in CM.
```

```
scaleCm = 1
```

```

PixelPerCm= round(scaleCm/scalePixel, 3);
```

Step 2: Load an image and generate a binary occupancy grid.

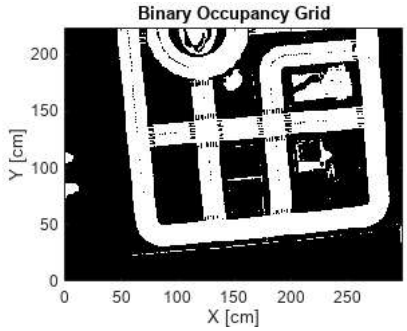
Load an image using the command 'imread'. Replace the filename with your image. **You can download the image from Canvas. Download the correct Calmap image with respect to the location of the CameraBot your team has been working with.**

Then, change the level value for converting a color image into black and white. [BW = im2bw(I, level)]. See the comment in the code for more detail.

```
image=imread('C:\Users\fhhoo\Downloads\Calmap_Door.jpg'); % Replace the filename if needed.
OrigionalImageSize = size(image);
```

```
image=imresize(image,PixelPerCm);

% Display input image
imshow(image)
```

```
% Convert image to black and white.
BW = im2bw(image,0.23); % Adjust the level value until the road is clearly defined in the binary occupancy grid.
                        % im2bw(I,level) converts the grayscale image I to binary image BW, by replacing all pixels in the input image wit
                        % luminance greater than level with the value 1 (white) and replacing all other pixels with the value 0 (black).

% Generate a biniary occupancy grid.
map = binaryOccupancyMap(BW);

% Display the map
show(map)
xlabel('X [cm]')
ylabel('Y [cm]')
```

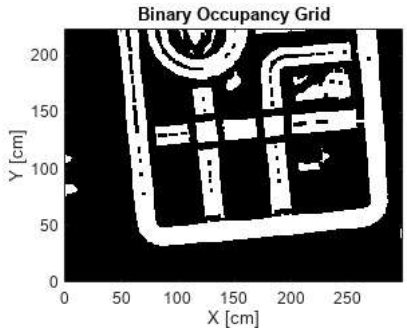


Step 3: Inflate the Map (You will not use the inflated map in the lab, but we would like you to learn this useful function.)

In this simulation, the differential drive robot is simulated as a point. However, an actual robot has a physical dimension. Inflating the occupancy map to create a thicker road boundary can compensate for the robot dimensions and prevent it from crashing into objects on the boundaries.

[inflate(map3D,radius)] inflates each occupied position of the specified in the input occupancyMap3D object by the radius specified in meters.

```
mapInflated = copy(map);
inflate(mapInflated, 0.1);
show(mapInflated);
xlabel('X [cm]')
ylabel('Y [cm]')
```



Step 4: Calculate the shortest distance from the current location to the target location.

Change the current location, target endpoint and heading angle based on your situation. You have obtained the current location of the Jetbot in the image captured by CameraBot using the April Tag. Select an endpoint and estimate the endpoint coordinate using software such as MS Paint. You can choose to use the units of cm or pixel, but you have to enter it correctly in the following code and make necessary edits.

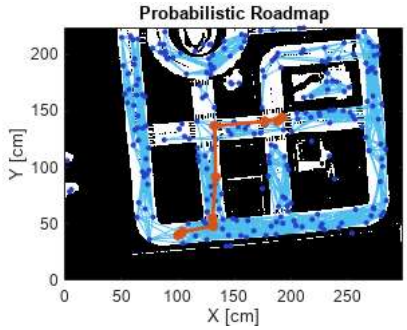
Note: JuypterLab, Matlab, and different graphic editors define the origin of an image differently. Check where the location is (0,0)!

Tune the number of nodes and connection distance of the PRM. A red path will be generated and will be highlighted as the found path. Read the comments in the code!

```
% Define current location, target endpoint and heading angle.
% Input current location in pixels as obtained from the Jupyter notebook.
currentLocation = [645.744116151798 485.1642619778119]; % Replace XXX with your AprilTag result, enter in pixels
Robot_heading_angle = 2.5159753940048417 ; % Replace XXX with your AprilTag result, enter in radian (Pointing along X: 0,  along Y: pi/2,


% Select and Estimate the endpoint location using the binary occupancy
% grid. Replace the XXs and input the value in cm or pixel.
endLocation = [100 40];

% Ensure the image alignment from image align with Matlab notation
% Add codes here if needed.

% Useful information - check where is the origin in the BOM. Thus, where is x = 0 and y = 0.
% When you define the endLocation, did you use the same notation?
% If not, consider using the OrigionalImageSize to convert the notations.
% endLocation has to define in Matlab notation. See the BOM.
% Beware of UNITS!


% Convert the unit of current location and endLocation (if needed) from pixel to cm
currentLocation = PixelPerCm .* currentLocation;
% endLocation =  PixelPerCm .* endLocation; % !!! Comment this line of code if you entered your endLocation in cm. !!!

% Use the BOM to generate a PRM, define the number of nodes and distance between two nodes.
% Tune the nodes and distance values, and observe how they affect your result.
prm = mobileRobotPRM(map);
prm.NumNodes = 250;
prm.ConnectionDistance = 50;

% Find a path from the current location to the target endpoint within the PRM.
path = findpath(prm, currentLocation, endLocation)
```

```
path = 9×2
  187.9115  141.1828
  192.0972  144.3873
  176.5740  140.7453
  132.4200  136.9226
  133.2539   91.4463
  130.4885   55.2877
  131.1350   48.2608
  103.6381   42.9602
  100.0000   40.0000
```

```
% Show the result
show(prm);
xlabel('X [cm]')
ylabel('Y [cm]')
```




```
% Check your result.
% 1. Are the start and end points correctly indicated on your map?
% If not, check your program. It's probably unit conversion, point identification, or alignment issues.
% 2. Does the red line indicate the shortest path?
```

Step 5: Use the following space to design an algorithm to return the decision at an intersection.

(Straight, Left Turn, or Right Turn)

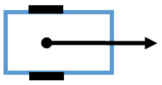
Why are we doing this? Assuming your Jetbot follows the yellow dotted line from the starting point along the path found in the previous step. The robot stops at the first intersection since no yellow line is detected. The robot needs to receive a command for turning or going straight. Assuming after it stops, you update your current location and heading angle using the SSH, and Apriltag techniques learned in this lab. Now, you need to develop an algorithm to command the Jetbot to go

straight or turn using the kinematics technique in Lab 1. After it decides on the intersection and moves, the jetbot will detect the yellow line. The jetbot will then follow the line again until it reaches the next intersection or the goal.

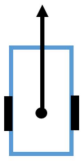
Example:

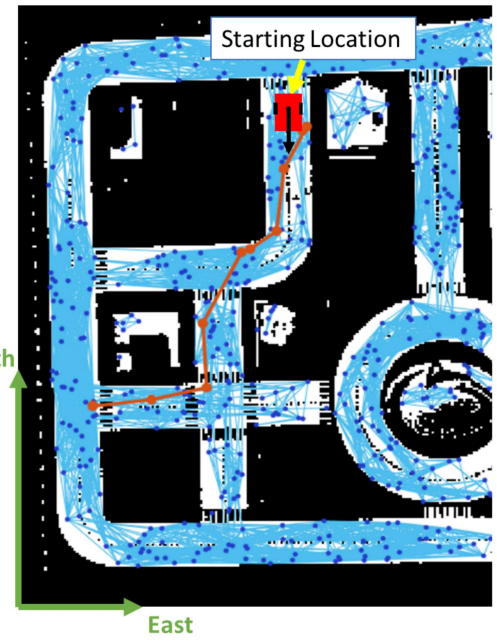
Assume this is the defined shortest path:

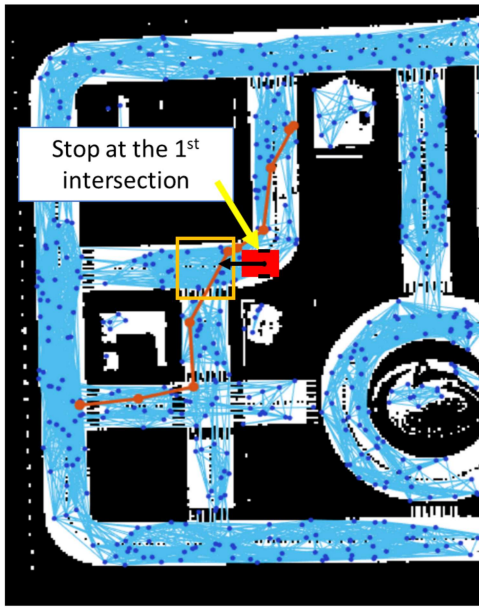
Heading angle = 0



Heading angle = $\frac{\pi}{2}$







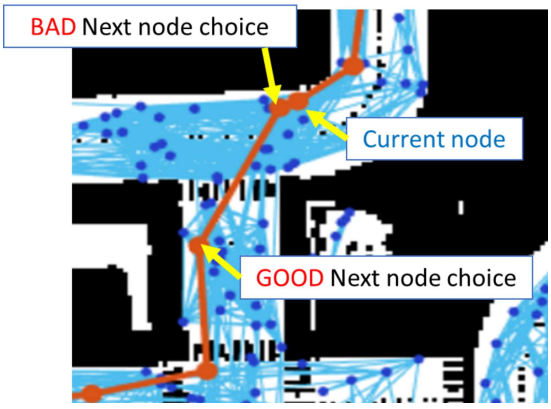
In this section, you will develop an algorithm to command the robot to turn or go straight at an intersection.

Step 5a: First, define your current location and heading angle. (See the comment in the provided code below for more detail.)

Step 5b: Then, identify the next node.

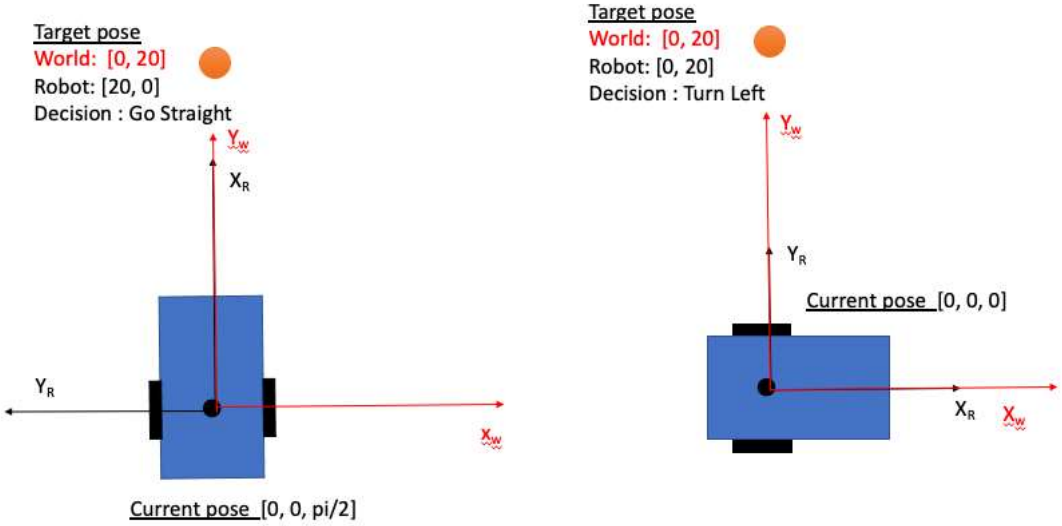
Which node is the appropriate next node? All the x and y coordinates of the found path (the orange line) are stored under an array named path. Depending on your choice of the number of nodes, the node immediately following the current position (the current node) may or may not be an ideal next node. For example, our current position is (20.5 cm, 101.5 cm), and the immediate next node is (19.2 cm, 98.0 cm). You can tell the immediate next node is not an appropriate choice since it is too close to the current node and does not provide valuable information to determine if the robot should go straight or turn at the interaction, as illustrated in the image belowimage below.

[Code for this part is provided for tuning.]



Step 5c: Once you identify the next node, we have to convert it from the world coordinate to the robot's local coordinate. For example, the current position is (0,0), and the next node is (0, 20). Should the robot go straight? This also depends on the robot's heading angle. If it was initially facing north (heading angle = pi/2), the decision is correct. However, if the robot was facing east (heading angle = 0), the decision should be left turn instead. Therefore, we should transform the next node from world coordinate to robot local coordinate. See the lecture notes if more information is needed.

[Codes NOT provided, students team's assignment.]



Step 5d: Lastly, define conditions to make a decision. Student teams need to write their own code inside the provided if condition: *"if (~isempty(ycompare)&& ~isempty(xcompare))"*.

Your code must display one of three decisions - "Straight", "Turn left", or "Turn Right"- which represents the robot's decision at an intersection. The decision should be appropriate for various circumstances. Therefore, if the robot encounters a different intersection or changes its heading angle, the displayed decision must still be correct for the particular situation.

Hints: Teams can consider how much the robot has to rotate from the current location to the next node to make a decision. You can use the command disp('straight') to output the decision.

[Codes NOT provided, students team's assignment.]

Run the following codes with the **Run Section** options so your found path won't update every time you run the code.


```
%%%% Run this section using the Run Section option %%%

%% Step 5a %%
% Define the current location
% Begin from the starting point, follow the path you found in step 4,
% stop at the node closest to the 1st intersection, and manually select that node as your current location.
% Checked the double array named "path" under your Workspace.
% Which row is your selected node corresponding to?
% Entered the row number as the defined value for selected_node below.

selected_node = 5; % Replace X with the corresponding row number in path
currentLocation = [path(selected_node,1) path(selected_node,2)];

%Approximate the heading angle. Select between 0 and 2pi.
% Then enter the selected heading angle to the space provided below.
Robot_heading_angle = pi/2; % Replace XX by the defined heading angle in radian (0: along X,  along Y: pi/2, along -ve Y: 3*pi/2


%% Step 5b - Identify the next node. %%
% To learn more about this step,
% Please see the text description above under step 5b.

nextnode=[]; % This link of code defines an empty variable

%"xcompare" and "ycompare" compare the horizontal and vertical distance from the current location to the next nodes.
for i=selected_node:length(path)
    xcompare = abs(path(i,1)-currentLocation(1,1));
```

```
    ycompare = abs(path(i,2)-currentLocation(1,2));

    % Define a reasonable distance (in cm) to define the next node.
    % How many cm away from the current location in either x or y direction is a reasonable next node?
    % Replace XX in the code below with the selected distance (in cm)

    if (xcompare > 55) || (ycompare > 15) % Define XX as an acceptable distance
       nextnode = [path(i,1) path(i,2)];
       break
    end

end

%% Step 5c %%
%Transform the nextnode from global reference frame (world coordinate) to local reference frame (robot coordinate)
Robot_rotation =  Robot_heading_angle ;
Transform_matrix=[cos(Robot_heading_angle) -sin(Robot_heading_angle); sin(Robot_heading_angle) cos(Robot_heading_angle) ]
```

```
Transform_matrix = 2×2
   -1.0000   -0.0000
    0.0000   -1.0000
```

```
Transformed_nextnode = Transform_matrix *transpose(nextnode-currentLocation);
% Hints: Define the Transform matrix
% Then, Transformed_nextnode = Transform_matrix *transpose(nextnode-currentLocation)


% Step 5d %%
if (~isempty(ycompare)&& ~isempty(xcompare))
    if Robot_heading_angle==pi/2
        next_action=disp('straight');
    else if Robot_heading_angle>=0 && Robot_heading_angle<pi/2
            next_action=disp('right');
    else if Robot_heading_angle>pi/2 && Robot_heading_angle<=pi
            next_action=disp('left');
    else if Robot_heading_angle>pi && Robot_heading_angle<=(3*pi/2)
            next_action=disp('left');
    else
        next_action=disp('right')
end
    % Define conditions to make a decision.
    % Then output one of the three decisions: straight, Turn left, or Turn Right, using the disp syntax, e.g., disp('left')
    %
    % Discuss with your team how your team defines each output.
    % Does a turn signal imply a small angle rotation or a more significant rotation? What is a reasonable condition for a driver to recei
    % We suggest considering how much the robot needs to rotate from the current location to the next node to decide on what is a reasonat
    % There is no correct answer to this question, but teams must make a reasonable selection.
    % Explain your choice in the space provided in the Lab 3 word document.


end
```

Submission

1. Save your this .mlx file as a PDF file and submit it to Canvas.
2. Answer the questions on the word document.
3. Demonstrate your output to a TA. Your TA will ask you to change the current location condition to verify your algorithm is making correct decisions.