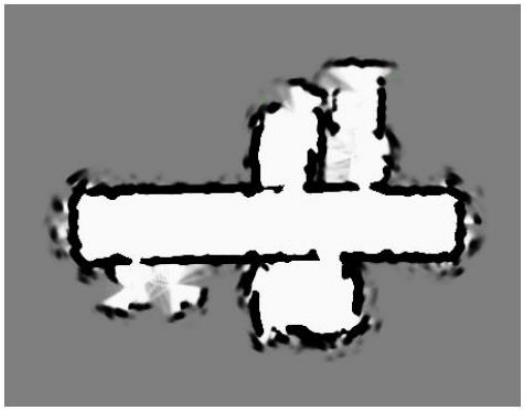
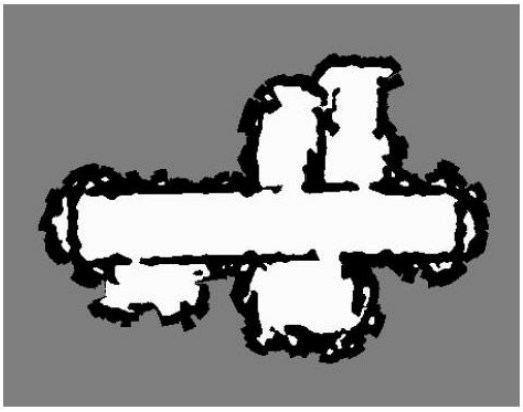
Homework 9

Instruction for Problems 2 and 3:





In this lecture, we have learned how to generate a map with known pose of the robot. Occupancy grid maps and reflection maps were introduced and discussed with their mathematic models. Occupancy grid maps are a popular approach to represent the environment of a mobile robot. The environment was divided into tiny grids, each cell is considered independently from all others and stores the probability that the corresponding area in the environment is occupied. Occupancy grid maps can be learned efficiently using a probabilistic approach. Reflection maps are an alternative representation. They store in each cell the probability that a beam is reflected by this cell. We provided a sensor model for computing the maximum likelihood of measurements and showed that the counting procedure underlying reflection maps yield the optimal map.

Simulation

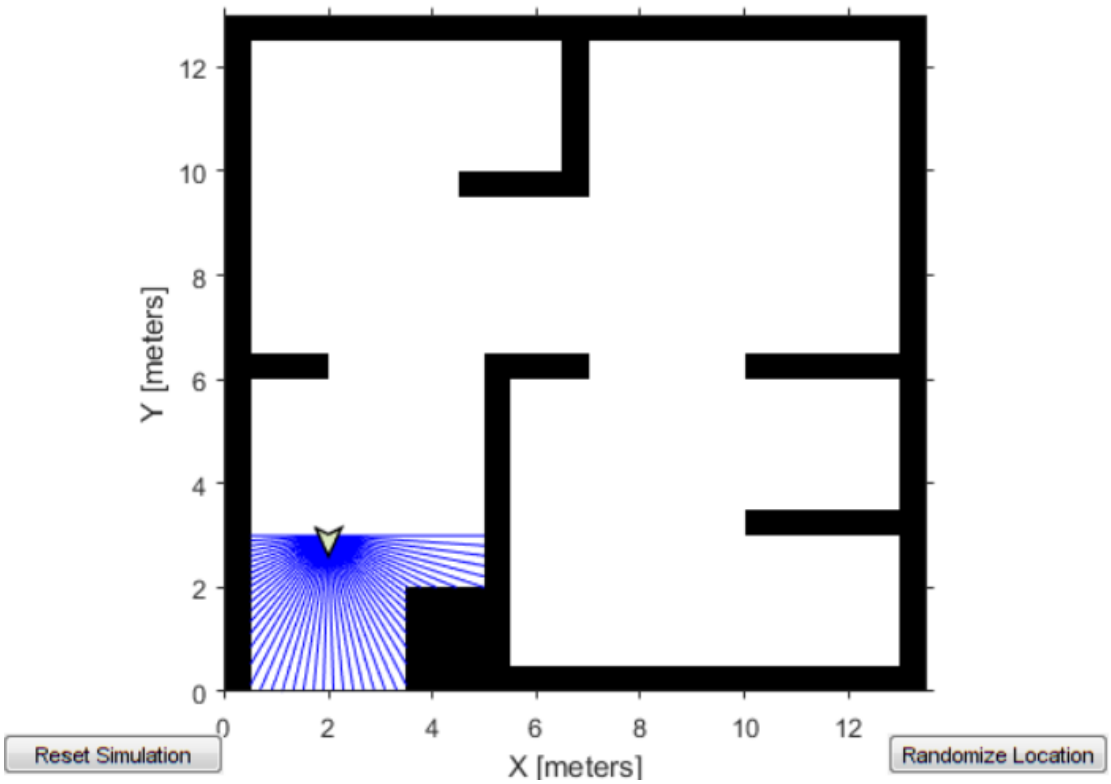
1. For implement our algorithms into robot mapping, first we should initialize a ROS system and build a given map. As we did it before, we use MATLAB to do this task conveniently. For driving the vehicle, we use the motion control model we have tested in former lectures. Now that the location and pose is given, the sensor model is not used and we can acquire this information from the system directly.

   Firstly, we start the ROS master in MATLAB.

```
rosinit
```

Initialize the robot simulator and assign an initial pose. The simulated robot is a two-wheeled differential drive robot we learned before with a laser range sensor.

```
sim = ExampleHelperRobotSimulator('simpleMap');
setRobotPose(sim, [2 3 -pi/2]);
% Enable ROS interface for the simulator. The simulator
% creates publishers and subscribers to send and receive data over ROS.
enableROSInterface(sim, true);
% Increase the laser sensor resolution in the simulator to
% facilitate map building.
sim.LaserSensor.NumReadings = 50;
```



2. Setup ROS Interface

Create ROS publishers and subscribers to communicate with the simulator. Create rossubscriber for receiving laser sensor data from the simulator. Create rospublisher to send velocity commands to the robot.

```
scanSub = rossubscriber('scan');
[velPub, velMsg] = rospublisher('/mobile_base/commands/velocity');
```

The MATLAB simulator publishes the position of the robot with respect to the map origin as a transformation on the topic /tf, which is used as the source for the

ground truth robot pose in this example. The simulator uses map and robot_base as frame names in this transformation.

Create a ROS transformation tree object using rostf function. For building a map, it is essential that robot pose and laser sensor reading correspond to the same time. The transformation tree allows you to find pose of the robot at the time the laser sensor reading was observed.

```
tftree = rostf;
```

```
% Pause for a second for the transformation tree object to finish
% initialization.
pause(1);
```

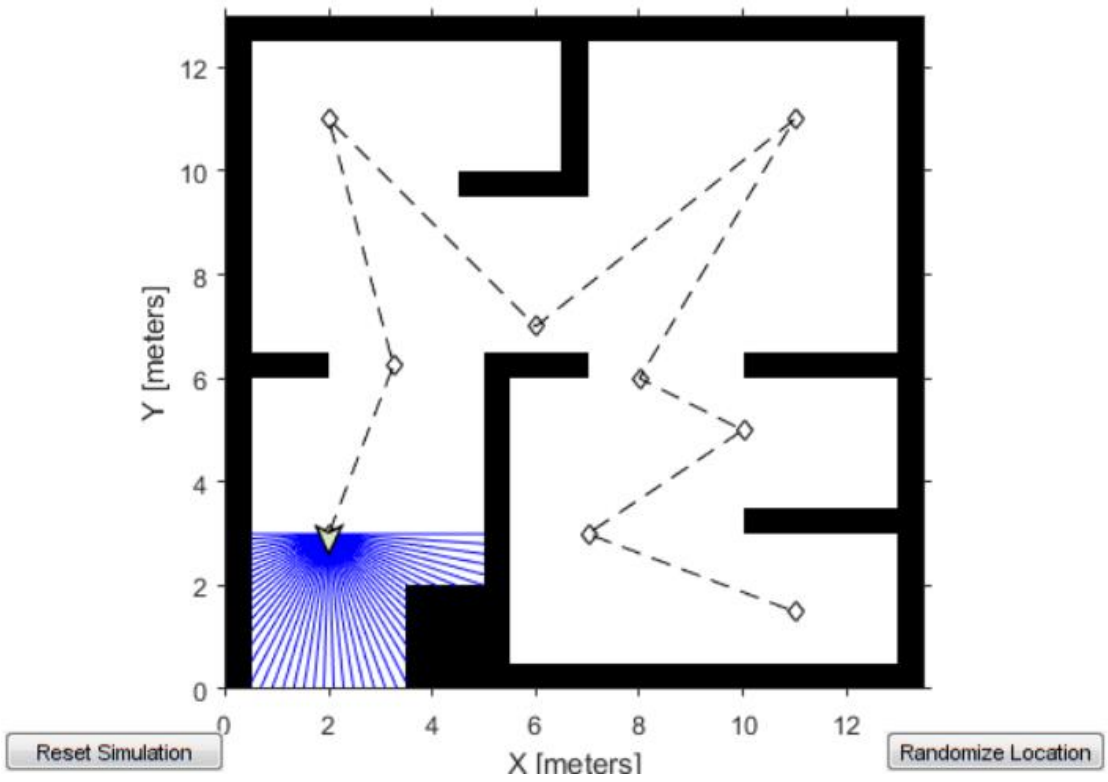
3. Create a Path Controller

The robot will have to drive around the entire map to collect laser sensor data and build a complete map. Assign a path with waypoints that cover the entire map.

```
path = [2, 3;3.25 6.25;2 11;6 7; 11 11;8 6; 10 5;7 3;11 1.5];
```

Visualize the path in the robot simulator

```
plot(path(:,1), path(:,2),'k--d'
```



Based on the path defined above and a robot motion model, you need a path following controller to drive the robot along the path. Create the path following

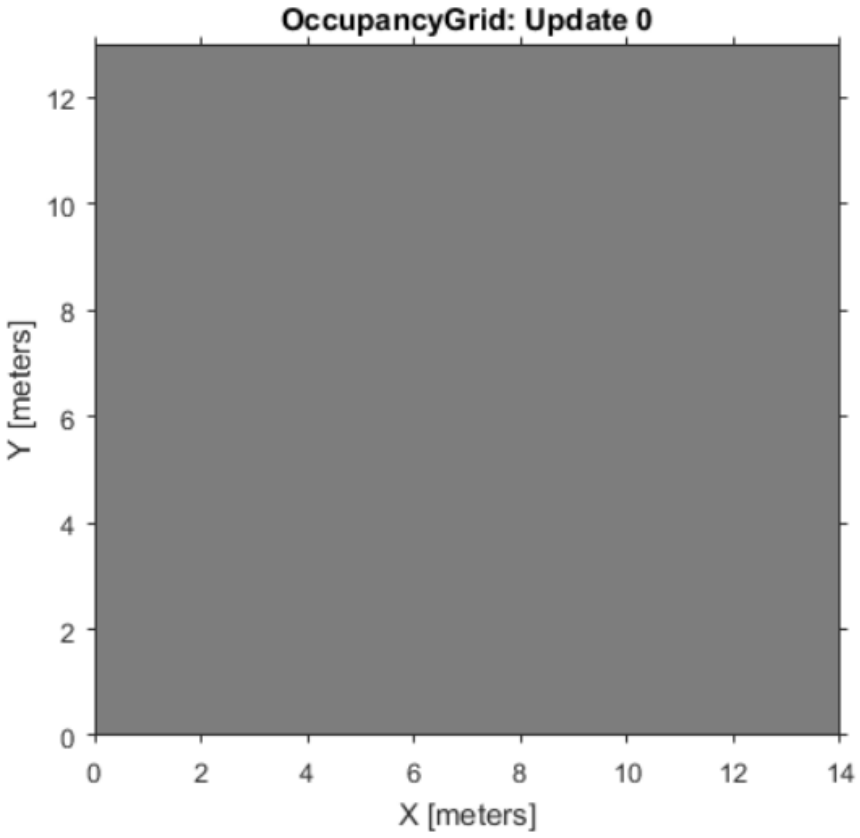
controller using the robotics.PurePursuit object. Set the controller rate to run at 10 Hz. Define a goal point and a goal radius to stop the robot at the end of the path.

```
controller = robotics.PurePursuit('Waypoints', path);
controller.DesiredLinearVelocity = 0.4;
controlRate = robotics.Rate(10);
goalRadius = 0.1;
robotCurrentLocation = path(1,:);
robotGoal = path(end,:);
distanceToGoal = norm(robotCurrentLocation - robotGoal);
```

4. Define an Empty Map

   Define a map with high resolution using robotics.OccupancyGrid to capture sensor readings. This creates a map with 14 m X 13 m size and a resolution of 20 cells per meter. Visualize the map in the figure window.

```
map = robotics.OccupancyGrid(14,13,20);
figureHandle = figure('Name', 'Map');
axesHandle = axes('Parent', figureHandle);
mapHandle = show(map, 'Parent', axesHandle);
title(axesHandle, 'OccupancyGrid: Update 0');
```



   Change the threshold values.

```
map.FreeThreshold = 0.5;
```

```
map.OccupiedThreshold = 0.5;
```

The following while loop will build the map of the environment while driving the robot. The following steps are performed:

First, you receive the laser scan data using the scanSub subscriber. Use the getTransform function with the time stamp on scan message to get the transformation between the map and robot_base frames at the time of the sensor reading.

Get the robot position and orientation from the transformation. The robot orientation is the Yaw rotation around the Z-axis of the robot. You can get the Yaw rotation by converting the quaternion to euler angles using quat2eul.

Pre-process laser scan data. The simulator returns NaN ranges for laser rays that do not hit any obstacle within the maximum range. Replace the NaN ranges by maximum range value.

Insert the laser scan observation using the insertRay method on the occupancy grid map.

Compute the linear and angular velocity commands using the controller object to drive the robot.

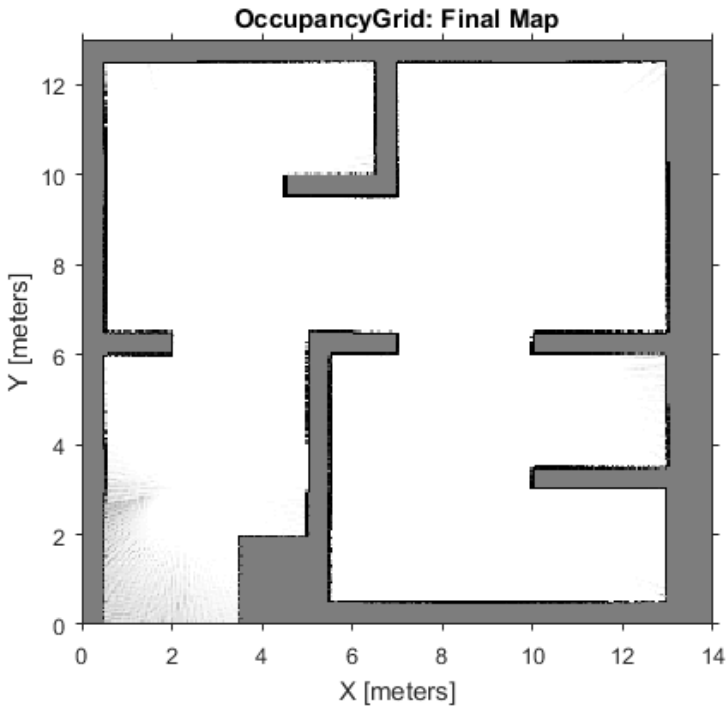
Visualize the map after every 50 updates.

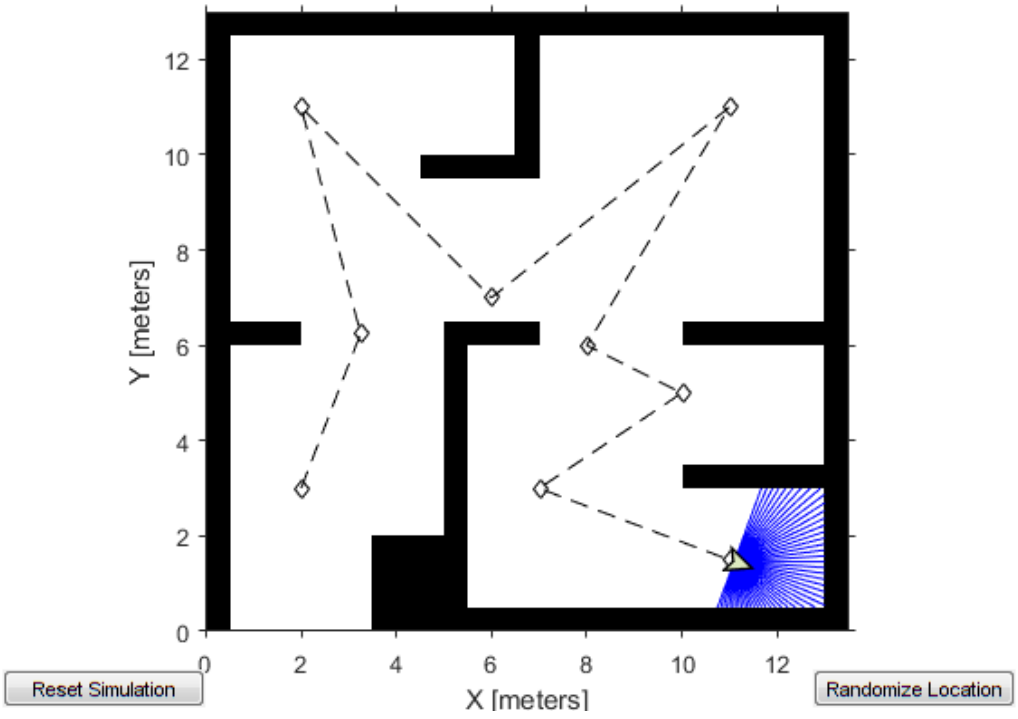
```
updateCounter = 1;
while( distanceToGoal > goalRadius )
    % Receive a new laser sensor reading
    scanMsg = receive(scanSub);

    % Get robot pose at the time of sensor reading
    pose = getTransform(tftree, 'map', 'robot_base', scanMsg.Header.Stamp,
'Timeout', 2);

    % Convert robot pose to 1x3 vector [x y yaw]
    position = [pose.Transform.Translation.X,
pose.Transform.Translation.Y];
    orientation =  quat2eul([pose.Transform.Rotation.W,
pose.Transform.Rotation.X, ...
        pose.Transform.Rotation.Y, pose.Transform.Rotation.Z], 'ZYX');
    robotPose = [position, orientation(1)];

    % Extract the laser scan
    scan = lidarScan(scanMsg);
    ranges = scan.Ranges;
    ranges(isnan(ranges)) = sim.LaserSensor.MaxRange;
    modScan = lidarScan(ranges, scan.Angles);
```

```
    % Insert the laser range observation in the map
    insertRay(map, robotPose, modScan, sim.LaserSensor.MaxRange);

    % Compute the linear and angular velocity of the robot and publish it
    % to drive the robot.
    [v, w] = controller(robotPose);
    velMsg.Linear.X = v;
    velMsg.Angular.Z = w;
    send(velPub, velMsg);

    % Visualize the map after every 50th update.
    if ~mod(updateCounter,50)
        mapHandle.CData = occupancyMatrix(map);
        title(axesHandle, ['OccupancyGrid: Update '
num2str(updateCounter)]);
    end

    % Update the counter and distance to goal
    updateCounter = updateCounter+1;
    distanceToGoal = norm(robotPose(1:2) - robotGoal);

    % Wait for control rate to ensure 10 Hz rate
    waitfor(controlRate);
end
```

Display the final map, which has incorporated all the sensor readings and shutdown the ROS.

```
show(map, 'Parent', axesHandle);
title(axesHandle, 'OccupancyGrid: Final Map');
```



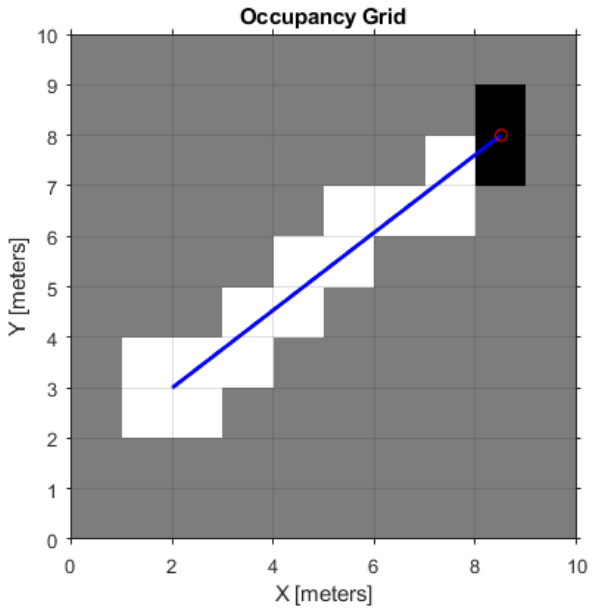


5. For reflection grid map, we should use an alternative algorithm to substitute the `insertRay(map, robotPose, modScan, sim.LaserSensor.MaxRange);`

   Before the while loop, we should firstly define two counters, alpha and beta, to count how many times a grid reflects a beam and passed through by a beam.

```
% initialize alpha and beta accumulators
alpha = zeros(map.GridSize);
beta = zeros(map.GridSize);
```

then use the raycast method to return the endpoints and midpoints grid indices of a beam, see as follow:



Every alpha-counter of endpoints increase 1 and beta-counter of midpoints increase 1. Note that if the beam range reaches the Maximum Range, the beta-counter of endpoints should increase rather than alpha-counter, because it indicates no reflection. Finally, refer the reflection probabilities of each grid to setOccupancy method to set the probabilities of grid map. The following code for reference.

```
% Insert the laser range observation in the map
    for n = 1:modScan.Count
        [endpoints, midpoints] = raycast(map, robotPose,
modScan.Ranges(n), ...
            modScan.Angles(n));
        if modScan.Ranges(n) == sim.LaserSensor.MaxRange
            beta = elementAdd(beta, endpoints);
        else
            alpha = elementAdd(alpha, endpoints);
        end
        beta = elementAdd(beta, midpoints);

        setij = [midpoints ; endpoints];
        [m, nc] = size(setij);
        occup = zeros(m, 1);
        for i = 1:m
            occup(i) = alpha(setij(i,1), setij(i,2)) / ...
                (alpha(setij(i,1), setij(i,2)) + beta(setij(i,1),
setij(i,2)));
        end
```

```
        setOccupancy(map, setij, occup, 'grid');
    end
```

where function elementAdd:

```
function matrix = elementAdd(matrix, addij)
%elementAdd Add 1 at specified elements of 'matrix'
%   Add 1 at each elements of 'matrix' specified by index vector 'addij'.

[m, n] = size(addij);
for i = 1:m
    matrix(addij(i,1), addij(i,2)) = matrix(addij(i,1), addij(i,2)) + 1;
end

end
```