Hw3_3

Simulation of GPS localization

clear;

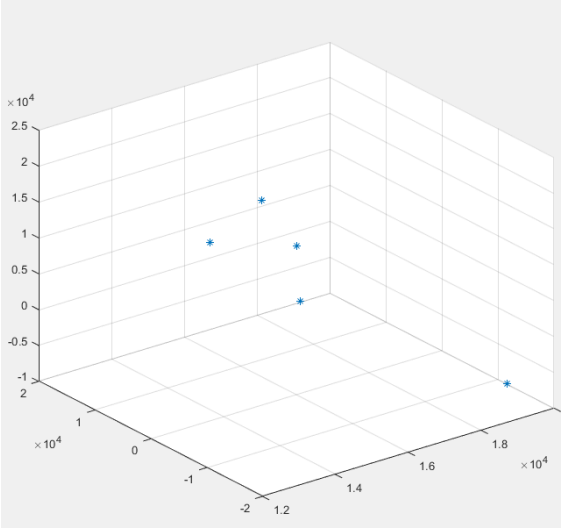
clc;

1, define the position of GPS satellite

SatellitePosition= _____;

2, use the scatter3 function to plot the satellite

scatter3()



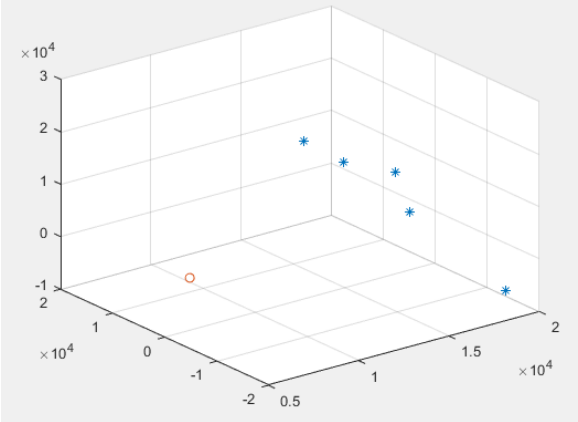
hold on;

3, define the position of user

UserPosition=[ ];

4, use the scatter3 function to plot the satellite

scatter3()



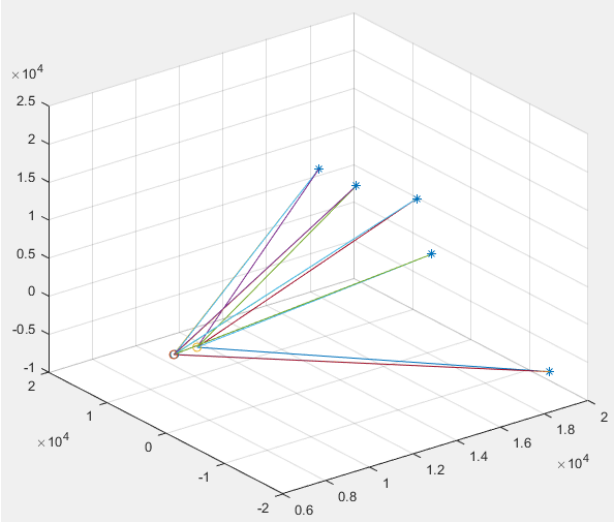
5, define your function to calculate the Pseudo Range

Prange=CalculatePseudoRange(SatellitePosition, UserPosition);

6, define your function to calculate the User position

[CalUserPosition, OK]=CalculateUserPosition2(SatellitePosition, Prange);

7, plot the satellite position and estimate user position



Hw3_4

1. Introduction

   In this lab, we will learn how to create a map of the environment using range sensor readings if the position of the robot is known at the time of sensor reading. For the purpose of this example, you will use a MATLAB based simulator to drive the robot, observe the range sensor readings and the robot poses.

   1.1 Range Sensor

   We have learned many kinds of sensors in class. A range sensor is able to detect the presence of nearby objects and calculate the distance between the object and the sensor. Ultrasonic range sensor, laser rangefinder and radar are all range sensors. Take laser rangefinder as an example. A laser rangefinder is a rangefinder that uses a laser beam to determine the distance to an object. The most common form of laser rangefinder operates on the time of flight principle by sending a laser pulse in a narrow beam towards the object and measuring the time taken by the pulse to be reflected off the target and returned to the sender.

   1.2 Mapping

   Mapping is the creation of maps, a graphic symbolic representation of the significant features of a part of the environment. There are many ways to represent the map and to create the map, which will be discussed in the later lecture. In this lab, In this lab, we create an occupancy grid map.

2. Simulation

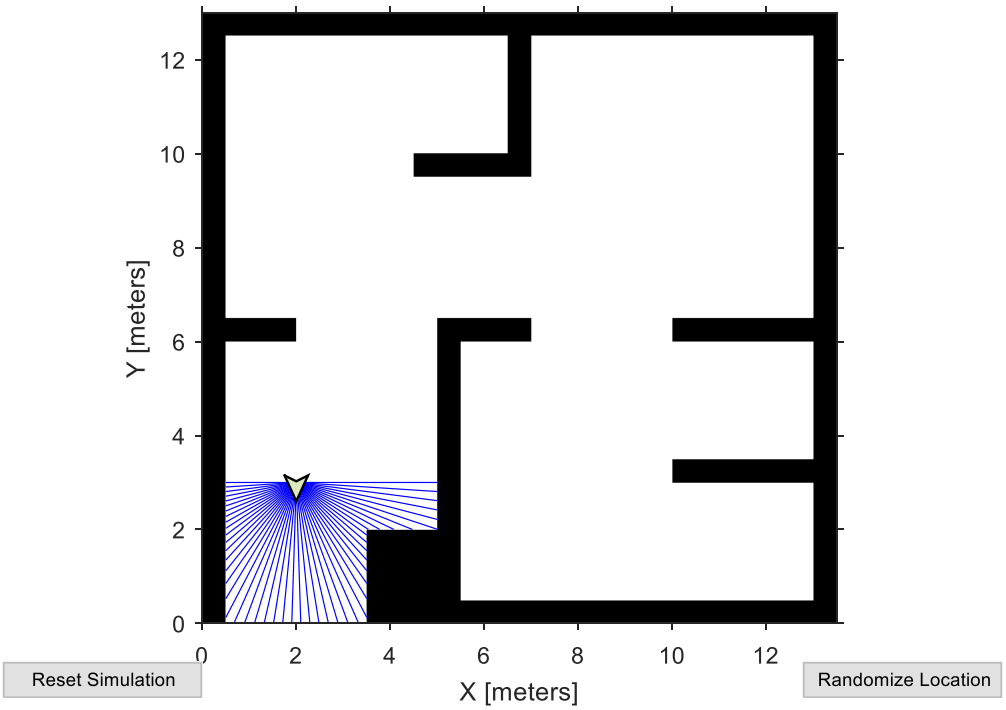
   2.1 Initialize the Robot Simulator

   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 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;
```



2.2 Setup ROS Interface

Create ROS publishers and subscribers to communicate with the simulator. Create `rossubscriber` for receiving laser sensor data from the simulator.

```
scanSub = rossubscriber('scan');
```

For building the map, the robot will drive around the map and collect sensor data. Create `rospublisher` to send velocity commands to the robot.

```
[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);
```

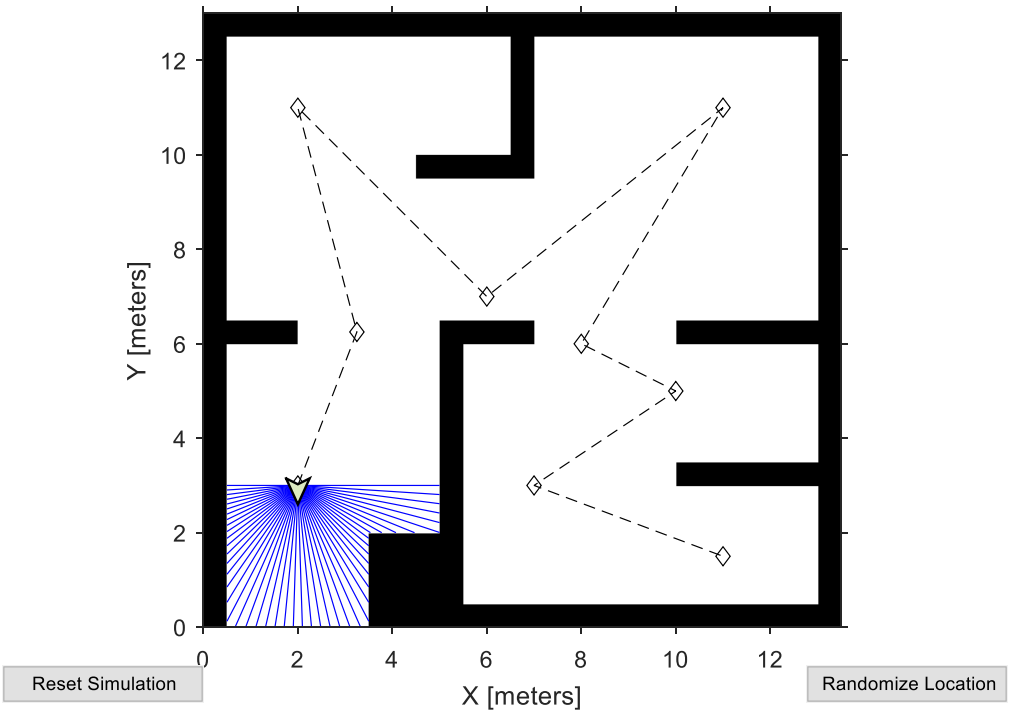
2.3 Create a Path Controller

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.

```
controller = robotics.PurePursuit('Waypoints', path);
controller.DesiredLinearVelocity = 0.4;
```

Set the controller rate to run at 10 Hz.

```
controlRate = robotics.Rate(10);
```

Define a goal point and a goal radius to stop the robot at the end of the path.

```
goalRadius = 0.1;
robotCurrentLocation = path(1,:);
robotGoal = path(end,:);
distanceToGoal = norm(robotCurrentLocation - robotGoal);
```

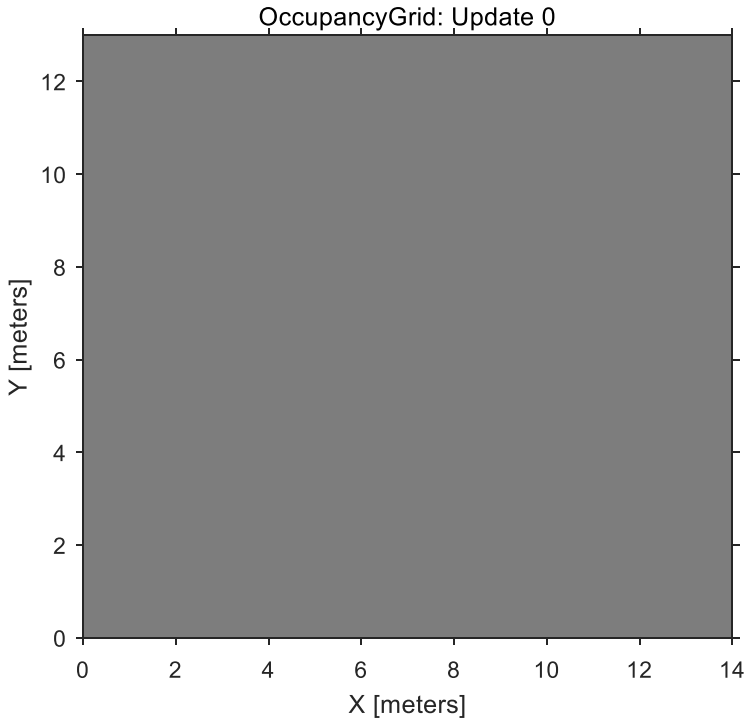
2.4 Define an Empty Map

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.

```
map = robotics.OccupancyGrid(14,13,20);
```

Visualize the map in the figure window.

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



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.

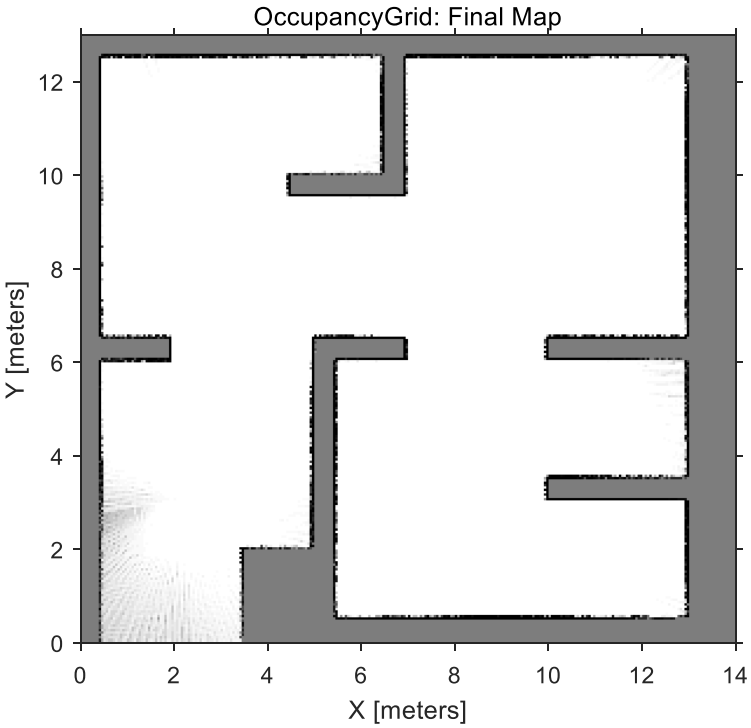
Visualize the map after every 50 updates.

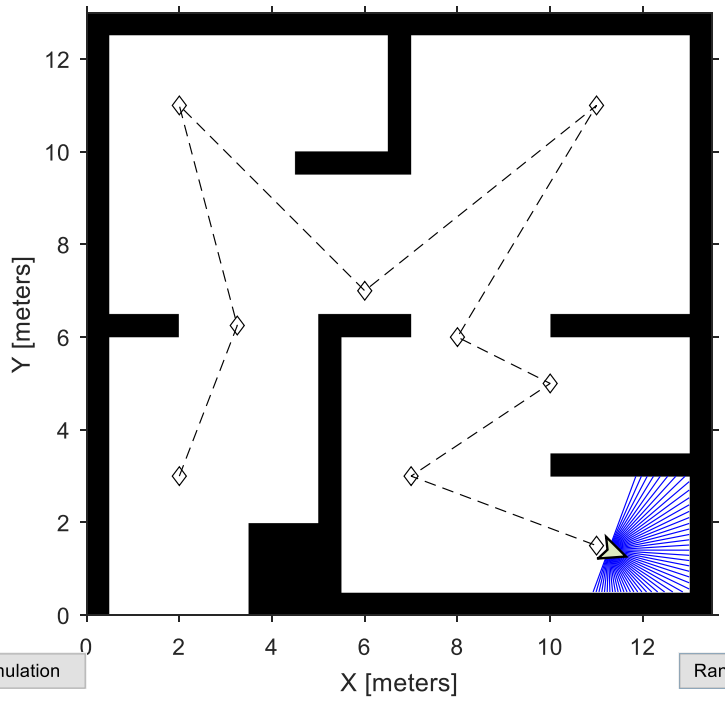
```
updateCounter = 1;
while( distanceToGoal > goalRadius )
    % Receive a new laser sensor reading
    scanMsg = receive(scanSub);

    % 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 incorpørated all the sensor readings.

Homework2 Problem3：Please implement the function insertRay yourself. (Tips: using "help insertRay" can get some information useful for you)

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





Reset Simulation

Randomize Location

2.5 Shutdown ROS Network

Shut down the ROS master and delete the global node.

```
rosshutdown
```