```
% =========
% ass3_q1.m
% =========
%
% This assignment will introduce you to the idea of first building an
% occupancy grid then using that grid to estimate a robot's motion
 using a
% particle filter.
%
% There are two questions to complete (5 marks each):
%
%    Question 1: code occupancy mapping algorithm
%    Question 2: see ass3_q2.m
%
% Fill in the required sections of this script with your code, run it
 to
% generate the requested plot/movie, then paste the plots into a short
 report
% that includes a few comments about what you've observed.  Append
 your
% version of this script to the report.  Hand in the report as a PDF
 file
% and the two resulting AVI files from Questions 1 and 2.
%
% requires: basic Matlab, 'gazebo.mat'
%
% T D Barfoot, January 2016
%
clear ;
close all
clc

% set random seed for repeatability
rng(1);

% ==========================
% load the dataset from file
% ==========================
%
%    ground truth poses: t_true x_true y_true theta_true
% odometry measurements: t_odom v_odom omega_odom
%           laser scans: t_laser y_laser
%    laser range limits: r_min_laser r_max_laser
%    laser angle limits: phi_min_laser phi_max_laser
%
load gazebo.mat;

% =======================================
% Question 1: build an occupancy grid map
% =======================================
%
```

```
% Write an occupancy grid mapping algorithm that builds the map from
 the
% perfect ground-truth localization.  Some of the setup is done for
 you
% below.  The resulting map should look like "ass2_q1_soln.png".  You
 can
% watch the movie "ass2_q1_soln.mp4" to see what the entire mapping
 process
% should look like.  At the end you will save your occupancy grid map
 to
% the file "occmap.mat" for use in Question 2 of this assignment.

% allocate a big 2D array for the occupancy grid
ogres = 0.05;                   % resolution of occ grid
ogxmin = -7;                    % minimum x value
ogxmax = 8;                     % maximum x value
ogymin = -3;                    % minimum y value
ogymax = 6;                     % maximum y value
ognx = (ogxmax-ogxmin)/ogres;   % number of cells in x direction
ogny = (ogymax-ogymin)/ogres;   % number of cells in y direction
oglo = zeros(ogny,ognx);        % occupancy grid in log-odds format
ogp = zeros(ogny,ognx);         % occupancy grid in probability format

% precalculate some quantities
numodom = size(t_odom,1);
npoints = size(y_laser,2);
angles = linspace(phi_min_laser, phi_max_laser,npoints);
angle_inc = (phi_max_laser-phi_min_laser)/npoints;
dx = ogres*cos(angles);
dy = ogres*sin(angles);

% interpolate the noise-free ground-truth at the laser timestamps
t_interp = linspace(t_true(1),t_true(numodom),numodom);
x_interp = interp1(t_interp,x_true,t_laser);
y_interp = interp1(t_interp,y_true,t_laser);
theta_interp = interp1(t_interp,theta_true,t_laser);
omega_interp = interp1(t_interp,omega_odom,t_laser);

% set up the plotting/movie recording
vid = VideoWriter('ass2_q1.avi');
open(vid);
figure(1);
clf;
pcolor(ogp);
colormap(1-gray);
shading('flat');
axis equal;
axis off;
M = getframe;
writeVideo(vid,M);

% loop over laser scans (every fifth)
for i=1:5:size(t_laser,1)
```

```
    % ------insert your occupancy grid mapping algorithm here------

    % grid mapping parameters
    alpha = 2;
    beta  = 1;

    for j = 1:size(angles, 2)

        % range value for given timestamp and angle
        range_i_j = y_laser(i, j);

        if ~isnan(range_i_j)

            % obstacle position in map frame
            angle_total = theta_interp(i) + (phi_min_laser
+j*angle_inc);
            x_robot_map = x_interp(i)-ogxmin;
            y_robot_map = y_interp(i)-ogymin;
            x_obs_map   = x_robot_map + range_i_j*cos(angle_total);
            y_obs_map   = y_robot_map + range_i_j*sin(angle_total);

            % mapping obstacle and robot position to grid frame
            x_obs_map_px = min(ognx, round(x_obs_map/ogres));
            y_obs_map_px = min(ogny, round(y_obs_map/ogres));
            x_robot_map_px = round(x_robot_map/ogres);
            y_robot_map_px = round(y_robot_map/ogres);

            % updating the log-odds matrix for obstacle position
            obs_row_point = y_obs_map_px;
            obs_col_point = x_obs_map_px;
            oglo(obs_row_point,obs_col_point) = ...
                oglo(obs_row_point,obs_col_point) + alpha;

            % updating the log-odds matrix for free positions
            rob_row_point = y_robot_map_px;
            rob_col_point = x_robot_map_px;
            oglo(rob_row_point,rob_col_point) = ...
                oglo(rob_row_point,rob_col_point) - beta;
            [row_idx, col_idx] = indices_along_diag(range_i_j,
 angle_total, ogres);
            for k = 1:size(row_idx, 2)
                id_row = rob_row_point + row_idx(k);
                id_col = rob_col_point + col_idx(k);
                oglo(id_row,id_col) = oglo(id_row,id_col) - beta;
            end

        end
%         break
    end

    % recovering probabilities from log-odds
    ogp = exp(oglo)./(1 + exp(oglo));
```

```
    % ------end of your occupancy grid mapping algorithm-------

    % draw the map
    clf;
    pcolor(ogp);
    colormap(1-gray);
    shading('flat');
    axis equal;
    axis off;

    % draw the robot
    hold on;
    x = (x_interp(i)-ogxmin)/ogres;
    y = (y_interp(i)-ogymin)/ogres;
    th = theta_interp(i);
    r = 0.15/ogres;
    set(rectangle( 'Position', [x-r y-r 2*r 2*r], 'Curvature', [1
 1]),'LineWidth',2,'FaceColor',[0.35 0.35 0.75]);
    set(plot([x x+r*cos(th)]', [y y+r*sin(th)]', 'k-'),'LineWidth',2);

    % save the video frame
    M = getframe;
    writeVideo(vid,M);

    pause(0.1);

end

close(vid);
print -dpng ass2_q1.png

save occmap.mat ogres ogxmin ogxmax ogymin ogymax ognx ogny oglo ogp;
```

custom function definitions

```
% given two points in a matrix, returns the diagonal indices
function [row_idx, col_idx] = indices_along_diag(range, angle, ogres)
    % angle        - angle made by the ray with robot's x axis

    row_idx    = [];
    col_idx    = [];
    new_angle = atan2(sin(angle), cos(angle));

    % angle is between -pi/4 and pi/4
    if -pi/4<=new_angle && new_angle<=pi/4
        x_length = round(range*cos(new_angle)/ogres);
        for i=1:x_length-1
            col_idx = [col_idx i];
            row_idx = [row_idx round(i*tan(new_angle))];

        end
```

```
    % angle is between -3pi/4 and 3pi/4
    elseif 3*pi/4<=new_angle || new_angle<=-3*pi/4
        x_length = round(range*cos(new_angle)/ogres);
        for i=1:abs(x_length)-1
            col_idx = [col_idx -i];
            row_idx = [row_idx -round(i*tan(new_angle))];

        end

    % angle is between pi/4 and 3pi/4
    elseif -pi/4<new_angle && new_angle<3*pi/4
        y_length = round(range*sin(new_angle)/ogres);
        for i=1:y_length-1
            row_idx = [row_idx i];
            col_idx = [col_idx round(i/tan(new_angle))];

        end

    % angle is between -pi/4 and -3pi/4
    else
        y_length = round(range*sin(new_angle)/ogres);
        for i=1:abs(y_length)-1
            row_idx = [row_idx -i];
            col_idx = [col_idx -round(i/tan(new_angle))];

        end

    end


end
```

Published with MATLAB® R2020a