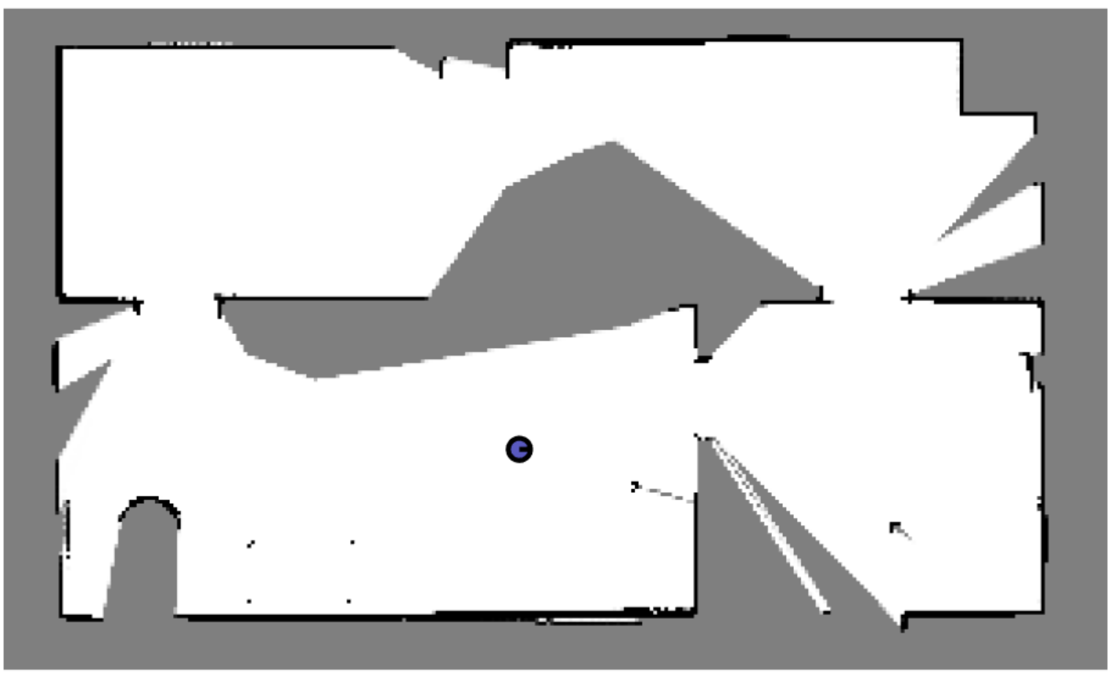
Assignment 3 Report - ROB521
Aditya Jain

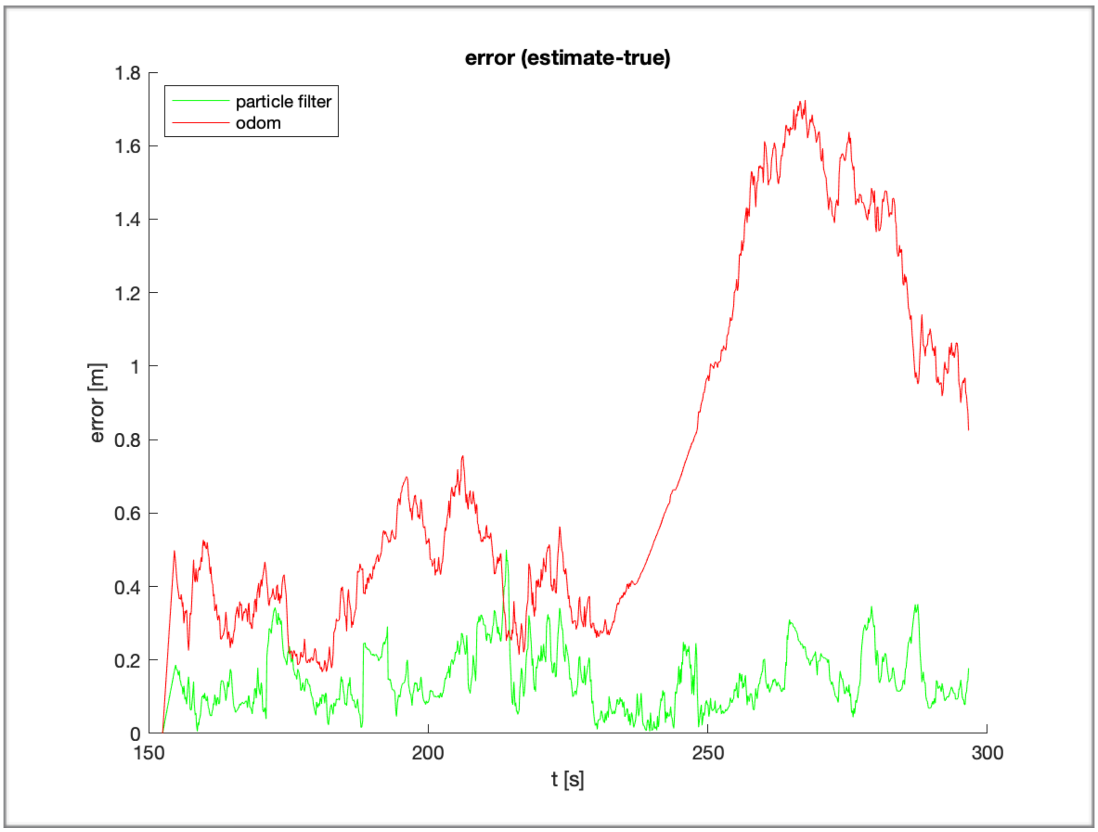
Part I: Occupancy Grid Map

The first part of the assignment is to build a map using occupancy grid mapping. For a given laser <range, angle> and robot pose <x, y, theta>, the log-odds matrix is incremented with alpha=+2 for the obstacle position and the free locations are decremented with beta=-1 along the ray direction. The final map is built by recovering the probabilities form the log-odds matrix. The black pixels are obstacles, white pixels are free locations and grey areas are the uncertain positions.



Part II: Localization using Particle Filters

The second part is to do state estimation of the robot using particle filters and the occupancy map built in previous part. The most important task in this part is the calculation of importance weight for each particle, which further influences the resampling step. Given a particle's pose <x_p, y_p, theta_p> and angle of the laser beam, the expected laser range *y_expected* is calculated using the occupancy map. We also have the actual measurement *y_measurement* from the laser sensor. The likelihood of *y_measurement* is calculated given a gaussian distribution with mean at *y_expected* and a given laser variance. This serves as the weight of the particle for the given laser measurement.



```
% =========
% 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

```
% =========
% ass3_q2.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 three questions to complete (5 marks each):
%
%    Question 1: see ass3_q1.m
%    Question 2: code particle filter to localize from known map
%
% 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', 'occmap.mat'
%
% T D Barfoot, January 2016
%
clear;
clc;
close all

% 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;

% =================================================
% load the occupancy map from question 1 from file
% =================================================
%  ogres: resolution of occ grid
% ogxmin: minimum x value
```

```
% ogxmax: maximum x value
% ogymin: minimum y value
% ogymax: maximum y value
%   ognx: number of cells in x direction
%   ogny: number of cells in y direction
%   oglo: occupancy grid in log-odds format
%    ogp: occupancy grid in probability format
load occmap.mat;

%
 =========================================================================
% Question 2: localization from an occupancy grid map using particle
 filter
%
 =========================================================================
%
% Write a particle filter localization algorithm to localize from the
 laser
% rangefinder readings, wheel odometry, and the occupancy grid map you
% built in Question 1.  We will only use two laser scan lines at the
% extreme left and right of the field of view, to demonstrate that the
% algorithm does not need a lot of information to localize fairly
 well.  To
% make the problem harder, the below lines add noise to the wheel
 odometry
% and to the laser scans.  You can watch the movie "ass2_q2_soln.mp4"
 to
% see what the results should look like.  The plot "ass2_q2_soln.png"
 shows
% the errors in the estimates produced by wheel odometry alone and by
 the
% particle filter look like as compared to ground truth; we can see
 that
% the errors are much lower when we use the particle filter.

% interpolate the noise-free ground-truth at the laser timestamps
numodom = size(t_odom,1);
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);

% interpolate the wheel odometry at the laser timestamps and
% add noise to measurements (yes, on purpose to see effect)
v_interp = interp1(t_interp,v_odom,t_laser) +
 0.2*randn(size(t_laser,1),1);
omega_interp = interp1(t_interp,omega_odom,t_laser) +
 0.04*randn(size(t_laser,1),1);

% add noise to the laser range measurements (yes, on purpose to see
 effect)
% and precompute some quantities useful to the laser
y_laser = y_laser + 0.1*randn(size(y_laser));
```

```
npoints = size(y_laser,2);
angles = linspace(phi_min_laser, phi_max_laser,npoints);
dx = ogres*cos(angles);
dy = ogres*sin(angles);
y_laser_max = 5;   % don't use laser measurements beyond this distance

% particle filter tuning parameters (yours may be different)
nparticles = 200;       % number of particles
v_noise = 0.2;          % noise on longitudinal speed for propagating
 particle
u_noise = 0.2;          % noise on lateral speed for propagating
 particle
omega_noise = 0.04;     % noise on rotational speed for propagating
 particle
laser_var = 0.5^2;      % variance on laser range distribution
w_gain = 10*sqrt( 2 * pi * laser_var );     % gain on particle weight

% generate an initial cloud of particles
x_particle = x_true(1) + 0.5*randn(nparticles,1);
y_particle = y_true(1) + 0.3*randn(nparticles,1);
theta_particle = theta_true(1) + 0.1*randn(nparticles,1);

% compute a wheel odometry only estimate for comparison to particle
% filter
x_odom_only = x_true(1);
y_odom_only = y_true(1);
theta_odom_only = theta_true(1);

% error variables for final error plots - set the errors to zero at
 the start
pf_err(1) = 0;
wo_err(1) = 0;

% set up the plotting/movie recording
vid = VideoWriter('ass2_q2.avi');
open(vid);
figure(2);
clf;
hold on;
pcolor(ogp);
set(plot( (x_particle-ogxmin)/ogres, (y_particle-ogymin)/ogres, 'g.'
 ),'MarkerSize',10,'Color',[0 0.6 0]);
set(plot( (x_odom_only-ogxmin)/ogres, (y_odom_only-ogymin)/ogres, 'r.'
 ),'MarkerSize',20);
x = (x_interp(1)-ogxmin)/ogres;
y = (y_interp(1)-ogymin)/ogres;
th = theta_interp(1);
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);
set(plot( (mean(x_particle)-ogxmin)/ogres, (mean(y_particle)-ogymin)/
ogres, 'g.' ),'MarkerSize',20);
colormap(1-gray);
```

```
shading('flat');
axis equal;
axis off;
M = getframe;
writeVideo(vid,M);


% loop over laser scans
for i=2:size(t_laser,1)

    % update the wheel-odometry-only algorithm
    dt = t_laser(i) - t_laser(i-1);
    v = v_interp(i);
    omega = omega_interp(i);
    x_odom_only = x_odom_only + dt*v*cos( theta_odom_only );
    y_odom_only = y_odom_only + dt*v*sin( theta_odom_only );
    phi = theta_odom_only + dt*omega;
    while phi > pi
        phi = phi - 2*pi;
    end
    while phi < -pi
        phi = phi + 2*pi;
    end
    theta_odom_only = phi;

    % loop over the particles
    for n=1:nparticles

        % propagate the particle forward in time using wheel odometry
        % (remember to add some unique noise to each particle so they
        % spread out over time)
        v = v_interp(i) + v_noise*randn(1);
        u = u_noise*randn(1);
        omega = omega_interp(i) + omega_noise*randn(1);
        x_particle(n) = x_particle(n) + dt*(v*cos( theta_particle(n) )
 - u*sin( theta_particle(n) ));
        y_particle(n) = y_particle(n) + dt*(v*sin( theta_particle(n) )
 + u*cos( theta_particle(n) ));
        phi = theta_particle(n) + dt*omega;
        while phi > pi
            phi = phi - 2*pi;
        end
        while phi < -pi
            phi = phi + 2*pi;
        end
        theta_particle(n) = phi;

        % pose of particle in initial frame
        T = [cos(theta_particle(n)) -sin(theta_particle(n))
 x_particle(n); ...
                sin(theta_particle(n))  cos(theta_particle(n))
 y_particle(n); ...
                    0                     0                    1];
```

```
        % compute the weight for each particle using only 2 laser rays
        % (right=beam 1 and left=beam 640)
        w_particle(n) = 1.0;
        for beam=1:2

            % we will only use the first and last laser ray for
            % localization
            if beam==1  % rightmost beam
                j = 1;
                beam_angle = phi_min_laser;
            elseif beam==2  % leftmost beam
                j = 640;
                beam_angle = phi_max_laser;
            end

            % ------insert your particle filter weight calculation
here ------
            threshold = 0.5;
            if ~isnan(y_laser(i,j))
                row_pos = max(1,round((y_particle(n)-ogymin)/ogres));
                col_pos = max(1,round((x_particle(n)-ogxmin)/ogres));

                y_measurement = y_laser(i,j);
                y_expected    = exp_meas(row_pos, col_pos,
theta_particle(n)...
                    , beam_angle, ogp, ogres, threshold, y_laser_max);

                w_particle(n) =
w_gain*w_particle(n)*normpdf(y_measurement,...
                    y_expected, sqrt(laser_var));

            end             % ------end of your particle filter weight
calculation-------
        end

    end

    % resample the particles using Madow systematic resampling
    w_bounds = cumsum(w_particle)/sum(w_particle);
    w_target = rand(1);
    j = 1;
    for n=1:nparticles
       while w_bounds(j) < w_target
           j = mod(j,nparticles) + 1;
       end
       x_particle_new(n) = x_particle(j);
       y_particle_new(n) = y_particle(j);
       theta_particle_new(n) = theta_particle(j);
       w_target = w_target + 1/nparticles;
       if w_target > 1
           w_target = w_target - 1.0;
           j = 1;
       end
    end
```

```
    x_particle = x_particle_new;
    y_particle = y_particle_new;
    theta_particle = theta_particle_new;

    % save the translational error for later plotting
    pf_err(i) = sqrt( (mean(x_particle) - x_interp(i))^2 +
 (mean(y_particle) - y_interp(i))^2 );
    wo_err(i) = sqrt( (x_odom_only - x_interp(i))^2 + (y_odom_only -
 y_interp(i))^2 );

    % plotting
    figure(2);
    clf;
    hold on;
    pcolor(ogp);
    set(plot( (x_particle-ogxmin)/ogres, (y_particle-ogymin)/
ogres, 'g.' ),'MarkerSize',10,'Color',[0 0.6 0]);
    set(plot( (x_odom_only-ogxmin)/ogres, (y_odom_only-ogymin)/
ogres, 'r.' ),'MarkerSize',20);
    x = (x_interp(i)-ogxmin)/ogres;
    y = (y_interp(i)-ogymin)/ogres;
    th = theta_interp(i);
    if ~isnan(y_laser(i,1)) & y_laser(i,1) <= y_laser_max
       set(plot([x x+y_laser(i,1)/ogres*cos(th+angles(1))]', [y y
+y_laser(i,1)/ogres*sin(th+angles(1))]', 'm-'),'LineWidth',1);
    end
    if ~isnan(y_laser(i,640)) & y_laser(i,640) <= y_laser_max
       set(plot([x x+y_laser(i,640)/ogres*cos(th+angles(640))]', [y y
+y_laser(i,640)/ogres*sin(th+angles(640))]', 'm-'),'LineWidth',1);
    end
    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);
    set(plot( (mean(x_particle)-ogxmin)/ogres, (mean(y_particle)-
ogymin)/ogres, 'g.' ),'MarkerSize',20);
    colormap(1-gray);
    shading('flat');
    axis equal;
    axis off;

    % save the video frame
    M = getframe;
    M.cdata = M.cdata(1:343, 1:434, :);
    writeVideo(vid,M);

    pause(0.01);

end

close(vid);

% final error plots
figure(3);
```

```
clf;
hold on;
plot( t_laser, pf_err, 'g-' );
plot( t_laser, wo_err, 'r-' );
xlabel('t [s]');
ylabel('error [m]');
legend('particle filter', 'odom', 'Location', 'NorthWest');
title('error (estimate-true)');
print -dpng ass2_q2.png
```

custom function definitions

```
% returns the expected laser measurement given the particle's
% position and map
function y_exp = exp_meas(row, col, theta, beam_angle, map, ogres,
 thresh, y_max)

    new_angle = atan2(sin(theta+beam_angle), cos(theta+beam_angle));
    incr  = 0;
    r_p = row;
    c_p = col;

    % angle is between -pi/4 and pi/4
    if -pi/4<=new_angle && new_angle<=pi/4
        while r_p>0 && c_p>0 && r_p<=180 && c_p<=300 && map(r_p,
 c_p)<thresh
            incr = incr + 1;
            c_p = col + incr;
            r_p = row + round(incr*tan(new_angle));

        end
        y_exp = abs((incr/cos(new_angle))*ogres);

    % angle is between -3pi/4 and 3pi/4
    elseif 3*pi/4<=new_angle || new_angle<=-3*pi/4
        while r_p>0 && c_p>0 && r_p<=180 && c_p<=300 && map(r_p,
 c_p)<thresh
            incr = incr + 1;
            c_p = col - incr;
            r_p = row - round(incr*tan(new_angle));

        end
        y_exp = abs((incr/cos(new_angle))*ogres);


    % angle is between pi/4 and 3pi/4
    elseif pi/4<new_angle && new_angle<3*pi/4
        while r_p>0 && c_p>0 && r_p<=180 && c_p<=300 && map(r_p,
 c_p)<thresh
            incr = incr + 1;
            r_p = row + incr;
            c_p = col + round(incr/tan(new_angle));
        end
```

```
        y_exp = abs((incr/sin(new_angle))*ogres);


    % angle is between -pi/4 and -3pi/4
    else
        while r_p>0 && c_p>0 && r_p<=180 && c_p<=300 && map(r_p,
 c_p)<thresh
            incr = incr + 1;
            r_p = row - incr;
            c_p = col - round(incr/tan(new_angle));
        end
        y_exp = abs((incr/sin(new_angle))*ogres);


    end

end
```

Published with MATLAB® R2020a