# ROB521 A2: Occupancy Grids and Particle Filters

Samuel Atkins, 1002951754
March 8, 2020

## Question 1: Occupancy Grid

In this question, students were to build an occupancy grid using the information produced by a simulated mobile robot at various time steps. The mobile robot yields laser scan information, the mobile robot's location in an inertial frame, and the mobile robot's orientation. To accomplish the task of constructing an occupancy grid, a simple algorithm was designed. At each time step, a transformation matrix that transforms points from the robot's frame into the inertial frame is constructed. Then, each laser in the scan corresponding to the current time step is iterated over. The points at the end of each laser are indicative of obstacles. As such, the point at the end of the laser being analyzed is translated into coordinates and increases the likelihood that that particular map entry is occupied. Each subsequent point prior to the end of the laser is also translated into coordinates. These points increase the probability that those particular squares are unoccupied. After every laser is iterated over, the map is thresholded. Squares that are more likely to be occupied are marked as occupied and squares that are more likely to be unoccupied are marked as unoccupied. Figure 1, below, is the result produced for this section:

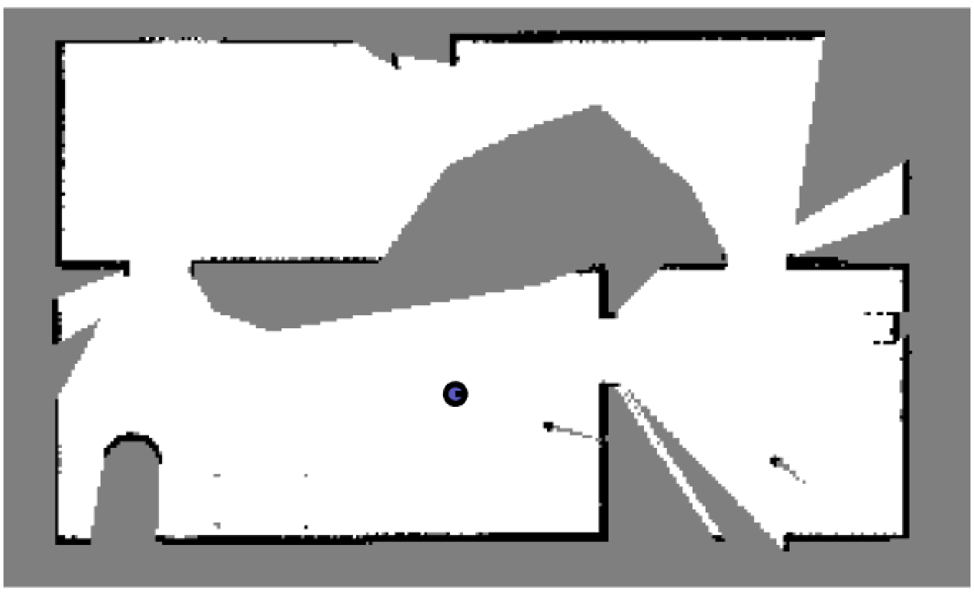

*Figure 1: Resultant occupancy grid*

Measurements in which the angular velocity was greater than 0.1 were ignored. Given the limited range of the sensor measurements and the robot's ignorance, the above Figure is a strong representation of the scene. If the robot stopped to look around every once in a while, the map would be much more complete. Furthermore, many measurements were ignored due to the large angular velocity magnitudes at certain steps.

## Question 2: Particle Filter

In this question, a particle filter was implemented to estimate the location of the robot using the sensor data. Given the computational complexity of implementing a particle filter, only the outermost sensor readings were used. These sensor readings were compared with the expected sensor readings given the positions of each point. To update the weight associated with each particle, the euclidean distance was calculated between the expected and actual laser measurements. Then, the weights were updated assuming a Gaussian distribution. To evaluate the performance of the particle filter, the weighted average of all of the particles was computed. The error produced by this weighted average and the error produced by an odometry based method are illustrated in Figure 2 below:

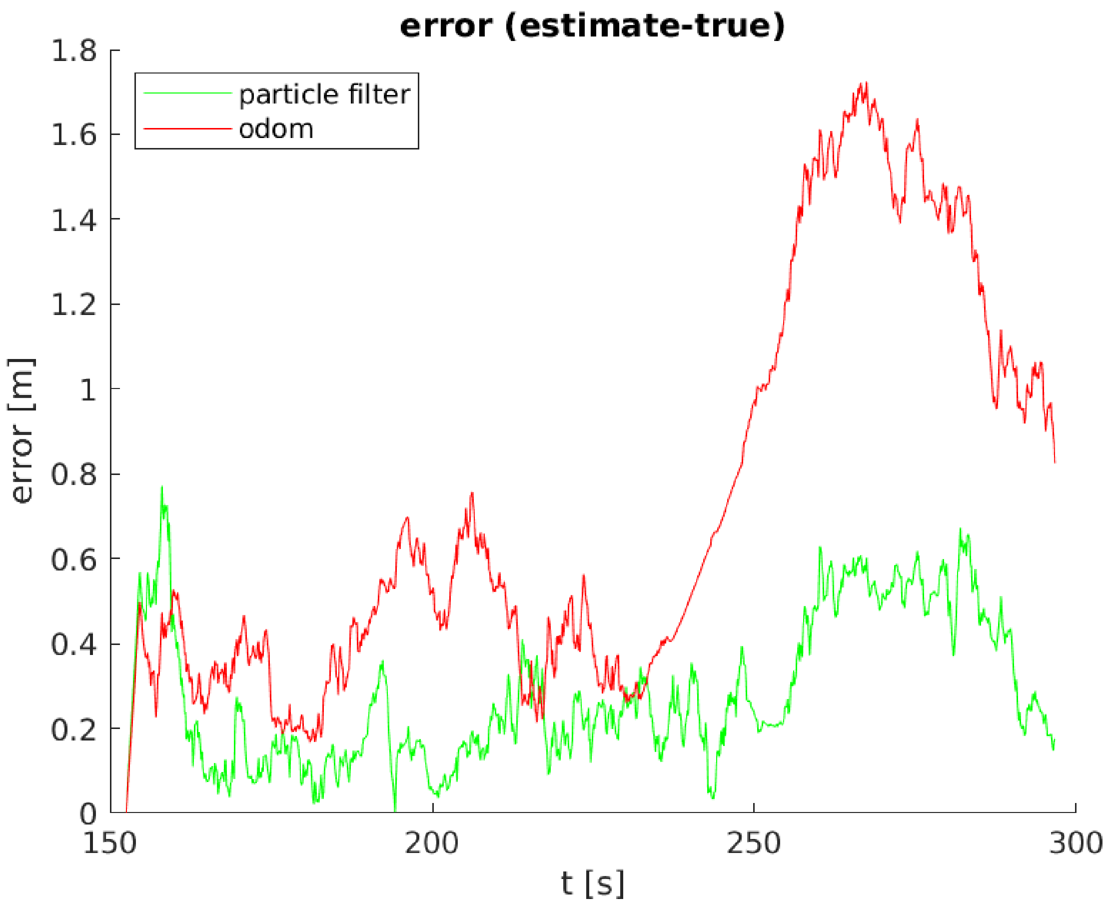


*Figure 2: Particle filter location estimation vs. an odometry based method*

From the above Figure, it is clear that the particle filter performs much better than the odometry based method, especially as time passes. At the beginning of the simulation, the particle filter has more error than the odometry based localization method. Unlike the odometry based localization method, which is a dead-reckoning method, the particle filter is able to constrain its error and correct its location estimate using the additional sensory data.

# Appendix A: MATLAB Code – ass2_q1.m

```
% =========
% ass2_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 ass2_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 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;

% =======================================
% 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);
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);

alpha = 10;
beta = 10;

% loop over laser scans (every fifth)
for i=1:5:size(t_laser,1)
    % ------insert your occupancy grid mapping algorithm here------
    if (abs(omega_interp(i)) > 0.1)
        continue
    end
    TIR = [
      cos(theta_interp(i)) -sin(theta_interp(i)) 0 x_interp(i) - 0.1 *
cos(theta_interp(i));
      sin(theta_interp(i)) cos(theta_interp(i)) 0 y_interp(i);
      0 0 1 0;
      0 0 0 1;
    ];
    unocc_pt_index = 1;
    occ_pt_index = 1;

    unocc_pts = zeros([4, 1]);
    occ_pts = zeros([4, 1]);
```

```
    for k=1:size(y_laser,2)
        if isnan(y_laser(i, k))
            continue
        end
        increments = round(y_laser(i, k)/0.05);
        for p=1:increments
            point_range = 0.05*p;
            if p <= increments - 2
                unocc_pts(:, unocc_pt_index) = [point_range * ...
                    cos(angles(k)); point_range * sin(angles(k)); 0; 1];
                unocc_pt_index = unocc_pt_index + 1;
            else
                occ_pts(:, occ_pt_index) = [point_range * ...
                    cos(angles(k)); point_range * sin(angles(k)); 0; 1];
                occ_pt_index = occ_pt_index + 1;
            end
        end
    end
    unocc_pts_inertial = TIR * unocc_pts;
    occ_pts_inertial = TIR * occ_pts;

    unocc_pts_coords = unocc_pts_inertial + [
        ones(1, size(unocc_pts_inertial, 2))*(7);
        ones(1, size(unocc_pts_inertial, 2))*(3);
        zeros(1, size(unocc_pts_inertial, 2));
        zeros(1, size(unocc_pts_inertial, 2));
        ];
    unocc_pts_coords(1:2, :) = round(unocc_pts_coords(1:2, :)/0.05);
    unocc_indeces = sub2ind(size(ogp), unocc_pts_coords(2, :),
unocc_pts_coords(1, :));

    occ_pts_coords = occ_pts_inertial + [
        ones(1, size(occ_pts_inertial, 2))*(7);
        ones(1, size(occ_pts_inertial, 2))*(3);
        zeros(1, size(occ_pts_inertial, 2));
        zeros(1, size(occ_pts_inertial, 2));
        ];
    occ_pts_coords(1:2, :) = round(occ_pts_coords(1:2, :)/0.05);
    occ_indeces = sub2ind(size(ogp), occ_pts_coords(2, :), occ_pts_coords(1,
:));


    oglo(unocc_indeces) = ogp(unocc_indeces) - beta;
    oglo(occ_indeces) = ogp(occ_indeces) + alpha;

    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;
```

# Appendix B: MATLAB Code – ass2_q2.m

```
% =========
% ass2_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 ass2_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 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;
            elseif beam==2  % leftmost beam
                j = 640;
            end

            % ------insert your particle filter weight calculation here ---
            % Ensuring real measurement:
            if isnan(y_laser(i, j))
                continue
            end

            % Transformation matrix from inertial frame to robot frame:
            TIR = [
                cos(theta_interp(i)) -sin(theta_interp(i)) x_interp(i);
                sin(theta_interp(i)) cos(theta_interp(i)) y_interp(i);
                0 0 1];

            % Calculating the end of the laser in the inertial frame:
            laser_end = [
                y_laser(i, j)*cos(angles(j)) - 0.1;
                y_laser(i, j)*sin(angles(j));
                1];
            laser_end = TIR * laser_end;
```

```
            % Transforming laser's point into coordinates:
            x_coord_laser = round((laser_end(1)-ogxmin)/ogres);
            y_coord_laser = round((laser_end(2)-ogymin)/ogres);


            % Calculating the laser prediction of the current particle:
            prediction_matrix = T * [cos(angles(j)) -sin(angles(j)) -0.1;
                    sin(angles(j)) cos(angles(j))  0;
                    0 0 1];
            x_prediction = prediction_matrix(1, 3);
            y_prediction = prediction_matrix(2, 3);
            theta_prediction = acos(prediction_matrix(1, 1));

            % Transforming to coordinates:
            x_coord_prediction = round((x_prediction - ogxmin)/ogres);
            if (x_coord_prediction > 300)
                x_coord_prediction = 300;
            else
                if (x_coord_prediction < 1)
                    x_coord_prediction = 1;
                end
            end
            y_coord_prediction = round((y_prediction - ogymin)/ogres);
            if (y_coord_prediction > 180)
                y_coord_prediction = 180;
            else
                if (y_coord_prediction < 1)
                    y_coord_prediction = 1;
                end
            end

            for p=0.45:(ogres/2):y_laser_max
                % Now we will traverse the ray and stop once we have found
                % an obstacle:
                if (oglo(round(y_coord_prediction),
round(x_coord_prediction)) >= 0)
                    continue
                end

                x_coord_prediction =  x_coord_prediction + ...
                    p * cos(theta_prediction);
                if (x_coord_prediction > 300)
                    x_coord_prediction = 300;
                else
                    if (x_coord_prediction < 1)
                        x_coord_prediction = 1;
                    end
                end
                y_coord_prediction =  y_coord_prediction + ...
                    p * sin(theta_prediction);
                if (y_coord_prediction > 180)
                    y_coord_prediction = 180;
                else
                    if (y_coord_prediction < 1)
                        y_coord_prediction = 1;
```

```
                end
            end
        end

        % Weight update equation:
        error = norm([x_coord_prediction; y_coord_prediction] - ...
            [x_coord_laser; y_coord_laser]);
        gaussian_update = (1/sqrt(2 * pi * laser_var)) * ...
            exp(-error^2/(2 * laser_var));
        w_particle(n) = w_particle(n) + w_gain * gaussian_update;

        % ------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;
    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
```