ROB521 Assignment 3: Particle Filtering on an Occupancy Map

Andrew Jairam
1006941972
andrew.jairam@mail.utoronto.ca

Part 1: Occupancy Mapping using LIDAR

In this part, LIDAR mapping is performed to create an occupancy map by first initializing a log-odds map of the environment and looping over all the laser scans. For each of the scans at timestep $$t$$, the scan is transformed into a world frame coordinate on the occupancy grid by using the robot pose $$t$$ and the occupancy map resolution, and the end point is taken as the nearest obstacle. A line is formed from the current robot coordinates and the obstacle, where both coordinates are taken in the world frame, by using linear interpolation to step in map coordinates by some stepsize determined as the direction of most change (i.e. the maximum difference in either $$x$$ or $$y$$). Any cell along this line is classified as unoccupied, while the end of the line is classified as occupied: which is the nearest obstacle. A factor of $$\alpha = 2$$ is added to the occupied cells and $$\beta$$ is decremented to the unoccupied cells of the log-odds map, where $$\alpha > \beta$$ is chosen so that obstacle detection is weighted higher than free space detection. If the occupancy grid coordinates of the laser endpoint lies outside the map, the scan at this timestep is ignored, since this could result in the map's borders being incorrectly classified as occupied space. This was not an issue for this map where all the borders are occupied space, but could be a potential issue for other scenarios where there could be an open passageway out of the room. The log-odds map is then converted to a probability map, where $$p > 0.5$$ can be taken as occupied cells.

Figure 1 shows the resulting map created from the LIDAR mapping process, which closely matches the real map. There are a few noisy artifacts where obstacles marked in black are contaminating the free space. The reason for this is that the LIDAR likely did not see these spots as freespace enough, or the linear interpolation method to connect the robot and the endpoint of the scan was too inaccurate. Potential fixes would include using more sophisticated line estimation or exploring for longer to give more data points to correctly classify these regions as freespace.

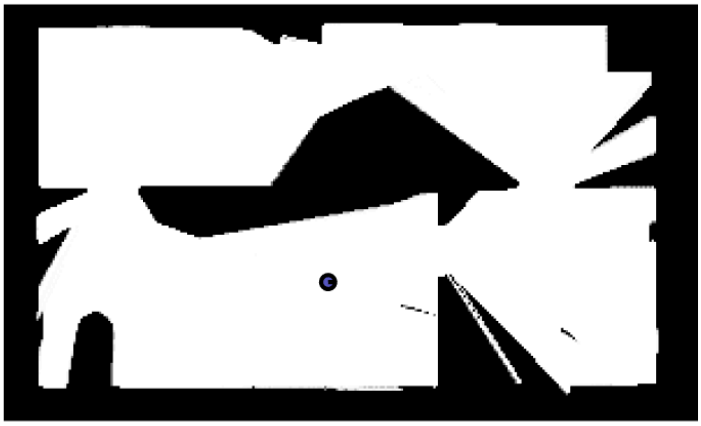

Figure 1: Resulting LIDAR Occupancy Map from Part 1

Part 2: Particle Filter Localization

With the built occupancy map, particle filtering to localize the robot is now performed. 200 particles are used, initialized at gaussian randomly distributed locations centered around the initial robot pose. To demonstrate that the algorithm does not need a lot of information to perform well, only the leftmost and rightmost scans are used at each timestep. To perform the weight updates for the particle filter, the scans are first saved as the current hypothesis measurement, then transformed into the world frame, and then into the occupancy grid frame to get coordinates of the endpoint in the grid. Using these coordinates, the expected measurement is calculated in three cases:

1. The endpoint is not within the map coordinates: the expected measurement is taken as the maximum range of the laser
2. The endpoint is in the map but is not an obstacle: the expected measurement is also taken as the maximum range of the laser since an obstacle is not seen in range
3. An obstacle in the map is detected: take the expected measurement as the euclidean distance from the current robot position and the obstacle, both in the world frame

The particles are then reweighted by passing through a Gaussian Likelihood model, where the mean is taken as the expected measurement, and the input is the hypothesis laser scan. The variance of the laser scans is assumed to be known and fixed, and the Gaussian likelihood is scaled by some gain.

Figure 2 compares the performance of wheel odometry to particle filtering. The below graph illustrates the advantage of particle filtering: being a probabilistic model, particle filtering handles uncertainty: correcting the significant drift that is exhibited by wheel odometry. The error stays within some low margin, while the error in wheel odometry blows up as the vehicle keeps driving. However, the main tradeoff in using this method is that a map is required beforehand: while wheel odometry can provide some

localization estimate without one. With a map, this experiment shows that we can take advantage of more robust localization methods, resulting in better pose estimates.

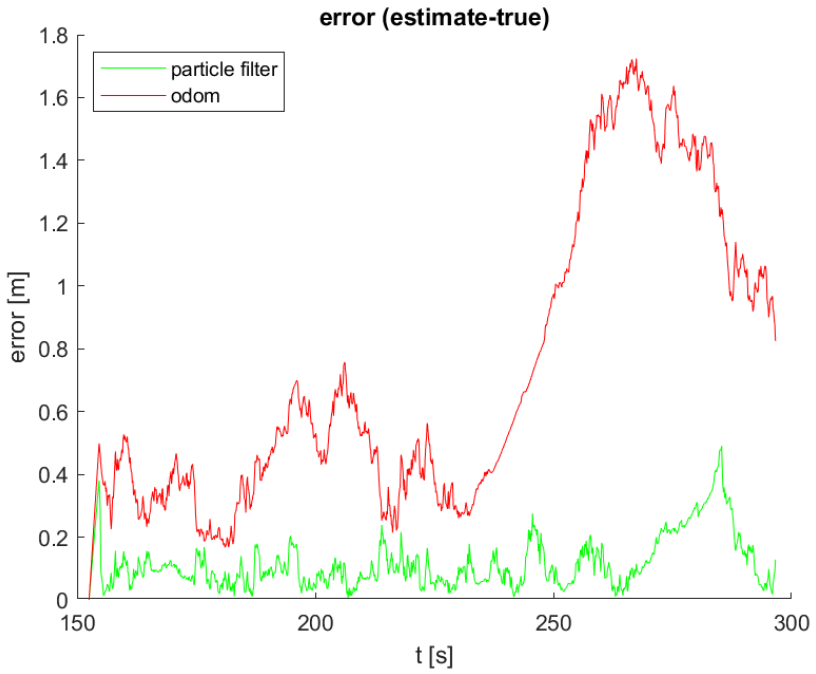

Figure 2: Comparison of Particle Filter Localization and Wheel Odometry

Code

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

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

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

    % 1. Initialize to prior likelihood of being occupied: Done above
    % 2. Loop over scans: Loop over all cells in field of view (we are in that
loop), update affected cells using equation slide 11 lec24
    % 3. Threshold cells based on whether they became more or less likely than
prior

    % Step (0a): Params for grid mapping algorithm: alpha: weight for occupied
cells, beta: weight for frre cells. TUNABLE PARAMS
    alpha = 2;
    beta = 1;

    % Step (0b): Start loop over each timestep, get current scans and continue
if scans are NaN (no data) or out of the valid range
    for j=1:npoints
        cur_scans = y_laser(i,j);
        cur_laser_angles = angles(j);
        if any(isnan(cur_scans)) || any(cur_scans < r_min_laser) ||
any(cur_scans > r_max_laser)
            continue;
        end

        % Step (1) First transform the laser scans to the robot frame, to get
laser endpoint in robot frame
        x_endpoint = x_interp(i) + cur_scans*cos(theta_interp(i) +
cur_laser_angles);
        y_endpoint = y_interp(i) + cur_scans*sin(theta_interp(i) +
cur_laser_angles);

        % Step (2) Convert to (integer) image coordinates:
        % convert both the robot position and the laser endpoint to image
coordinates, so we can get all cells between them.
        x_endpt_map = round((x_endpoint - ogxmin)/ogres);
        y_endpt_map = round((y_endpoint - ogymin)/ogres);
        x_robot_map = round((x_interp(i) - ogxmin)/ogres);
        y_robot_map = round((y_interp(i) - ogymin)/ogres);
        % We don't want to clip because we might classify the map bounds as
obstacles if we clipped lidar, just skip this iteration instead
        if x_endpt_map < 1 || x_endpt_map > ognx || y_endpt_map < 1 ||
```

```
y_endpt_map > ogny
            continue;
        end
        if x_robot_map < 1 || x_robot_map > ognx || y_robot_map < 1 ||
y_robot_map > ogny
            continue;
        end

        % Step (3) Get all cells along the line from the robot to the laser
endpoint
        % Easy (and inefficient) way: use linear interpolation: step in x and y
by some stepsize
        % Number of steps: approx. with the max of the difference in x and y.
This will step in the direction of most change
        num_steps = max(abs(x_endpt_map - x_robot_map), abs(y_endpt_map -
y_robot_map));
        x_step = (x_endpt_map - x_robot_map)/num_steps;
        y_step = (y_endpt_map - y_robot_map)/num_steps;
        % Can update the map with cells along the line in this loop: exclude
last entry, which is the endpoint
        for k=0:num_steps-1
            x_pixel = round(x_robot_map + k*x_step);
            y_pixel = round(y_robot_map + k*y_step);
            % If within bounds, Update the occupancy grid:
            if x_pixel >= 1 && x_pixel <= ognx && y_pixel >= 1 && y_pixel <=
ogny
                % If the cell is the endpoint, it is occupied
                if i == num_steps-1
                    oglo(y_pixel, x_pixel) = oglo(y_pixel, x_pixel) + alpha;
                % If the cell is not the endpoint, it is free
                else
                    oglo(y_pixel, x_pixel) = oglo(y_pixel, x_pixel) - beta;
                end
            end
        end


        % Step (4) Update the (log odds)occupancy grid:
        % The endpoint: occupied (hit an obstacle)
        % The cells along the line excluding end point: free (no obstacle)
        % Done above, while finding the cells along the line!

        % Step (5) At the end, convert log odds to probabilities
        ogp = exp(oglo)./(1 + exp(oglo));
```

```
    end
    % ------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;




% =========
% 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 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 ------

            % Step 1: Transform the scan to the world frame (SAME AS PART 1)
            % IMPORTANT: cur_scans: the measurement for this particle (distance
to closest obstacle)
            cur_scans = y_laser(i,j);
            cur_laser_angles = angles(j);
            if any(isnan(cur_scans)) || any(cur_scans < r_min_laser) ||
any(cur_scans > r_max_laser)
                continue;
            end
            x_endpoint = x_interp(i) + cur_scans*cos(theta_interp(i) +
cur_laser_angles);
            y_endpoint = y_interp(i) + cur_scans*sin(theta_interp(i) +
cur_laser_angles);

            % Step 2: Get the expected measurement for gaussian likelihood
reweighting
            % case a) we need to check if the endpoint is within the map. If it
isn't, instead of skipping here can take the measurement as the maxrange
            % case b) the endpoint is in the map but is not an obstacle: the
expected measurement is also the maxrange.
            % Can treat case a and b together
            % Get grid coordinates of the endpoint
            x_endpt_map = round((x_endpoint - ogxmin)/ogres);
            y_endpt_map = round((y_endpoint - ogymin)/ogres);
            % Check if inside:
            if x_endpt_map < 1 || x_endpt_map > ognx || y_endpt_map < 1 ||
y_endpt_map > ogny || ogp(y_endpt_map, x_endpt_map) < 0.5
                expected = y_laser_max;
            else
```

```
                % case c) In the map and obstacle was detected at scan
location:: the expected measurement is the euclidean distance from robot (i.e.
particle position) to the obstacle
                dx = x_endpoint - x_particle(n);
                dy = y_endpoint - y_particle(n);
                expected = sqrt(dx^2 + dy^2);
            end

            % Reweight by passing through Gaussian Likelihood: x = measurement,
mu = expected measurement. w_gain is the 2pi sigma of the gaussian
            w_particle(n) = w_particle(n) * exp(-0.5 * ((cur_scans -
expected)^2 / laser_var)) / w_gain;

            % ------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
```