ROB521: Mobile Robotics and Perception
Assignment #3: Lidar Mapping & Localization

Question 1: Code Occupancy Mapping Algorithm

The first question consisted of working on an occupancy grid mapping algorithm that builds a map of the environment from ground-truth localization. Since the timestamp interpolation errors have more of an effect when the robot turns quickly, laser scans were only used if the angular velocity was less than 0.1 rad/s. Moreover, laser points with NaN values were skipped. The laser endpoints indicate where the robot identifies an obstacle. They represent parts of the map that are occupied and some surrounding points are likely to be also occupied. Any points preceding that are likely unoccupied. After converting into coordinates in the inertial frame, lists for the occupancy grid in log-odds format and occupancy grid in probability format are built for plotting.

The map would likely be a lot more accurate if the robot stopped and rotated around the rooms instead of moving directly forward along its path. Nonetheless, due to the robot's movement and range of the sensor, Figure 1 is a good representation of the environment and includes key obstacles like the balls, shelves, etc.

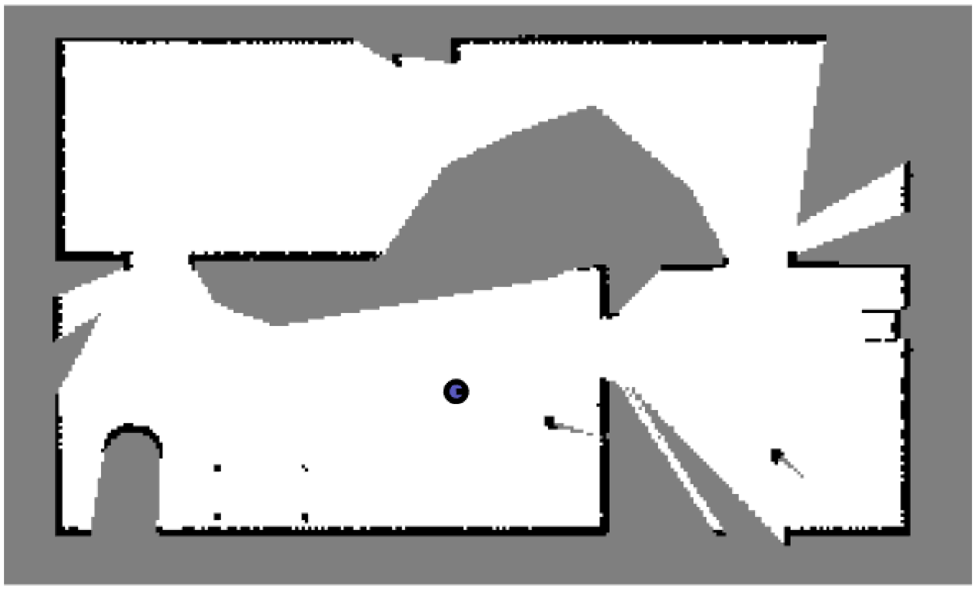

Figure 1: *Occupancy Grid Map*

Question 2: Localization from an Occupancy Grid Map using Particle Filter

The second part of the assignment focused on robot localization using a particle filter algorithm that makes use of laser rangefinder readings, wheel odometry, and the occupancy grid map from Question 1. Only 2 laser scan lines on the extreme left and right of the field of view were used. The laser readings are compared with readings that are expected (obtained using the ground truth positions) to update the particle weights. In terms of particle filter tuning parameters, only the number of particles was reduced to 100.

The localization is more accurate when the robot moves around areas with distinguishable obstacles. When the object is further away from obstacles (i.e. in open areas), small changes in position have less of an effect on the measurements obtained, preventing the particle filter method from being able to determine the robot's location accurately in open areas. Figure 2 shows the error between the estimate and true positions of the robot for the particle filter method versus the wheel odometry method. Evidently, errors for the particle filter method are much lower than using the odometry method. The wheel odometry method is a dead-reckoning method in which errors accumulate over time because of wheel slippage the encoders can't measure. Whereas, the errors from the particle filter approach look more steady and lower than errors from the wheel odometry method.

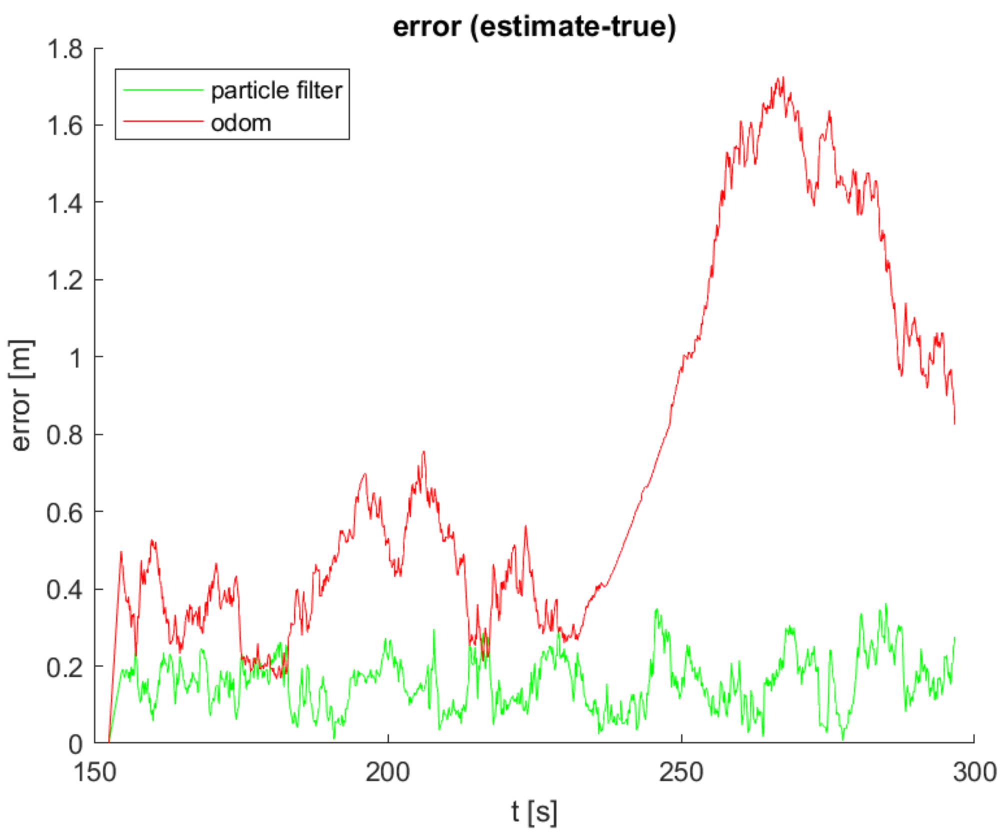

Figure 2: *Error from particle filter approach vs odometry approach*

```
% =========
% 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)
    % skip laser scans with angular velocity > 0.1
    if abs(omega_interp(i)) > 0.1
        continue
    end

    % ------insert your occupancy grid mapping algorithm here------
    % define rotation matrix and translation vector

    C = [cos(theta_interp(i)) -sin(theta_interp(i)) 0;
```

```
        sin(theta_interp(i)) cos(theta_interp(i)) 0;
        0 0 1];

    % translation
    r = [x_interp(i)-0.1*cos(theta_interp(i));
        y_interp(i)-0.1*sin(theta_interp(i));
        0];

    occupied = [];
    unoccupied = [];

    % loop over laser points
    for j=1:npoints
        % if value is NaN, then skip
        if isnan(y_laser(i, j))
            continue
        end

        % determine occupied and unoccipied points
        dist_points = round(y_laser(i, j)/ogres);
        for k=1:dist_points
            points = [k*ogres*cos(angles(j)); k*ogres*sin(angles(j)); 0];
            if k <= dist_points - 2
                unoccupied = [unoccupied points];
            else
                occupied = [occupied points];
            end
        end
    end

    % converting to intertial frame
    occupiedInertial = C*occupied + r;
    unoccupiedInertial = C*unoccupied + r;

    % fixing shift based on on ogxmin and ogymin and converted to
    % coordinates
    adding = [7; 3; 0];

    occupiedInertial = occupiedInertial + adding(:)*ones(1, size(occupiedInertial, 2));
    occupiedInertial = round(occupiedInertial / ogres);
    occupiedIndex = sub2ind(size(ogp), occupiedInertial(2, :), occupiedInertial(1, :));

    unoccupiedInertial = unoccupiedInertial + adding(:)*ones(1, size↙
(unoccupiedInertial, 2));
    unoccupiedInertial = round(unoccupiedInertial / ogres);
    unoccupiedIndex = sub2ind(size(ogp), unoccupiedInertial(2, :), unoccupiedInertial↙
(1, :));

    % building occupancy grid in log-odds format and occupancy
    % grid in probability format
    oglo(occupiedIndex) = ogp(occupiedIndex) + 100;
    oglo(unoccupiedIndex) = ogp(unoccupiedIndex) - 100;

    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;
```

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

```matlab
%   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 = 100;       % 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 ------

            % skip over NAN values
            if isnan(y_laser(i,j))
                continue
            end

            rayMeasures = y_laser_max/ogres+1;

            % convert into frame
            C = [cos(angles(j)), -sin(angles(j)); sin(angles(j)) cos(angles(j))];

            a = 0:ogres:y_laser_max;
            rayLidar = [a; zeros(1, rayMeasures)];
            rayLidar = C*rayLidar;

            % inertial/initial frame
            rayLidar = T*[rayLidar; ones(1, rayMeasures)];

            % coordinates
            rayOccupancyGrid = round((rayLidar(1:2, :)- ...
                [ogxmin; ogymin])./ ogres);

            % determining valid coordinates
            validCoor = rayOccupancyGrid(1, :) <= ognx & rayOccupancyGrid(1, :) > 0 ...
                & rayOccupancyGrid(2, :) <= ogny & rayOccupancyGrid(2, :) > 0;
            rayOccupancyGrid = rayOccupancyGrid(:, validCoor);

            % occupancy grid in probability
            coordinates = sub2ind([ogny, ognx], rayOccupancyGrid(2, :), ...
                rayOccupancyGrid(1, :));
            ogpRay = ogp(coordinates);

            % finding obstacle
            obstacleCoorIndex = find(ogpRay > 0.5, 1);
            obstacle = rayOccupancyGrid(:, obstacleCoorIndex);

            % checking is empty
            if ~isempty(obstacle)
                % coor in world
                obstacle = [ogxmin; ogymin] + (obstacle.*ogres);

                % particle weights using gaussian for probability
                est = sqrt((x_particle(n) - obstacle(1))^2 + ...
                    (y_particle(n) - obstacle(2))^2);
                w_particle(n) = w_particle(n)*pdf('Normal', ...
                    abs(est - y_laser(i,j)), 0, laser_var)*w_gain;
            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;
    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
```