ASSIGNMENT 2

ROB521: Mobile Robotics and Perception

Jojo (Yizhi) Zhou, 1003002396

February 3, 2020

1 Occupancy Grid

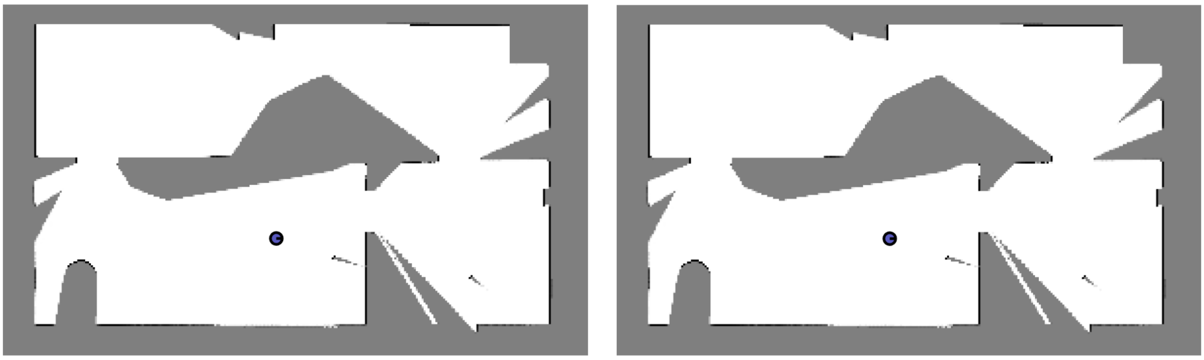

Figure 1: $\alpha = 0.1, \beta = 1$. Grids updated with absolute value (left) vs. trend (right)

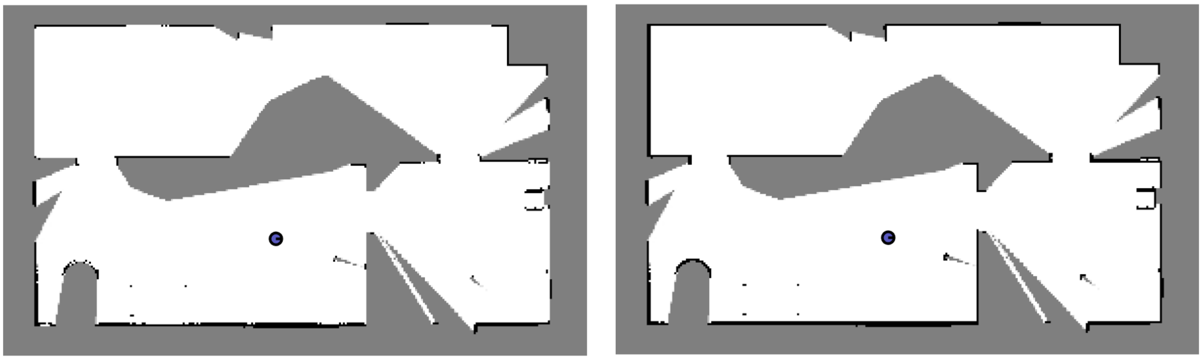

Figure 2: $\alpha = 10, \beta = 1$. Grids updated with absolute value (left) vs. trend (right)

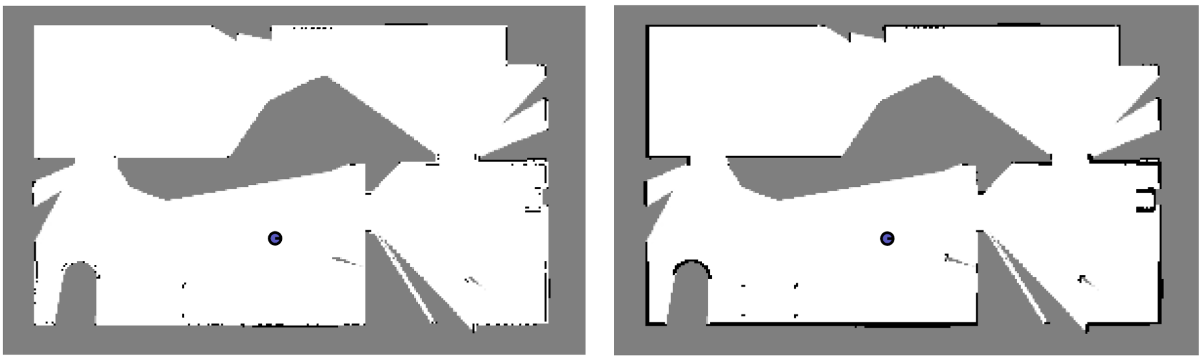

Figure 3: $\alpha = 100, \beta = 1$. Grids updated with absolute value (left) vs. trend (right)

The occupancy grids are iteratively updated by the laser scans at each time step. Each laser beam is divided into segments roughly the length of the grids, ranging from the closest to the furthest from the robot:

- The furthest laser segment is the obstacle, which blocked the laser beam from reaching further. We add α to the corresponding grid's log-odd value, indicating an increased probability that it is occupied.
- Conversely, all the other segments correspond to free space, and we therefore subtract β from their log-odd values.

While the algorithm is very straightforward, I tuned the α, β values and compared the results (fig.1, 2, 3). We can see that,

- In term of $\alpha : \beta$ ratio:
  - A smaller $\alpha : \beta$ ratio (fig.1) allows the free-space readings to easily overwrite the previous occupancy reading, resulting in missing walls and obstacles - in fact, the four smaller dots in the bottom-left corner are completely gone.
  - Conversely, a large $\alpha : \beta$ (fig.3) also results in missing walls, but instead of simply overwriting the probability of occupancy, the behaviour is more inconsistent.
- In term of updating heuristic:
  - The left ones update the grid values according to the absolute probability values at each time step, while the right ones' heuristic also requires a tendency. That is, for a grid to be marked occupied, its probability needs to be not only $> 50\%$, but also larger than its previous probability (i.e., from the last time step), and vice versa.
  - We can see, especially from fig.2 and 3, that using the heuristic with both absolute and relative probability results in a much more consistent result. For example, the walls are less fuzzy.
  - That being said, the heuristic can't solve the problems from an inappropriate $\alpha : \beta$ ratio (i.e., walls are still missing in fig.3).

I ended up choosing $\alpha : \beta = 10 : 1$ and the heuristic with both absolute probability and tendency, which is shown in fig.4. To further improve the map accuracy, we could simply increase the grid resolution and let the robot explore more, as currently there are apparently some areas that haven't been coved by the laser scans.

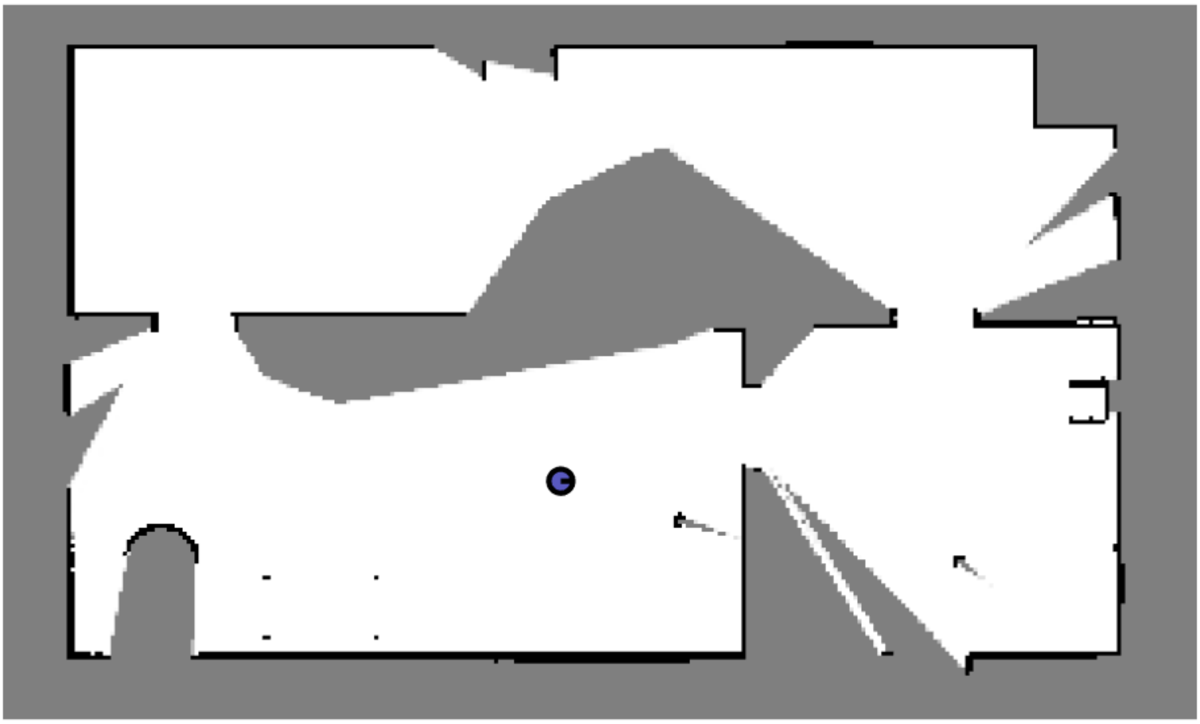

Figure 4: Final output plot from Question 1

2 Noisy Wheel Odometry

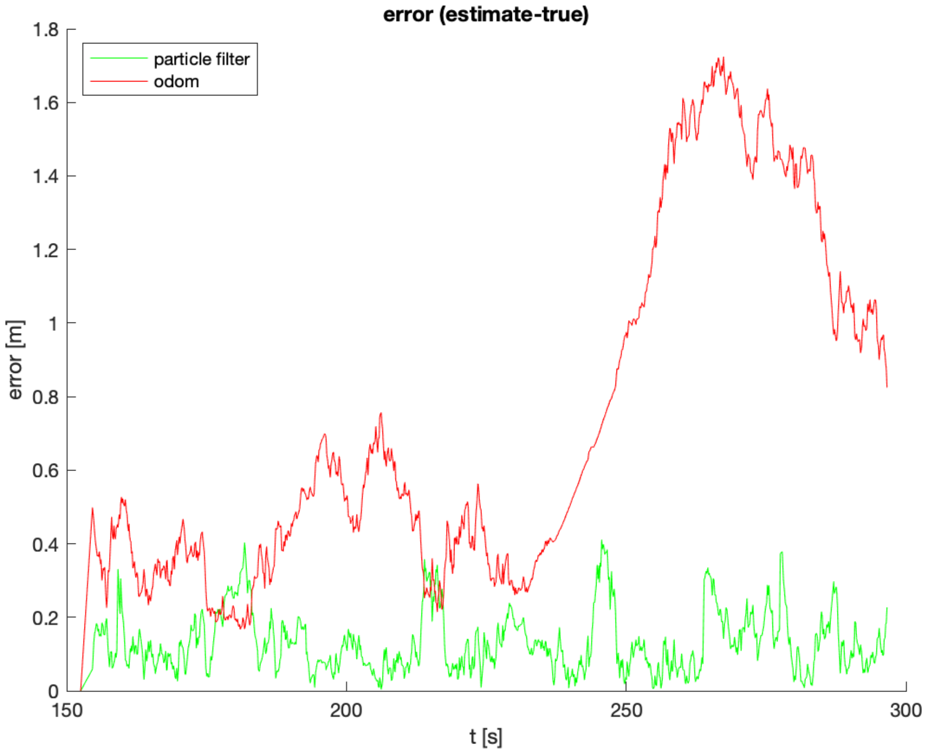

Figure 5: $\alpha = 10, \beta = 1$ with posterior update

In the beginning, the odometry estimate is slightly better the particle filter, as there is little drift in the odometry so far. Near the end of trajectory, the particle filter has trouble estimating the once again, though this time it is significantly lower than the odometry data. This is expected, the divergence of the odometry pose from the true path grows over distance travel. The growth in error for certain portions likely has to do with the ambiguity of the rooms and the limits of the laser, which caps at 5 meters. With similar features and only a single range value, it is easy for erroneous particles to have high weights. Perhaps with the full 640 angles of laser data, the particle filter will do much better.

The solution error is significantly lower in the beginning and end of the plot; it appears the solution model is more robust to changes in the location compared to my code. As I am using the same parameters, provided by the model, the main difference should lie in the observation model. I estimated the expected sensor data from the particle using a ray extending from the laser scan pose, incrementally increasing the length of the ray and checking for obstacles.

Appendix I: Code for Occupancy Grid

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

```matlab
% 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);

% Other variables and quantities
cos_angles = cos(angles);
sin_angles = sin(angles);
one = ones(1, npoints);     % Just for homogenroes transformatiom
alpha = 10;                 % log-odd of occupied cell (+)
beta = 1;                   % log-odd of occupied free cell (-)

```

```
% 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

    % Step 0. Preprocess the laser scans
    y_laser_t = y_laser(i,:);               % Get the laser scans at time t
    y_laser_t(y_laser_t<r_min_laser) = NaN; % Remove all invalid values from the laser scan
    y_laser_t(y_laser_t>r_max_laser) = NaN;

    % Step 1. Transform frames from laser -> robot (v) -> inertial (i) -> grid (g)
    % 1.1 laser -> robot (vehicle)
    x_v = y_laser_t .* cos_angles - 0.1;    % Note laser is 10 cm behind origin of frame v's origin
    y_v = y_laser_t .* sin_angles;

    % 1.2 vehicle -> inertial
    robot_sin_i = sin(theta_interp(i));     % Vehecle's current position/orientation in inertial frame
    robot_cos_i = cos(theta_interp(i));     % Convention: x_,y_ for laser scanned "obstacle"
    robot_x_i = x_interp(i);                %             robot_ for vehicle pose
    robot_y_i = y_interp(i);
    H_iv = [robot_cos_i -robot_sin_i robot_x_i; % Homogenous matrix defining the transformation
            robot_sin_i robot_cos_i  robot_y_i;
            0           0            1];
    xy_i = H_iv*[x_v; y_v; one];

    % 1.3 inertial -> grid (i.e., distance -> grid index)
    robot_x_g = (robot_x_i-ogxmin)/ogres;   % Scalars
    robot_y_g = (robot_y_i-ogymin)/ogres;
    x_g = (xy_i(1,:)-ogxmin)/ogres;         % Arrays
    y_g = (xy_i(2,:)-ogymin)/ogres;

    % Step 2. Update the grid using ray tracking of each (j-th) laser ray, in log-odds form
    for j = 1:npoints
        if isnan(x_g(j)) || isnan(y_g(j))   % Only use valid measurements
            continue;
        end

        % 2.1 Break laser into segments (in x, y grids)
```

```
        nsegments = round(y_laser_t(j)/ogres);
        segs_x_g = round(linspace(robot_x_g, x_g(j), nsegments));
        segs_y_g = round(linspace(robot_y_g, y_g(j), nsegments));

        % 2.2 Mark all segments up till the scanned obstacle as free (-beta)
        %     and the obstacle as occupied (+alpha)
        for k = 1:nsegments-1
            oglo(segs_y_g(k), segs_x_g(k)) = oglo(segs_y_g(k), segs_x_g(k)) - beta;
        end
        oglo(segs_y_g(nsegments), segs_x_g(nsegments)) = oglo(segs_y_g(nsegments), segs_x_g(nsegments)) + alpha;
    end

    % Step 3. Threshold the grids based on tendency (posterior vs. prior), in probability form
    %         Only update those with increasing likelihood
%     ogp = exp(oglo)./(1+exp(oglo));
    ogp_post = exp(oglo)./(1+exp(oglo));
    for y = 1:ogny
        for x = 1:ognx
            if ogp_post(y,x)>ogp(y,x) && ogp_post(y,x)>=0.5     % Occupieed
                ogp(y,x) = ogp_post(y,x);
            elseif ogp_post(y,x)<ogp(y,x) && ogp_post(y,x)<0.5  % Free
                ogp(y,x) = ogp_post(y,x);
            end
        end
    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;
```

Appendix II: Code for Particle Filtering

```
% =========
% 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
ogp_threshold = 0.5;    % minimum probability reuired to declare that the grid is occupied

% 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
```

```matlab

        % 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 0. Rule out invalid scans and, if valid, prepare some variables
        laser_dist_measured = y_laser(i, j);
        if isnan(laser_dist_measured) || laser_dist_measured > y_laser_max || laser_dist_measured < r_min_laser
            continue
        end
        laser_dist_segs = r_min_laser:ogres:laser_dist_measured;        % Breaks laser into segments of grid size
        x_particle_g = x_particle(n)-ogxmin;
        y_particle_g = y_particle(n)-ogymin;
        T_lg = T(1:2, 1:2) * [cos(angles(j)); sin(angles(j))];          % Transformation from laser dist to inertial frame

        % Step 1. Step from the pasticle to its furthest laser range and find the laser range w/ pose estimate
        for k = 1:length(laser_dist_segs)
            laser_dist_estimated = laser_dist_segs(k);
            likelihood = 1.0;

            % 1.1 Estimate laser distance based on particle position
            laser_xy = T_lg * laser_dist_estimated;  % Transform from laser -> robot -> inertial -> grid
            x_g = round((x_particle_g + laser_xy(1))/ogres);            % Estimated obstacle location
            y_g = round((y_particle_g + laser_xy(2))/ogres);            %   given pose == particle && range = laser_dist_segs(k)
            if x_g > size(ogp,2) || y_g > size(ogp,1) || isnan(x_g) || isnan(y_g) || x_g<1 || y_g<1
                break                                                   % Stop if going out of map
            end

            % 1.2 Weight = (Gaussian) likelihood of measurement @ t, given map and current pose estimate (of particle)
            if ogp(y_g, x_g) >= ogp_threshold                           % If grid is occupied => estimated dist is right there
                likelihood = normpdf(laser_dist_measured, laser_dist_estimated, laser_var);
```

```
                break
            end
        end

        % Step 3. Update weight - multiplying it by each beam's weight gain
        w_particle(n) = w_particle(n) * likelihood * 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
```