

# ASSIGNMENT 2

---

ROB521: Mobile Robotics and Perception  
Jojo (Yizhi) Zhou, 1003002396

February 3, 2020

# 1 Occupancy Grid

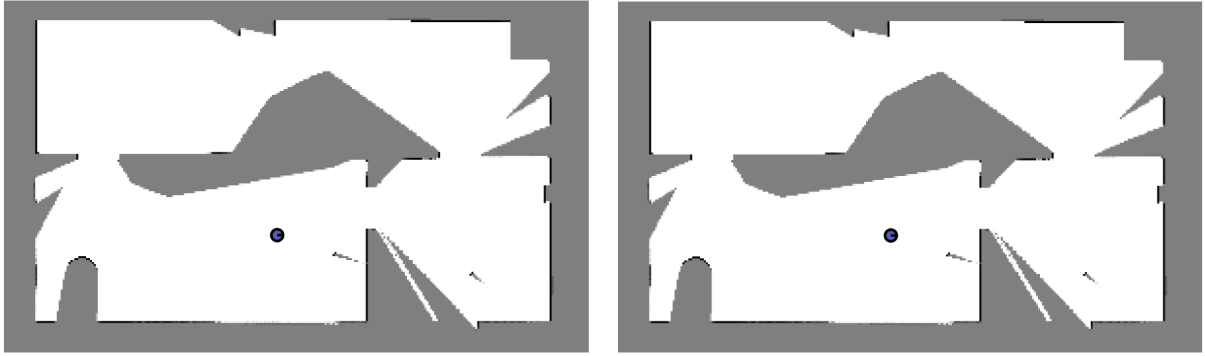


Figure 1:  $\alpha = 0.1, \beta = 1$ . Grids updated with absolute value (left) vs. trend (right)



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

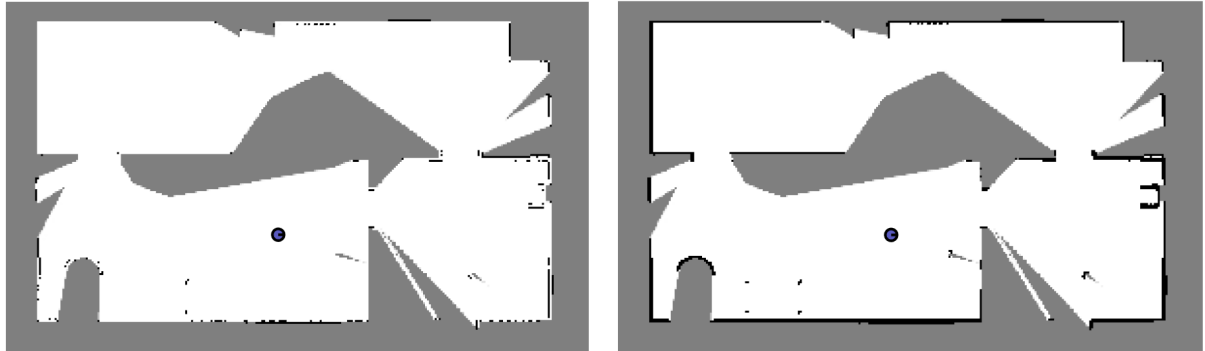


Figure 3:  $\alpha = 100, \beta = 1$ . Grids updated with absolute value (left) vs. trend (right)

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  $\alpha$  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  $\beta$  from their log-odd values.

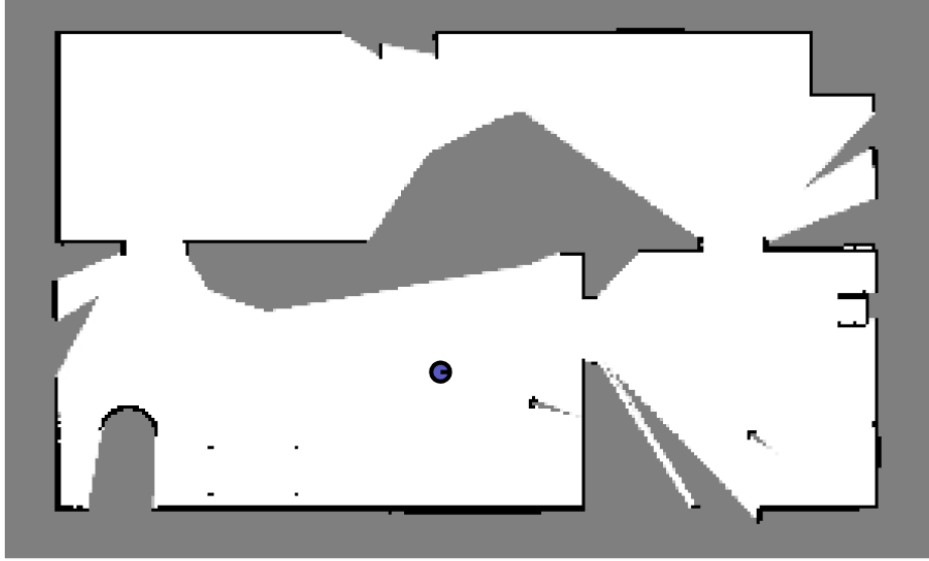


Figure 4: Output plot from to Question 1

While the algorithm is very straightforward, I tuned the  $\alpha, \beta$  values and compared the results (fig.123). We can see that a smaller  $\alpha : \beta$  ratio (fig.1) allows the free-space readings to easily override the previous occupancy reading, resulting in missing walls and obstacles. Conversely, a  $\alpha : \beta$  The Picking a higher resolution for the rays, which are used to update the grid, reduced raggedness, but it decreased the number of walls in the map. Another issue that occurred is picking the correct alpha and beta, which is causing the missing walls in some parts of the map. A small alpha is easily overwritten by the beta values from the newer updates; a very large alpha has very inconsistent behavior, leaving large portions of the map without walls. Overall, the map is pretty accurate.

## 2 Noisy Wheel Odometry

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,

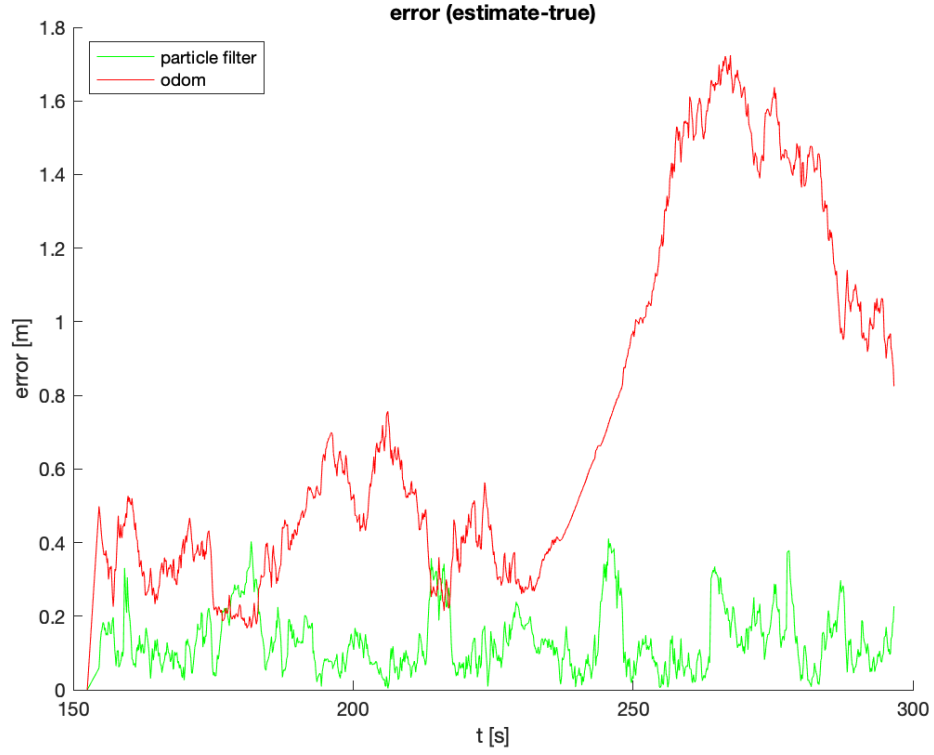


Figure 5:  $\alpha = 10, \beta = 1$  with posterior update

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

```
1 % =====
2 % ass2_q1.m
3 % =====
4 %
5 % This assignment will introduce you to the idea of first building an
6 % occupancy grid then using that grid to estimate a robot's motion using a
7 % particle filter.
8 %
9 % There are two questions to complete (5 marks each):
10 %
11 %     Question 1: code occupancy mapping algorithm
12 %     Question 2: see ass2_q2.m
13 %
14 % Fill in the required sections of this script with your code, run it to
15 % generate the requested plot/movie, then paste the plots into a short report
16 % that includes a few comments about what you've observed. Append your
17 % version of this script to the report. Hand in the report as a PDF file
18 % and the two resulting AVI files from Questions 1 and 2.
19 %
20 % requires: basic Matlab, 'gazebo.mat'
21 %
22 % T D Barfoot, January 2016
23 %
24 clear all;
25
26 % set random seed for repeatability
27 rng(1);
28
29 % =====
30 % load the dataset from file
31 % =====
32 %
33 %     ground truth poses: t_true x_true y_true theta_true
34 %     odometry measurements: t_odom v_odom omega_odom
35 %     laser scans: t_laser y_laser
36 %     laser range limits: r_min_laser r_max_laser
37 %     laser angle limits: phi_min_laser phi_max_laser
38 %
39 load gazebo.mat;
40
41 % =====
42 % Question 1: build an occupancy grid map
43 % =====
44 %
45 % Write an occupancy grid mapping algorithm that builds the map from the
46 % perfect ground-truth localization. Some of the setup is done for you
47 % below. The resulting map should look like "ass2_q1_soln.png". You can
```

```

48 % watch the movie "ass2_q1_soln.mp4" to see what the entire mapping process
49 % should look like. At the end you will save your occupancy grid map to
50 % the file "occmap.mat" for use in Question 2 of this assignment.
51
52 % allocate a big 2D array for the occupancy grid
53 ogres = 0.05; % resolution of occ grid
54 ogxmin = -7; % minimum x value
55 ogxmax = 8; % maximum x value
56 ogymmin = -3; % minimum y value
57 ogymax = 6; % maximum y value
58 ognx = (ogxmax-ogxmin)/ogres; % number of cells in x direction
59 ogny = (ogymax-ogymmin)/ogres; % number of cells in y direction
60 oglo = zeros(ogny,ognx); % occupancy grid in log-odds format
61 ogp = zeros(ogny,ognx); % occupancy grid in probability format
62
63 % precalculate some quantities
64 numodom = size(t_odom,1);
65 npoints = size(y_laser,2);
66 angles = linspace(phi_min_laser, phi_max_laser, npoints);
67 dx = ogres*cos(angles);
68 dy = ogres*sin(angles);
69
70 % interpolate the noise-free ground-truth at the laser timestamps
71 t_interp = linspace(t_true(1), t_true(numodom), numodom);
72 x_interp = interp1(t_interp, x_true, t_laser);
73 y_interp = interp1(t_interp, y_true, t_laser);
74 theta_interp = interp1(t_interp, theta_true, t_laser);
75 omega_interp = interp1(t_interp, omega_odom, t_laser);
76
77 % set up the plotting/movie recording
78 vid = VideoWriter('ass2_q1.avi');
79 open(vid);
80 figure(1);
81 clf;
82 pcolor(ogp);
83 colormap(1-gray);
84 shading('flat');
85 axis equal;
86 axis off;
87 M = getframe;
88 writeVideo(vid,M);
89
90 % Other variables and quantities
91 cos_angles = cos(angles);
92 sin_angles = sin(angles);
93 one = ones(1, npoints); % Just for homogenroes transformation
94 alpha = 10; % log-odd of occupied cell (+)
95 beta = 1; % log-odd of occupied free cell (-)
96

```

```

97 % loop over laser scans (every fifth)
98 for i=1:5:size(t_laser,1)
99
100 % -----insert your occupancy grid mapping algorithm here-----
101 %     if abs(omega_interp(i)) < 0.1
102 %         continue;
103 %     end
104
105 % Step 0. Preprocess the laser scans
106 y_laser_t = y_laser(i,:); % Get the laser scans at time t
107 y_laser_t(y_laser_t < r_min_laser) = NaN; % Remove all invalid values from
the laser scan
108 y_laser_t(y_laser_t > r_max_laser) = NaN;
109
110 % Step 1. Transform frames from laser -> robot (v) -> inertial (i) -> grid
(g)
111 % 1.1 laser -> robot (vehicle)
112 x_v = y_laser_t .* cos_angles - 0.1; % Note laser is 10 cm behind origin
of frame v's origin
113 y_v = y_laser_t .* sin_angles;
114
115 % 1.2 vehicle -> inertial
116 robot_sin_i = sin(theta_interp(i)); % Vehicle's current position/
orientation in inertial frame
117 robot_cos_i = cos(theta_interp(i)); % Convention: x_,y_ for laser
scanned "obstacle"
118 robot_x_i = x_interp(i); % robot_ for vehicle
pose
119 robot_y_i = y_interp(i);
120 H_iv = [robot_cos_i -robot_sin_i robot_x_i; % Homogenous matrix defining
the transformation
121         robot_sin_i robot_cos_i robot_y_i;
122         0 0 1];
123 xy_i = H_iv*[x_v; y_v; one];
124
125 % 1.3 inertial -> grid (i.e., distance -> grid index)
126 robot_x_g = (robot_x_i-ogxmin)/ogres; % Scalars
127 robot_y_g = (robot_y_i-ogymin)/ogres;
128 x_g = (xy_i(1,:)-ogxmin)/ogres; % Arrays
129 y_g = (xy_i(2,:)-ogymin)/ogres;
130
131 % Step 2. Update the grid using ray tracking of each (j-th) laser ray, in
log-odds form
132 for j = 1:npoints
133     if isnan(x_g(j)) || isnan(y_g(j)) % Only use valid measurements
134         continue;
135     end
136
137 % 2.1 Break laser into segments (in x, y grids)

```

```

138     nsegments = round(y_laser_t(j)/ogres);
139     segs_x_g = round(linspace(robot_x_g, x_g(j), nsegments));
140     segs_y_g = round(linspace(robot_y_g, y_g(j), nsegments));
141
142     % 2.2 Mark all segments up till the scanned obstacle as free (-beta)
143     %     and the obstacle as occupied (+alpha)
144     for k = 1:nsegments-1
145         oglo(segs_y_g(k), segs_x_g(k)) = oglo(segs_y_g(k), segs_x_g(k)) -
146         beta;
147     end
148     oglo(segs_y_g(nsegments), segs_x_g(nsegments)) = oglo(segs_y_g(
149     nsegments), segs_x_g(nsegments)) + alpha;
150 end
151
152 % Step 3. Threshold the grids based on tendency (posterior vs. prior), in
153 probability form
154 %     Only update those with increasing likelihood
155 % ogp = exp(oglo)./(1+exp(oglo));
156 ogp_post = exp(oglo)./(1+exp(oglo));
157 for y = 1:ogny
158     for x = 1:ognx
159         if ogp_post(y,x)>ogp(y,x) && ogp_post(y,x)>=0.5 % Occupieed
160             ogp(y,x) = ogp_post(y,x);
161         elseif ogp_post(y,x)<ogp(y,x) && ogp_post(y,x)<0.5 % Free
162             ogp(y,x) = ogp_post(y,x);
163         end
164     end
165 end
166
167 % -----end of your occupancy grid mapping algorithm-----
168
169 % draw the map
170 clf;
171 pcolor(ogp);
172 colormap(1-gray);
173 shading('flat');
174 axis equal;
175 axis off;
176
177 % draw the robot
178 hold on;
179 x = (x_interp(i)-ogxmin)/ogres;
180 y = (y_interp(i)-ogymin)/ogres;
181 th = theta_interp(i);
182 r = 0.15/ogres;
183 set(rectangle( 'Position', [x-r y-r 2*r 2*r], 'Curvature', [1 1]), '
184 LineWidth',2,'FaceColor',[0.35 0.35 0.75]);
185 set(plot([x x+r*cos(th)], [y y+r*sin(th)], 'k-'),'LineWidth',2);
186
187 % save the video frame

```



```
183     M = getframe;  
184     writeVideo(vid,M);  
185  
186     pause(0.1);  
187  
188 end  
189  
190 close(vid);  
191 print -dpng ass2_q1.png  
192  
193 save occmap.mat ogres ogxmin ogxmax ogymin ogymax ognx ogny oglo ogp;
```

## Appendix II: Code for Particle Filtering

```
1 % =====
2 % ass2_q2.m
3 % =====
4 %
5 % This assignment will introduce you to the idea of first building an
6 % occupancy grid then using that grid to estimate a robot's motion using a
7 % particle filter.
8 %
9 % There are three questions to complete (5 marks each):
10 %
11 %     Question 1: see ass2_q1.m
12 %     Question 2: code particle filter to localize from known map
13 %
14 % Fill in the required sections of this script with your code, run it to
15 % generate the requested plot/movie, then paste the plots into a short report
16 % that includes a few comments about what you've observed. Append your
17 % version of this script to the report. Hand in the report as a PDF file
18 % and the two resulting AVI files from Questions 1 and 2.
19 %
20 % requires: basic Matlab, 'gazebo.mat', 'occmap.mat'
21 %
22 % T D Barfoot, January 2016
23 %
24 clear all;
25
26 % set random seed for repeatability
27 rng(1);
28
29 % =====
30 % load the dataset from file
31 % =====
32 %
33 %     ground truth poses: t_true x_true y_true theta_true
34 %     odometry measurements: t_odom v_odom omega_odom
35 %     laser scans: t_laser y_laser
36 %     laser range limits: r_min_laser r_max_laser
37 %     laser angle limits: phi_min_laser phi_max_laser
38 %
39 load gazebo.mat;
40
41 % =====
42 % load the occupancy map from question 1 from file
43 % =====
44 % ogres: resolution of occ grid
45 % ogxmin: minimum x value
46 % ogxmax: maximum x value
47 % ogymin: minimum y value
```

```

48 % ogymax: maximum y value
49 % ognx: number of cells in x direction
50 % ogny: number of cells in y direction
51 % oglo: occupancy grid in log-odds format
52 % ogp: occupancy grid in probability format
53 load occmap.mat;
54
55 % =====
56 % Question 2: localization from an occupancy grid map using particle filter
57 % =====
58 %
59 % Write a particle filter localization algorithm to localize from the laser
60 % rangefinder readings, wheel odometry, and the occupancy grid map you
61 % built in Question 1. We will only use two laser scan lines at the
62 % extreme left and right of the field of view, to demonstrate that the
63 % algorithm does not need a lot of information to localize fairly well. To
64 % make the problem harder, the below lines add noise to the wheel odometry
65 % and to the laser scans. You can watch the movie "ass2_q2_soln.mp4" to
66 % see what the results should look like. The plot "ass2_q2_soln.png" shows
67 % the errors in the estimates produced by wheel odometry alone and by the
68 % particle filter look like as compared to ground truth; we can see that
69 % the errors are much lower when we use the particle filter.
70
71 % interpolate the noise-free ground-truth at the laser timestamps
72 numodom = size(t_odom,1);
73 t_interp = linspace(t_true(1),t_true(numodom),numodom);
74 x_interp = interp1(t_interp,x_true,t_laser);
75 y_interp = interp1(t_interp,y_true,t_laser);
76 theta_interp = interp1(t_interp,theta_true,t_laser);
77 omega_interp = interp1(t_interp,omega_odom,t_laser);
78
79 % interpolate the wheel odometry at the laser timestamps and
80 % add noise to measurements (yes, on purpose to see effect)
81 v_interp = interp1(t_interp,v_odom,t_laser) + 0.2*randn(size(t_laser,1),1);
82 omega_interp = interp1(t_interp,omega_odom,t_laser) + 0.04*randn(size(t_laser
    ,1),1);
83
84 % add noise to the laser range measurements (yes, on purpose to see effect)
85 % and precompute some quantities useful to the laser
86 y_laser = y_laser + 0.1*randn(size(y_laser));
87 npoints = size(y_laser,2);
88 angles = linspace(phi_min_laser, phi_max_laser,npoints);
89 dx = ogres*cos(angles);
90 dy = ogres*sin(angles);
91 y_laser_max = 5; % don't use laser measurements beyond this distance
92
93 % particle filter tuning parameters (yours may be different)
94 nparticles = 200; % number of particles
95 v_noise = 0.2; % noise on longitudinal speed for propagating particle

```

```

96 u_noise = 0.2;           % noise on lateral speed for propagating particle
97 omega_noise = 0.04;      % noise on rotational speed for propagating particle
98 laser_var = 0.5^2;       % variance on laser range distribution
99 w_gain = 10*sqrt( 2 * pi * laser_var );    % gain on particle weight
100 ogp_threshold = 0.5;    % minimum probability required to declare that the grid
    is occupied
101
102 % generate an initial cloud of particles
103 x_particle = x_true(1) + 0.5*randn(nparticles,1);
104 y_particle = y_true(1) + 0.3*randn(nparticles,1);
105 theta_particle = theta_true(1) + 0.1*randn(nparticles,1);
106
107 % compute a wheel odometry only estimate for comparison to particle
108 % filter
109 x_odom_only = x_true(1);
110 y_odom_only = y_true(1);
111 theta_odom_only = theta_true(1);
112
113 % error variables for final error plots - set the errors to zero at the start
114 pf_err(1) = 0;
115 wo_err(1) = 0;
116
117 % set up the plotting/movie recording
118 vid = VideoWriter('ass2_q2.avi');
119 open(vid);
120 figure(2);
121 clf;
122 hold on;
123 pcolor(ogp);
124 set(plot( (x_particle-ogxmin)/ogres, (y_particle-ogymin)/ogres, 'g.' ),'
    MarkerSize',10,'Color',[0 0.6 0]);
125 set(plot( (x_odom_only-ogxmin)/ogres, (y_odom_only-ogymin)/ogres, 'r.' ),'
    MarkerSize',20);
126 x = (x_interp(1)-ogxmin)/ogres;
127 y = (y_interp(1)-ogymin)/ogres;
128 th = theta_interp(1);
129 r = 0.15/ogres;
130 set(rectangle( 'Position', [x-r y-r 2*r 2*r], 'Curvature', [1 1]),'LineWidth'
    ,2,'FaceColor',[0.35 0.35 0.75]);
131 set(plot([x x+r*cos(th)]', [y y+r*sin(th)]', 'k-'),'LineWidth',2);
132 set(plot( (mean(x_particle)-ogxmin)/ogres, (mean(y_particle)-ogymin)/ogres, 'g.
    ' ),'MarkerSize',20);
133 colormap(1-gray);
134 shading('flat');
135 axis equal;
136 axis off;
137 M = getframe;
138 writeVideo(vid,M);
139

```

```

140 % loop over laser scans
141 for i=2:size(t_laser,1)
142
143     % update the wheel-odometry-only algorithm
144     dt = t_laser(i) - t_laser(i-1);
145     v = v_interp(i);
146     omega = omega_interp(i);
147     x_odom_only = x_odom_only + dt*v*cos( theta_odom_only );
148     y_odom_only = y_odom_only + dt*v*sin( theta_odom_only );
149     phi = theta_odom_only + dt*omega;
150     while phi > pi
151         phi = phi - 2*pi;
152     end
153     while phi < -pi
154         phi = phi + 2*pi;
155     end
156     theta_odom_only = phi;
157
158     % loop over the particles
159     for n=1:nparticles
160
161         % propagate the particle forward in time using wheel odometry
162         % (remember to add some unique noise to each particle so they
163         % spread out over time)
164         v = v_interp(i) + v_noise*randn(1);
165         u = u_noise*randn(1);
166         omega = omega_interp(i) + omega_noise*randn(1);
167         x_particle(n) = x_particle(n) + dt*(v*cos( theta_particle(n) ) - u*sin(
168         theta_particle(n) ));
169         y_particle(n) = y_particle(n) + dt*(v*sin( theta_particle(n) ) + u*cos(
170         theta_particle(n) ));
171         phi = theta_particle(n) + dt*omega;
172         while phi > pi
173             phi = phi - 2*pi;
174         end
175         while phi < -pi
176             phi = phi + 2*pi;
177         end
178         theta_particle(n) = phi;
179
180         % pose of particle in initial frame
181         T = [cos(theta_particle(n)) -sin(theta_particle(n)) x_particle(n); ...
182             sin(theta_particle(n))  cos(theta_particle(n)) y_particle(n); ...
183             0 0 1];
184
185         % compute the weight for each particle using only 2 laser rays
186         % (right=beam 1 and left=beam 640)
187         w_particle(n) = 1.0;
188         for beam=1:2

```

```

187
188     % we will only use the first and last laser ray for
189     % localization
190     if beam==1 % rightmost beam
191         j = 1;
192     elseif beam==2 % leftmost beam
193         j = 640;
194     end
195
196     % -----insert your particle filter weight calculation here -----\
197     % Step 0. Rule out invalid scans and, if valid, prepare some
variables
198     laser_dist_measured = y_laser(i, j);
199     if isnan(laser_dist_measured) || laser_dist_measured > y_laser_max
|| laser_dist_measured < r_min_laser
200         continue
201     end
202     laser_dist_segs = r_min_laser:ogres:laser_dist_measured; %
Breaks laser into segments of grid size
203     x_particle_g = x_particle(n)-ogxmin;
204     y_particle_g = y_particle(n)-ogymin;
205     T_lg = T(1:2, 1:2) * [cos(angles(j)); sin(angles(j))]; %
Transformation from laser dist to inertial frame
206
207     % Step 1. Step from the pasticle to its furthest laser range and
find the laser range w/ pose estimate
208     for k = 1:length(laser_dist_segs)
209         laser_dist_estimated = laser_dist_segs(k);
210         likelihood = 1.0;
211
212         % 1.1 Estimate laser distance based on particle position
213         laser_xy = T_lg * laser_dist_estimated; % Transform from laser
-> robot -> inertial -> grid
214         x_g = round((x_particle_g + laser_xy(1))/ogres); %
Estimated obstacle location
215         y_g = round((y_particle_g + laser_xy(2))/ogres); %
given pose == particle && range = laser_dist_segs(k)
216         if x_g > size(ogp,2) || y_g > size(ogp,1) || isnan(x_g) ||
isnan(y_g) || x_g<1 || y_g<1
217             break %
Stop if going out of map
218         end
219
220         % 1.2 Weight = (Gaussian) likelihood of measurement @ t, given
map and current pose estimate (of particle)
221         if ogp(y_g, x_g) >= ogp_threshold %
If grid is occupied => estimated dist is right there
222             likelihood = normpdf(laser_dist_measured,
laser_dist_estimated, laser_var);

```

```

223         break
224     end
225 end
226
227 % Step 3. Update weight - multiplying it by each beam's weight gain
228 w_particle(n) = w_particle(n) * likelihood * w_gain;
229 % -----end of your particle filter weight calculation-----
230 end
231
232 end
233
234 % resample the particles using Madow systematic resampling
235 w_bounds = cumsum(w_particle)/sum(w_particle);
236 w_target = rand(1);
237 j = 1;
238 for n=1:nparticles
239     while w_bounds(j) < w_target
240         j = mod(j,nparticles) + 1;
241     end
242     x_particle_new(n) = x_particle(j);
243     y_particle_new(n) = y_particle(j);
244     theta_particle_new(n) = theta_particle(j);
245     w_target = w_target + 1/nparticles;
246     if w_target > 1
247         w_target = w_target - 1.0;
248         j = 1;
249     end
250 end
251 x_particle = x_particle_new;
252 y_particle = y_particle_new;
253 theta_particle = theta_particle_new;
254
255 % save the translational error for later plotting
256 pf_err(i) = sqrt( (mean(x_particle) - x_interp(i))^2 + (mean(y_particle) -
y_interp(i))^2 );
257 wo_err(i) = sqrt( (x_odom_only - x_interp(i))^2 + (y_odom_only - y_interp(i)
))^2 );
258
259 % plotting
260 figure(2);
261 clf;
262 hold on;
263 pcolor(ogp);
264 set(plot( (x_particle-ogxmin)/ogres, (y_particle-ogymin)/ogres, 'g.' ),'
MarkerSize',10,'Color',[0 0.6 0]);
265 set(plot( (x_odom_only-ogxmin)/ogres, (y_odom_only-ogymin)/ogres, 'r.' ),'
MarkerSize',20);
266 x = (x_interp(i)-ogxmin)/ogres;
267 y = (y_interp(i)-ogymin)/ogres;

```

```

268     th = theta_interp(i);
269     if ~isnan(y_laser(i,1)) & y_laser(i,1) <= y_laser_max
270         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);
271     end
272     if ~isnan(y_laser(i,640)) & y_laser(i,640) <= y_laser_max
273         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);
274     end
275     r = 0.15/ogres;
276     set(rectangle('Position', [x-r y-r 2*r 2*r], 'Curvature', [1 1]),'
LineWidth',2,'FaceColor',[0.35 0.35 0.75]);
277     set(plot([x x+r*cos(th)]', [y y+r*sin(th)]', 'k-'),'LineWidth',2);
278     set(plot((mean(x_particle)-ogxmin)/ogres, (mean(y_particle)-ogymin)/ogres,
'g.' ),'MarkerSize',20);
279     colormap(1-gray);
280     shading('flat');
281     axis equal;
282     axis off;
283
284     % save the video frame
285     M = getframe;
286     writeVideo(vid,M);
287
288     pause(0.01);
289
290 end
291
292 close(vid);
293
294 % final error plots
295 figure(3);
296 clf;
297 hold on;
298 plot(t_laser, pf_err, 'g-');
299 plot(t_laser, wo_err, 'r-');
300 xlabel('t [s]');
301 ylabel('error [m]');
302 legend('particle filter', 'odom', 'Location', 'NorthWest');
303 title('error (estimate-true)');
304 print -dpng ass2_q2.png

```