AER521 Assignment 2 Report

# Part 1: Building an Occupancy Grid

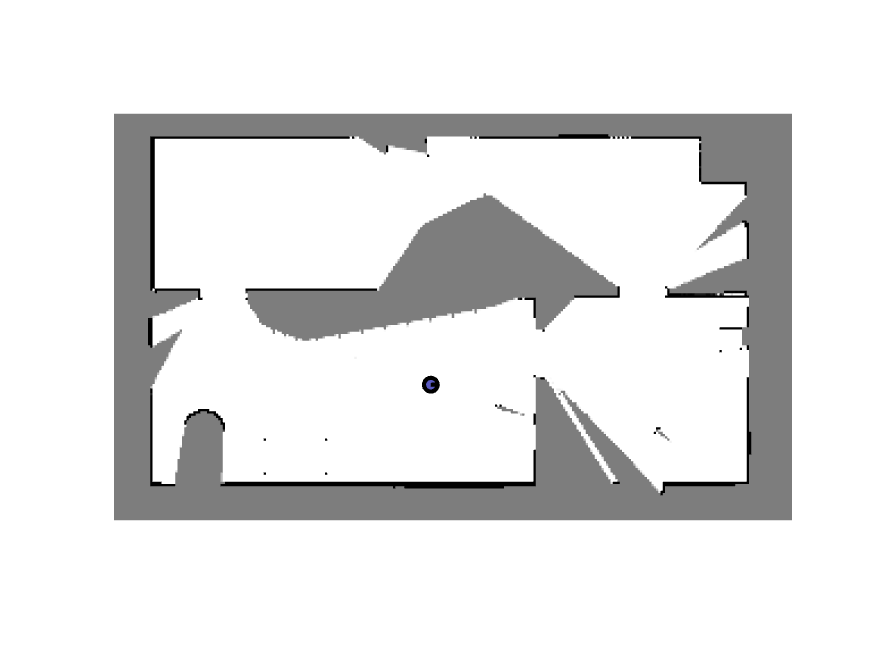


Figure 1: Occupancy grid from laser

The occupancy grid looks similar to the solution grid, but with certain walls replaced by free space and edges looking more ragged.

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.

# Part 2: Particle Filter

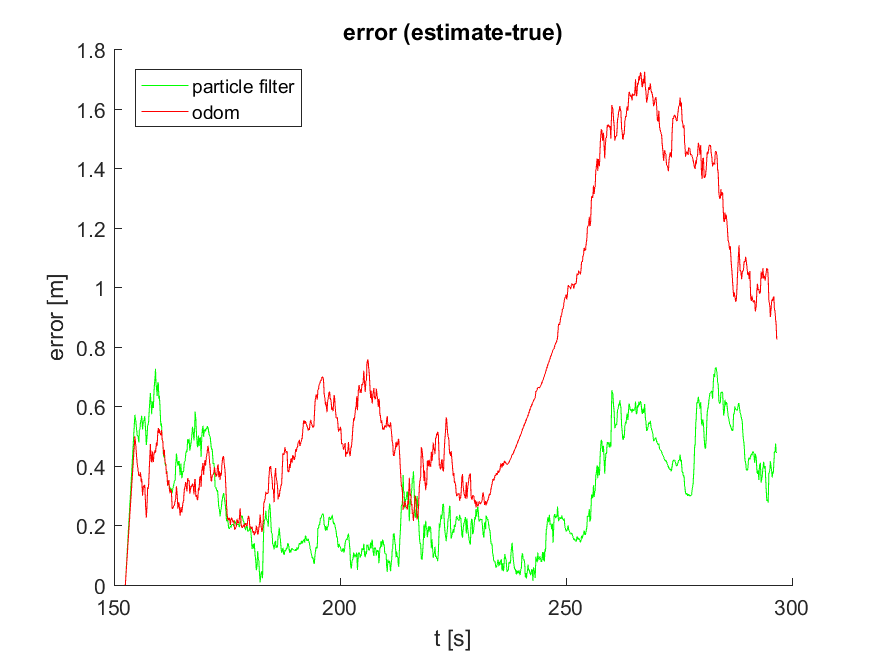


Figure 2: Position error of odometry and particle filter

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 A: Question 1

% =========

% ass2\_q1.m

% =========

%

% This assignment will introduce you to the idea of first building an

% occupancy grid then using that grid to estimate a robot's motion using a

% particle filter.

%

% There are two questions to complete (5 marks each):

%

% Question 1: code occupancy mapping algorithm

% Question 2: see ass2\_q2.m

%

% Fill in the required sections of this script with your code, run it to

% generate the requested plot/movie, then paste the plots into a short report

% that includes a few comments about what you've observed. Append your

% version of this script to the report. Hand in the report as a PDF file

% and the two resulting AVI files from Questions 1 and 2.

%

% requires: basic Matlab, 'gazebo.mat'

%

% T D Barfoot, January 2016

%

clear all;

% set random seed for repeatability

rng(1);

% ==========================

% load the dataset from file

% ==========================

%

% ground truth poses: t\_true x\_true y\_true theta\_true

% odometry measurements: t\_odom v\_odom omega\_odom

% laser scans: t\_laser y\_laser

% laser range limits: r\_min\_laser r\_max\_laser

% laser angle limits: phi\_min\_laser phi\_max\_laser

%

load gazebo.mat;

% =======================================

% Question 1: build an occupancy grid map

% =======================================

%

% Write an occupancy grid mapping algorithm that builds the map from the

% perfect ground-truth localization. Some of the setup is done for you

% below. The resulting map should look like "ass2\_q1\_soln.png". You can

% watch the movie "ass2\_q1\_soln.mp4" to see what the entire mapping process

% should look like. At the end you will save your occupancy grid map to

% the file "occmap.mat" for use in Question 2 of this assignment.

% allocate a big 2D array for the occupancy grid

ogres = 0.05; % resolution of occ grid

ogxmin = -7; % minimum x value

ogxmax = 8; % maximum x value

ogymin = -3; % minimum y value

ogymax = 6; % maximum y value

ognx = (ogxmax-ogxmin)/ogres; % number of cells in x direction

ogny = (ogymax-ogymin)/ogres; % number of cells in y direction

oglo = zeros(ogny,ognx); % occupancy grid in log-odds format

ogp = zeros(ogny,ognx); % occupancy grid in probability format

% precalculate some quantities

numodom = size(t\_odom,1);

npoints = size(y\_laser,2);

angles = linspace(phi\_min\_laser, phi\_max\_laser,npoints);

dx = ogres\*cos(angles);

dy = ogres\*sin(angles);

% interpolate the noise-free ground-truth at the laser timestamps

t\_interp = linspace(t\_true(1),t\_true(numodom),numodom);

x\_interp = interp1(t\_interp,x\_true,t\_laser);

y\_interp = interp1(t\_interp,y\_true,t\_laser);

theta\_interp = interp1(t\_interp,theta\_true,t\_laser);

omega\_interp = interp1(t\_interp,omega\_odom,t\_laser);

% set up the plotting/movie recording

vid = VideoWriter('ass2\_q1.avi');

open(vid);

figure(1);

clf;

pcolor(ogp);

colormap(1-gray);

shading('flat');

axis equal;

axis off;

M = getframe;

writeVideo(vid,M);

% Variables

nsamples = length(t\_laser);

cos\_angles = cos(angles);

sin\_angles = sin(angles);

% loop over laser scans (every fifth)

for i=1:5:size(t\_laser,1)

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

% Transform from laser frame to robot frame

x\_turtle = y\_laser(i, :) .\* cos\_angles - 0.1;

y\_turtle = y\_laser(i, :) .\* sin\_angles;

% Make into homogeneous vector

laser = [ x\_turtle; y\_turtle; ones(1, npoints)];

% Rigid transform to world frame

H = [ cos(theta\_interp(i)) -sin(theta\_interp(i)) x\_interp(i);

sin(theta\_interp(i)) cos(theta\_interp(i)) y\_interp(i);

0 0 1];

laser = H \* laser;

% Convert to grid frame

x\_mat = (x\_interp(i)-ogxmin)/ogres;

y\_mat = (y\_interp(i)-ogymin)/ogres;

l\_x = (laser(1, :)-ogxmin)/ogres;

l\_y = (laser(2, :)-ogymin)/ogres;

l\_x(isnan(l\_x)) = [];

l\_y(isnan(l\_y)) = [];

% Occupancy grid updates

resol = 110;

for n = 1:size(l\_x, 2)

% Find Interpolated distance

x\_line = min(300, round(linspace(x\_mat, l\_x(n), resol)));

y\_line = min(180, round(linspace(y\_mat, l\_y(n), resol)));

% UPdate free space

for m = 1:resol

oglo(y\_line(m), x\_line(m)) = oglo(y\_line(m), x\_line(m)) - 1;

end

% Update obstacle grids

oglo(y\_line(m), x\_line(m)) = oglo(y\_line(m), x\_line(m)) + 5;

end

% Update probability grid

ogp = exp(oglo)./(1+exp(oglo));

% ------end of your occupancy grid mapping algorithm-------

% draw the map

clf;

pcolor(ogp);

colormap(1-gray);

shading('flat');

axis equal;

axis off;

% draw the robot

hold on;

x = (x\_interp(i)-ogxmin)/ogres;

y = (y\_interp(i)-ogymin)/ogres;

th = theta\_interp(i);

r = 0.15/ogres;

set(rectangle( 'Position', [x-r y-r 2\*r 2\*r], 'Curvature', [1 1]),'LineWidth',2,'FaceColor',[0.35 0.35 0.75]);

set(plot([x x+r\*cos(th)]', [y y+r\*sin(th)]', 'k-'),'LineWidth',2);

% save the video frame

M = getframe;

writeVideo(vid,M);

pause(0.1);

end

close(vid);

print -dpng ass2\_q1.png

save occmap.mat ogres ogxmin ogxmax ogymin ogymax ognx ogny oglo ogp;

# Appendix B: Question 2

% =========

% ass2\_q2.m

% =========

%

% This assignment will introduce you to the idea of first building an

% occupancy grid then using that grid to estimate a robot's motion using a

% particle filter.

%

% There are three questions to complete (5 marks each):

%

% Question 1: see ass2\_q1.m

% Question 2: code particle filter to localize from known map

%

% Fill in the required sections of this script with your code, run it to

% generate the requested plot/movie, then paste the plots into a short report

% that includes a few comments about what you've observed. Append your

% version of this script to the report. Hand in the report as a PDF file

% and the two resulting AVI files from Questions 1 and 2.

%

% requires: basic Matlab, 'gazebo.mat', 'occmap.mat'

%

% T D Barfoot, January 2016

%

clear all;

% set random seed for repeatability

rng(1);

% ==========================

% load the dataset from file

% ==========================

%

% ground truth poses: t\_true x\_true y\_true theta\_true

% odometry measurements: t\_odom v\_odom omega\_odom

% laser scans: t\_laser y\_laser

% laser range limits: r\_min\_laser r\_max\_laser

% laser angle limits: phi\_min\_laser phi\_max\_laser

%

load gazebo.mat;

% ================================================

% load the occupancy map from question 1 from file

% ================================================

% ogres: resolution of occ grid

% ogxmin: minimum x value

% ogxmax: maximum x value

% ogymin: minimum y value

% ogymax: maximum y value

% ognx: number of cells in x direction

% ogny: number of cells in y direction

% oglo: occupancy grid in log-odds format

% ogp: occupancy grid in probability format

load occmap.mat;

% =========================================================================

% Question 2: localization from an occupancy grid map using particle filter

% =========================================================================

%

% Write a particle filter localization algorithm to localize from the laser

% rangefinder readings, wheel odometry, and the occupancy grid map you

% built in Question 1. We will only use two laser scan lines at the

% extreme left and right of the field of view, to demonstrate that the

% algorithm does not need a lot of information to localize fairly well. To

% make the problem harder, the below lines add noise to the wheel odometry

% and to the laser scans. You can watch the movie "ass2\_q2\_soln.mp4" to

% see what the results should look like. The plot "ass2\_q2\_soln.png" shows

% the errors in the estimates produced by wheel odometry alone and by the

% particle filter look like as compared to ground truth; we can see that

% the errors are much lower when we use the particle filter.

% interpolate the noise-free ground-truth at the laser timestamps

numodom = size(t\_odom,1);

t\_interp = linspace(t\_true(1),t\_true(numodom),numodom);

x\_interp = interp1(t\_interp,x\_true,t\_laser);

y\_interp = interp1(t\_interp,y\_true,t\_laser);

theta\_interp = interp1(t\_interp,theta\_true,t\_laser);

omega\_interp = interp1(t\_interp,omega\_odom,t\_laser);

% interpolate the wheel odometry at the laser timestamps and

% add noise to measurements (yes, on purpose to see effect)

v\_interp = interp1(t\_interp,v\_odom,t\_laser) + 0.2\*randn(size(t\_laser,1),1);

omega\_interp = interp1(t\_interp,omega\_odom,t\_laser) + 0.04\*randn(size(t\_laser,1),1);

% add noise to the laser range measurements (yes, on purpose to see effect)

% and precompute some quantities useful to the laser

y\_laser = y\_laser + 0.1\*randn(size(y\_laser));

npoints = size(y\_laser,2);

angles = linspace(phi\_min\_laser, phi\_max\_laser,npoints);

dx = ogres\*cos(angles);

dy = ogres\*sin(angles);

y\_laser\_max = 5; % don't use laser measurements beyond this distance

% particle filter tuning parameters (yours may be different)

nparticles = 200; % number of particles

v\_noise = 0.2; % noise on longitudinal speed for propagating particle

u\_noise = 0.2; % noise on lateral speed for propagating particle

omega\_noise = 0.04; % noise on rotational speed for propagating particle

laser\_var = 0.5^2; % variance on laser range distribution

w\_gain = 10\*sqrt( 2 \* pi \* laser\_var ); % gain on particle weight

% generate an initial cloud of particles

x\_particle = x\_true(1) + 0.5\*randn(nparticles,1);

y\_particle = y\_true(1) + 0.3\*randn(nparticles,1);

theta\_particle = theta\_true(1) + 0.1\*randn(nparticles,1);

% compute a wheel odometry only estimate for comparison to particle

% filter

x\_odom\_only = x\_true(1);

y\_odom\_only = y\_true(1);

theta\_odom\_only = theta\_true(1);

% error variables for final error plots - set the errors to zero at the start

pf\_err(1) = 0;

wo\_err(1) = 0;

% set up the plotting/movie recording

vid = VideoWriter('ass2\_q2.avi');

open(vid);

figure(2);

clf;

hold on;

pcolor(ogp);

set(plot( (x\_particle-ogxmin)/ogres, (y\_particle-ogymin)/ogres, 'g.' ),'MarkerSize',10,'Color',[0 0.6 0]);

set(plot( (x\_odom\_only-ogxmin)/ogres, (y\_odom\_only-ogymin)/ogres, 'r.' ),'MarkerSize',20);

x = (x\_interp(1)-ogxmin)/ogres;

y = (y\_interp(1)-ogymin)/ogres;

th = theta\_interp(1);

r = 0.15/ogres;

set(rectangle( 'Position', [x-r y-r 2\*r 2\*r], 'Curvature', [1 1]),'LineWidth',2,'FaceColor',[0.35 0.35 0.75]);

set(plot([x x+r\*cos(th)]', [y y+r\*sin(th)]', 'k-'),'LineWidth',2);

set(plot( (mean(x\_particle)-ogxmin)/ogres, (mean(y\_particle)-ogymin)/ogres, 'g.' ),'MarkerSize',20);

colormap(1-gray);

shading('flat');

axis equal;

axis off;

M = getframe;

writeVideo(vid,M);

% loop over laser scans

for i=2:size(t\_laser,1)

% update the wheel-odometry-only algorithm

dt = t\_laser(i) - t\_laser(i-1);

v = v\_interp(i);

omega = omega\_interp(i);

x\_odom\_only = x\_odom\_only + dt\*v\*cos( theta\_odom\_only );

y\_odom\_only = y\_odom\_only + dt\*v\*sin( theta\_odom\_only );

phi = theta\_odom\_only + dt\*omega;

while phi > pi

phi = phi - 2\*pi;

end

while phi < -pi

phi = phi + 2\*pi;

end

theta\_odom\_only = phi;

% loop over the particles

for n=1:nparticles

% propagate the particle forward in time using wheel odometry

% (remember to add some unique noise to each particle so they

% spread out over time)

v = v\_interp(i) + v\_noise\*randn(1);

u = u\_noise\*randn(1);

omega = omega\_interp(i) + omega\_noise\*randn(1);

x\_particle(n) = x\_particle(n) + dt\*(v\*cos( theta\_particle(n) ) - u\*sin( theta\_particle(n) ));

y\_particle(n) = y\_particle(n) + dt\*(v\*sin( theta\_particle(n) ) + u\*cos( theta\_particle(n) ));

phi = theta\_particle(n) + dt\*omega;

while phi > pi

phi = phi - 2\*pi;

end

while phi < -pi

phi = phi + 2\*pi;

end

theta\_particle(n) = phi;

% pose of particle in initial frame

T = [cos(theta\_particle(n)) -sin(theta\_particle(n)) x\_particle(n); ...

sin(theta\_particle(n)) cos(theta\_particle(n)) y\_particle(n); ...

0 0 1];

% compute the weight for each particle using only 2 laser rays

% (right=beam 1 and left=beam 640)

w\_particle(n) = 1.0;

for beam=1:2

% we will only use the first and last laser ray for

% localization

if beam==1 % rightmost beam

j = 1;

elseif beam==2 % leftmost beam

j = 640;

end

% ------insert your particle filter weight calculation here ------

% Check if the laser measurement is valid

if isnan(y\_laser(i, j)) || y\_laser(i, j) > y\_laser\_max

continue

end

% Laser value in world frame

x\_turtle = y\_laser(i, j) \* cos(angles(j)) - 0.1;

y\_turtle = y\_laser(i, j) \* sin(angles(j));

% Make into homogeneous vector

laser = [x\_turtle; y\_turtle; 1];

% Rigid transform to world frame

R = [ cos(theta\_interp(i)) -sin(theta\_interp(i)) x\_interp(i);

sin(theta\_interp(i)) cos(theta\_interp(i)) y\_interp(i);

0 0 1];

laser = R \* laser;

% Convert to MATLAB frame

x\_l = round((laser(1)-ogxmin)/ogres);

y\_l = round((laser(2)-ogymin)/ogres);

% Particle laser prediction

H = T \* [ cos(angles(j)) -sin(angles(j)) -0.1;

sin(angles(j)) cos(angles(j)) 0;

0 0 1];

x\_pl = H(1, 3);

y\_pl = H(2, 3);

theta\_pl = acos(H(1, 1));

% To occupancy grid frame

x\_pl = min(300, max(1, (x\_pl-ogxmin)/ogres));

y\_pl = min(180, max(1, (y\_pl-ogymin)/ogres));

% Trace laser scan angle to calculate the expected range value

for incr = r\_min\_laser:(ogres/2):y\_laser\_max

% Stop if the laser hit an obstacle

if (oglo(round(y\_pl), round(x\_pl)) >= 0)

continue

end

% Increment along ray

x\_pl = max(1, min(300, x\_pl + incr\*cos(theta\_pl)));

y\_pl = max(1, min(180, y\_pl + incr\*sin(theta\_pl)));

end

% Calculate and update weights, assuming Gaussian distribution

gauss = (1/sqrt(2\*pi\*laser\_var))\*exp(-norm([x\_pl; y\_pl]-[x\_l; y\_l])^2/(2\*laser\_var));

w\_particle(n) = w\_particle(n) + w\_gain\*gauss;

% ------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