ROB521 A3 - Lidar Mapping & Localization

Ethan Rajah

April 7, 2025

1 Question 1

This question required the implementation of an occupancy grid mapping algorithm using Lidar data to build a map of the environment. This required tracing lidar distance measurements from the current robot pose at each timestep during the robots motion and updating a probability grid map based on the if cells were occupied or not. More specifically, the algorithm marks free space grid indices along the ray traced by the Lidar sensor from the current robot pose by decrementing the probability of occupancy of the cell by 0.5, and marks occupied space grid indices (the end points of the ray denoted by the Lidar scan) by incrementing the probability of occupancy of the cell by 1.5. These values were tuned by running the algorithm and observing how defined the free (white) and occupied (black) space was in the resulting map. Finally, the logits are converted into probabilities using the sigmoid function. The resulting map is shown in **Figure 1**. The figure shows that this algorithm is able to build a map of complex environments with high accuracy, maintaining the structure of the robot's surroundings.

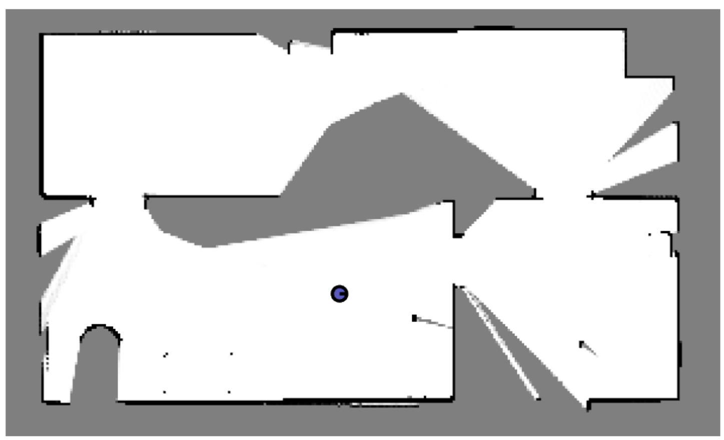

Figure 1: Built map from occupancy grid mapping algorithm and Lidar data.

2 Question 2

This question required the implementation of a particle filter algorithm to localize a robot in a known map using Lidar data. The algorithm initializes a set of particles in the map, and at each timestep, the particles are propagated through a motion model using the odometry data and corrected using the Lidar data. The particles are weighted based on how likely they are to be in the correct position given the current Lidar measurements. This is done by reweighing the particles at each timestep based on their residual error from the propagated predicted measurement given the predicted pose and the actual Lidar measurement. Similar to Question 1, the measurements at each timestep are ray traced based on the estimated pose of the particle to obtain the closest occupied cell in the map. The actual measurement is set as the mean of a Gaussian distribution defining the likelihood of the particle given the measurement, and the predicted measurement is used to aid in forming the PDF, which is then used to reweigh the particles. The particles are then resampled based on their weights to ensure that the particles with higher weights are more likely to be selected. The resulting map is shown in **Figure 2**, which shows the localization error between the odometry estimates and the particle filter estimates. The figure shows that the particle filtering technique is far superior to odometry estimates, as the localization error is significantly lower on average.

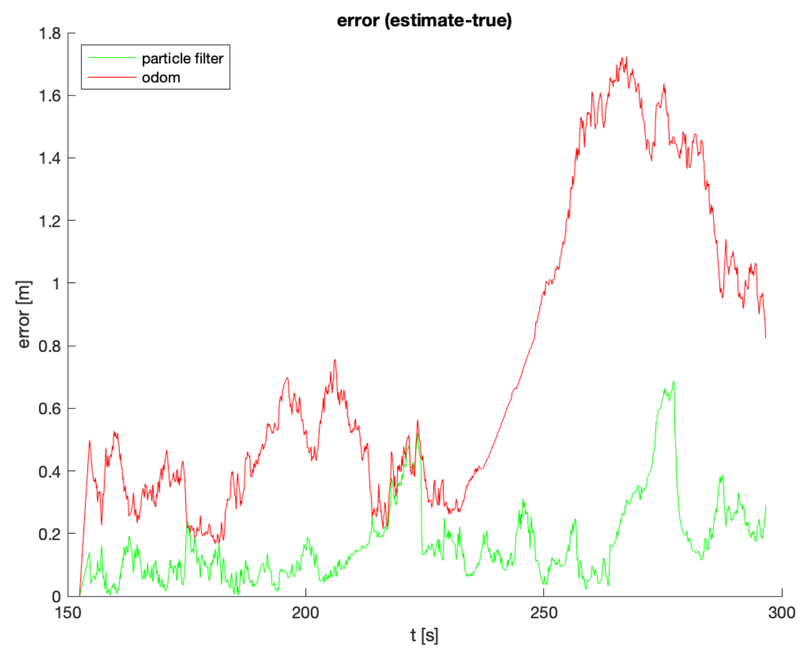

Figure 2: Localization error comparison between odometry estimates and particle filter estimates.

3 MATLAB Code

The MATLAB code for these implementations are provided here, as well as are attached in the submission.

```
% =========
% 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------

    % Get current robot pose
    x = (x_interp(i)-ogxmin)/ogres;
    y = (y_interp(i)-ogymin)/ogres;

    % Loop over each laser scan point at this timestep. Imagine each
        laser scan point as a ray/line from the robot to the
```

```
        measured endpoint
    for j=1:npoints
        % Check if the laser scan point is within the range of the laser
        if y_laser(i,j) <= r_max_laser && y_laser(i,j) >= r_min_laser
            % Get laser scan in grid coordinates
            range_pixel = y_laser(i,j) / ogres;
            % Get theta range based on the robot's orientation and laser scan angle
            theta_laser = theta_interp(i) + angles(j);
            % Normalize the angle to be between -pi and pi
            theta_laser = atan2(sin(theta_laser), cos(theta_laser));

            % Get ray endpoints - need this to increase logits as it marks obstacles
            x_end = round(x + range_pixel * cos(theta_laser));
            y_end = round(y + range_pixel * sin(theta_laser));

            % Get ray indices by using ray angle to mark free space until the range is reached
            x_idxs = [];
            y_idxs = [];
            x_step = x;
            y_step = y;
            for step = 1:ceil(range_pixel)
                x_step = round(x + step * cos(theta_laser));
                y_step = round(y + step * sin(theta_laser));
                % Stop if out of bounds
                if x_step <= 0 || x_step > ognx || y_step <= 0 || y_step > ogny
                    break;
                end
                x_idxs = [x_idxs; x_step];
                y_idxs = [y_idxs; y_step];
            end

            % Update the occupancy grid log-odds. Idxs are free space so decrease the log-odds, endpoints are obstacles so increase the log-odds
            for k = 1:length(x_idxs)-1
                if x_idxs(k) > 0 && x_idxs(k) <= ognx && y_idxs(k) > 0 && y_idxs(k) <= ogny
                    oglo(y_idxs(k), x_idxs(k)) = oglo(y_idxs(k), x_idxs(k)) - 0.5;
                end
            end
            if x_end > 0 && x_end <= ognx && y_end > 0 && y_end <= ogny
                oglo(y_end, x_end) = oglo(y_end, x_end) + 1.5;
            end
```

```
        end
    end

    % Update the occupancy grid in probability format
    ogp = 1 - 1./(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
%   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 ------

            % Previous steps computed the prediction step of the particle filter. Need to implement corrections step based on the two laser beams
            laser_range = y_laser(i,j);
            if isnan(laser_range) || laser_range > y_laser_max
                continue;
            end

            % Get set of occupied cells from ray
            range_pixel = laser_range / ogres;
            theta_laser = theta_particle(n) + angles(j);
            theta_laser = atan2(sin(theta_laser), cos(theta_laser));
            x_particle_pixel = round((x_particle(n) - ogxmin) / ogres);
            y_particle_pixel = round((y_particle(n) - ogymin) / ogres);
            x_end = round(x_particle_pixel + range_pixel * cos(theta_laser));
            y_end = round(y_particle_pixel + range_pixel * sin(theta_laser));
            x_idxs = [];
            y_idxs = [];
            x_step = x;
            y_step = y;
            for step = 1:ceil(range_pixel)
                x_step = round(x + step * cos(theta_laser));
                y_step = round(y + step * sin(theta_laser));
                % Stop if out of bounds
```

```
                if x_step <= 0 || x_step > ognx || y_step <= 0 || y_step > ogny
                    break;
                end
                x_idxs = [x_idxs; x_step];
                y_idxs = [y_idxs; y_step];
            end

            % Find closest occupied cell to current particle position
            threshold = 0.6;
            for k = 1:length(x_idxs)-1
                if x_idxs(k) > 0 && x_idxs(k) <= ognx && y_idxs(k) > 0 && y_idxs(k) <= ogny
                    if ogp(y_idxs(k), x_idxs(k)) > threshold
                        break;
                    end
                end
            end
            x_end = x_idxs(k);
            y_end = y_idxs(k);
            x_end = x_end * ogres + ogxmin;
            y_end = y_end * ogres + ogymin;
            laser_pred = sqrt((x_end - x_particle(n))^2 + (y_end - y_particle(n))^2);

            % Create Gaussian distribution for the laser range based on prediction and actual measurement. Allows us to reweigh particles based on
            % how well they match the actual laser range
            w_particle(n) = w_particle(n) * w_gain * normpdf(laser_range, laser_pred, sqrt(laser_var));

            % ------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
```