Assignment #3

Lidar Mapping and Localization

Seonghak Lee

Occupancy Grid Mapping (ass3_q1.m)

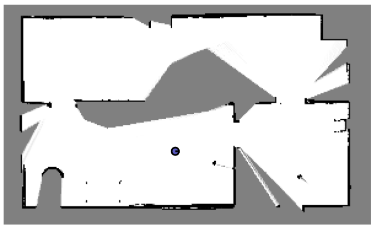
Process:

1. **Grid Initialization:**
   - A grid is initialized based on a predefined resolution and map dimensions. Each cell in this grid will represent whether the corresponding area in the environment is free, occupied, or unknown.
2. **Data Acquisition:**
   - The robot uses laser scans and its known ground-truth position to update the occupancy grid.
3. **Mapping Algorithm:**
   - For each laser scan reading, the robot's position is converted into grid coordinates.
   - If the laser reading is valid (within the range limits), the endpoint of the laser beam in the map's coordinates is calculated.
   - A line is drawn from the robot to this endpoint using a ray-tracing algorithm. Cells along this line are updated as free, and the endpoint cell is marked as occupied.
   - Log-odds representation is used to handle updates, which helps in maintaining numerical stability and provides an efficient way to integrate multiple measurements.
4. **Visualization:**
   - The probability grid is visualized using a grayscale color map, and the robot's position is marked. The resulting map is recorded frame by frame into a video file.

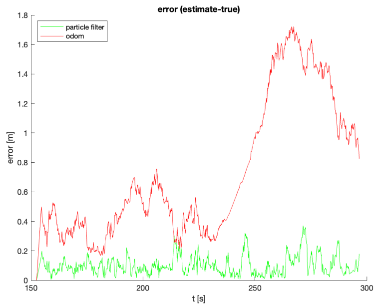
Particle Filter Localization (ass3_q2.m)

Process:

1. **Particle Initialization:**
   - A set of particles representing possible robot states (positions and orientations) is generated.
2. **Motion Model:**
   - Each particle is propagated over time using a motion model that incorporates the latest odometry readings and adds noise to account for uncertainty in movement.
3. **Weighting Particles:**
   - Weights are assigned to each particle based on how well the particle's predicted measurements align with actual laser scans, given the current map. This step uses a Gaussian probability density function to compare the observed laser range with the expected range determined by casting a ray from the particle's estimated position on the map.
4. **Resampling:**
   - Particles are resampled based on their weights, allowing those with higher likelihoods to be selected more often, while those with lower likelihoods are discarded. This resampling step effectively concentrates the particle cloud around the most probable states.
5. **Estimation and Comparison:**
   - The final pose estimation is derived from the mean of all particle states. This estimate is compared against the wheel odometry estimate and the true pose for accuracy assessment. The error between the estimated pose and the true pose is plotted over time to evaluate the performance of the particle filter.
6. **Visualization and Output:**
   - The process is visualized by plotting the particles on the occupancy grid, showcasing the estimated position and uncertainty of the robot. The frames are saved to create an animation of the localization process.



Q1 Figure



Q2 Figure

Result

The Q1 Figure shows an occupancy grid map of an environment where a robot has navigated. In this map, darker areas indicate the presence of obstacles, lighter areas signify open space, and gray areas represent unexplored or uncertain regions. The map is created by converting laser scan data into a grid format, where each cell's color reflects the likelihood of being occupied. The robot's path and orientation are marked, showing where it has traveled and taken scans. This occupancy grid is useful for the robot to understand its surroundings, avoid obstacles, and plan its movement.

The Q2 Figure illustrates the performance of a particle filter in localizing a robot against the baseline of wheel odometry measurements. The green line, representing the particle filter, generally maintains a lower error compared to the wheel odometry (red line), indicating that the particle filter provides a more accurate estimation of the robot's position. The graph shows time on the x-axis and the localization error on the y-axis. The particle filter's lower error trajectory shows its robustness in estimating the robot's position, even with noisy sensor data. Conversely, the odometry's error increases significantly over time, demonstrating that reliance solely on wheel odometry for localization can lead to large drifts in positional accuracy. The particle filter's ability to fuse multiple sensor readings and probabilistically determine the robot's position results in a much more reliable localization over time.

Conclusion

Through the above steps, the assignments utilize laser scans and odometry data to build an occupancy grid and localize the robot within that grid. The occupancy grid mapping script demonstrates how an environment can be represented probabilistically, and the particle filter localization script shows an approach to estimate the robot's pose within that environment. These components are crucial for autonomous navigation and decision-making in robotics.

Q1 Code

```
% =========
% 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.mbresenham_line
%
% 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------

    x_pos = x_interp(i);
    y_pos = y_interp(i);
    theta_pos = theta_interp(i);

    x_start = round((x_pos - ogxmin) / ogres);
    y_start = round((y_pos - ogymin) / ogres);

    for j = 1:npoints
```

```
    if ~isnan(y_laser(i,j)) && r_min_laser < y_laser(i,j) && y_laser(i,j) < r_max_laser

        range_mes = y_laser(i,j);
        cur_angle = angles(j) + theta_pos;
        x_end = x_start + round(range_mes * cos(cur_angle) / ogres);
        y_end = y_start + round(range_mes * sin(cur_angle) / ogres);
        [rr, cc] = ray_trace(x_start, y_start, x_end, y_end);

        for k = 1:length(rr)
            if rr(k) > 0 && rr(k) <= ogny && cc(k) > 0 && cc(k) <= ognx
                if k < length(rr)
                    oglo(rr(k),cc(k)) = oglo(rr(k),cc(k)) - 0.5;
                else
                    oglo(rr(k),cc(k)) = oglo(rr(k),cc(k)) + 3;
                end
            end
        end
    end
end

oglo = min(max(oglo, -10), 10);
ogp = 1 - 1./(1 + exp(oglo));
oglo;
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;

function [rr, cc] = ray_trace(x1, y1, x2, y2)
    % Initialize the return arrays
    rr = [];
    cc = [];

    % Calculate the x and y distances between the two points
    dx = abs(x2 - x1);
    dy = abs(y2 - y1);

    % Calculate the step size for the x and y axes
    if x1 < x2
        sx = 1;
    else
        sx = -1;
    end

    if y1 < y2
        sy = 1;
    else
        sy = -1;
    end

    % Initialize the error term
    err = dx - dy;

    % Traverse the line using Bresenham's algorithm
    while true
        % Add the current point to the return arrays
        rr = [rr; y1];
        cc = [cc; x1];

        % Check if we've reached the end point
        if x1 == x2 && y1 == y2
            break;
        end

        % Update the error term
        e2 = 2*err;
        if e2 > -dy
            err = err - dy;
            x1 = x1 + sx;
        end
        if e2 < dx
            err = err + dx;
            y1 = y1 + sy;
        end
```

```
    end
end
```

Q2 Code

```
% =========
% ass3_q2.m
% =========
%
% This assignment will in   troduce 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 ------

      if ~isnan(y_laser(i,j)) && y_laser(i,j) < y_laser_max

         row_pos = max(1,round((y_particle(n)-ogymin)/ogres));
         col_pos = max(1,round((x_particle(n)-ogxmin)/ogres));
         threshold = 0.5;
         y_laser_pred = predict_laser_range(row_pos, col_pos, theta_particle(n), angles(j), ogp, ogres, threshold);
```

```
            w_particle(n) = w_particle(n)*w_gain*pdf(y_laser(i, j), y_laser_pred, sqrt(laser_var));

          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

function y_exp = predict_laser_range(row, col, theta, beam_angle, map, ogres, thresh)

    new_angle = atan2(sin(theta+beam_angle), cos(theta+beam_angle));
    incr  = 0;
    r_p = row;
    c_p = col;

    if -pi/4<=new_angle && new_angle<=pi/4
        c_inc = 1;
        r_inc = tan(new_angle);
        a = cos(new_angle);
    elseif 3*pi/4<=new_angle || new_angle<=-3*pi/4
        c_inc = -1;
        r_inc = -tan(new_angle);
        a = cos(new_angle);
    elseif pi/4<new_angle && new_angle<3*pi/4
        c_inc = 1/tan(new_angle);
        r_inc = 1;
        a = sin(new_angle);
    else
```

```
        c_inc = -1/tan(new_angle);
        r_inc = -1;
        a = sin(new_angle);

    end


    [row_bound, col_bound] = size(map);
    while r_p > 0 && c_p>0 && r_p<=row_bound && c_p<=col_bound && map(r_p, c_p) < thresh
        incr = incr +1;
        r_p = row + round(incr * r_inc);
        c_p = col + round(incr * c_inc);
    end
    y_exp = abs((incr/a)*ogres);
end
function p = pdf(x, mu, sigma)
    coeff = 1 / (sigma * sqrt(2 * pi));
    exponent = exp(-0.5 * ((x - mu) / sigma).^2);
    p = coeff * exponent;
end
```