ROB521 Assignment1

Erkang Liu 1002048429

February 23rd 2020

Introduction

This assignment will introduce you to the idea of estimating the motion of a mobile robot using wheel odometry, and then also using that wheel odometry to make a simple map. It uses a dataset previously gathered in a mobile robot simulation environment called Gazebo. Watch the video, 'gazebo.mp4' to visualize what the robot did, what its environment looks like, and what its sensor stream looks like.

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

Question 1: code (noise-free) wheel odometry algorithm

Question 2: add noise to data and re-run wheel odometry algorithm

Question 3: build a map from ground truth and noisy wheel odometry

Fill in the required sections of this script with your code, run it to generate the requested plots, 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.

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

1 Noise-free wheel odometry algorithm

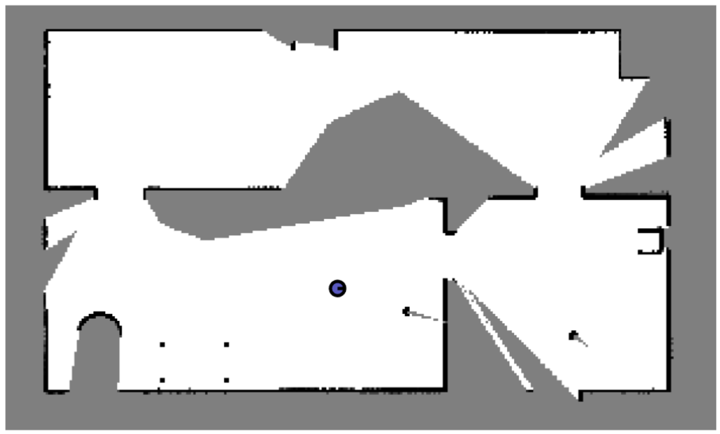

Figure 1: Position, heading, position error and heading error over time

This resulted figure is similar ass1_q1_soln.png. As can be seen, obstacles are mapped black, unknown areas are gray and clear areas as white. Additionally, it even performed better at the top right corner where the "box" in the corner is mapped, which results from capping the omega of the robot.

2 Mapping with wheel odometry

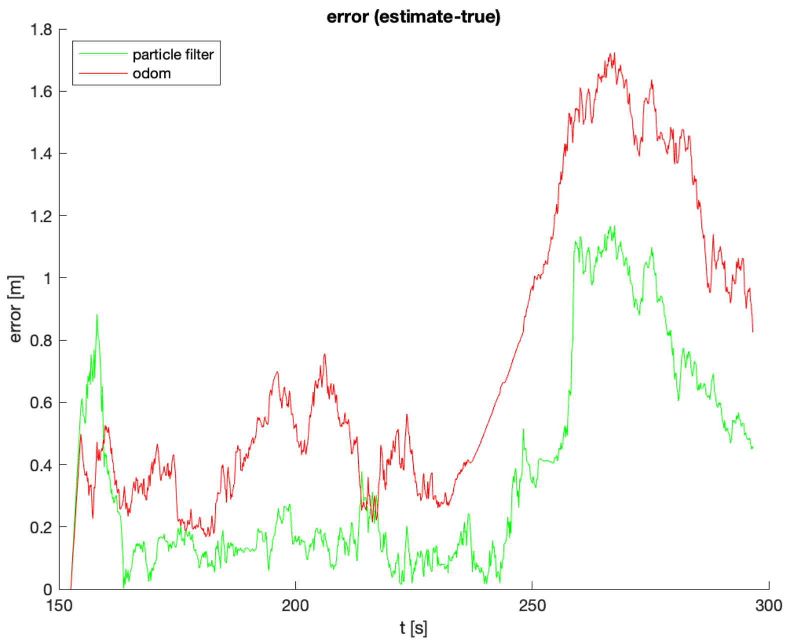

Figure 2: Mapping results with both the true poses and odometry poses

This result did not match ass2_q2_soln.png perfectly. However, it can be observed that the error from particle filter is consistently lower than odometry after the initial phase, which corresponds to our intuition and expectation.

3 Appendix: MATLAB scripts

3.1 Q1

```
% ------insert your occupancy grid mapping algorithm here------
x = x_interp(i);
y = y_interp(i);
theta = theta_interp(i);
R = [cos(theta), -sin(theta); sin(theta), cos(theta)];

omega = omega_interp(i);
if omega > max_omega
    continue
end
for j = 1:size(y_laser,2)
    ang = angles(j);
    range = y_laser(i,j);

    if isnan(range)
        continue;
    end

    for depth = r_min_laser:ogres:range
        if depth > r_max_laser
            continue
        end


        depth_coords = depth*[cos(ang);sin(ang)];
        pt = [x;y] + R*(depth_coords - [0.1;0]) - [ogxmin;ogymin];
        pt_c = round(pt/ogres);

        if abs(depth-range) <= ogres
            oglo(pt_c(2), pt_c(1)) = oglo(pt_c(2), pt_c(1)) + alpha;
        elseif depth < range
            oglo(pt_c(2), pt_c(1)) = oglo(pt_c(2), pt_c(1)) - beta;
        end
    end
end

%     disp(oglo)
%     image(oglo)
%     break

ogp = exp(oglo) ./ (1 + exp(oglo));
ogp(ogp<0.5)=0;
```

```
%       disp(ogp)


    % ———end of your occupancy grid mapping algorithm————
```

3.2 Q2

```
% ———insert your particle filter weight calculation here ———
            ang = angles(j);
    x = x_interp(i);
    y = y_interp(i);
    theta = theta_interp(i);


    if isnan(y_laser(i, j))
        continue
    end
    if y_laser(i, j) > y_laser_max
        continue
    end

    H = T * [    cos(ang)   -sin(ang)  -0.1;
            sin(ang)   cos(ang)   0;
            0                0                 1];
    xpar =        H(1, 3);
    ypar =        H(2, 3);
    thetapar =    acos(H(1, 1));

    xpar_c =    min(300, max(1, (xpar-ogxmin)/ogres));
    ypar_c =    min(180, max(1, (ypar-ogymin)/ogres));

    for incr = r_min_laser:(ogres/2):y_laser_max
        % Only increment if particle haven't hit obstacle
        if (oglo(round(ypar_c), round(xpar_c)) < 0)
        xpar_c =  max(1, min(300, xpar_c + incr*cos(thetapar)));
        ypar_c =  max(1, min(180, ypar_c + incr*sin(thetapar)));
        end
    end

    laser = [cos(theta), -sin(theta), x;
             sin(theta), cos(theta), y;
             0, 0, 1]*...
            [y_laser(i, j) * cos(ang) - 0.1;...
             y_laser(i, j) * sin(ang);...
             1];
```

```
    xl =    round((laser(1)-ogxmin)/ogres);
    yl =    round((laser(2)-ogymin)/ogres);

    diff = [xpar_c; ypar_c]-[xl; yl];
%              disp(diff);
    gauss = (1/sqrt(2*pi*laser_var))*exp(-norm(diff)^2/(2*laser_var));
    w_particle(n) = w_particle(n) + w_gain*gauss;
% ------end of your particle filter weight calculation-------
```