ROB521 Assignment 1: Wheel Odometry and Mapping

Samuel Atkins | 1002951754

February 23, 2020

Table of Contents

Table of Figures

Part 1: Noise-free Wheel Odometry

In this part of the assignment a wheel odometry algorithm was implemented to predict the heading and position of the robot based on noise-free measurements. Using the velocity measurements from the previous frame, the x_odom, y_odom, and theta_odom values at each measurement iteration were predicted. The odometry algorithm used to update the aforementioned values relies on the following update formulae:

$$x(i) = x(i-1) + \cos(theta\_odom(i-1)) * v\_odom(i-1) * (t\_odom(i) - t\_odom(i-1))$$

$$y(i) = y(i-1) + \sin(theta\_odom(i-1)) * v\_odom(i-1) * (t\_odom(i) - t\_odom(i-1))$$

$$theta\_odom(i) = theta\_odom(i-1) + omega\_odom(i-1) * (t\_odom(i) - t\_odom(i-1))$$

Equation 1: Odometry update equations

Plots were generated to measure the performance of the odometry algorithm versus the robobts true position and heading. The following Figure illustrates the difference between the predicted path and heading versus the actual path and heading:

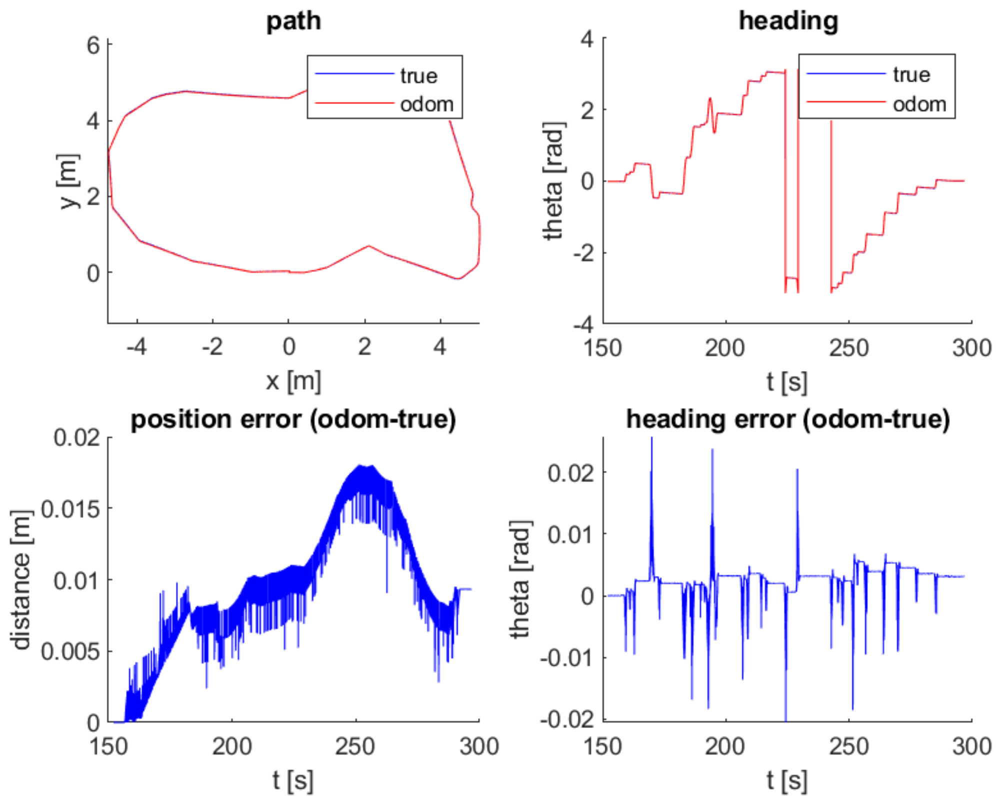

Figure 1: Part 1 true path and heading versus predicted path and heading

Since the measurements were not corrupted with any noise, we expect the difference between the predicted and actual path and heading measurements to be very small. The error subplots show that both the positional error and the heading error remain below 0.02m and 0.025rad, respectively. The true path and heading versus the path and heading measured through odometry are so similar that the difference between the two cannot be observed for the majority of the path and heading plots.

Part 2: Wheel Odometry with Added Noise

In this part, random noise was added to the linear and angular velocity measurements. Then, the wheel odometry algorithm from the previous part was applied to 100 simulations of noisy odometry data. The results of the application of the previous algorithm to the noisy data is illustrated by Figure 2:

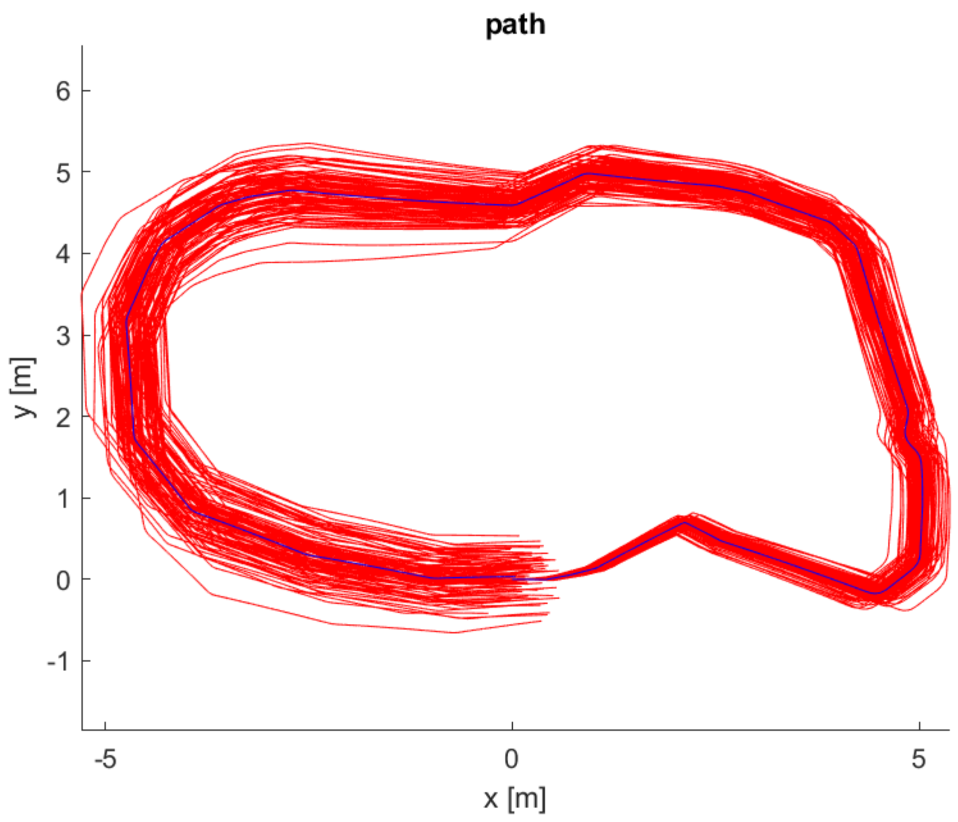

Figure 2: Part 2 predicted path with added noise

The blue line is the true path that the robot followed. The red paths are the odometry predictions for each of the 100 noisy simulations. The difference between the predicted odometry and the actual odometry increases as the robot progresses through the scene. Since wheel odometry is a dead reckoning method, the variance grows without bound. This is demonstrated by the above Figure because the error in the measurements grows as the robot continues to traverse the scene.

Part 3: Building a Map

In this part of the assignment, the sensor measurements were used to create a map of the scene. To accomplish this task, the x and y coordinates of each sensor measurement were calculated in the sensor frame. Then, these points were transformed back into the inertial frame using a transformation matrix. The structure of this transformation matrix is shown below:

Equation 2: Transformation matrix used to transform points from the sensor frame to the inertial frame

cos(theta_odom(i))	-sin(theta_odom(i))	0	x_odom - 0.1*cos(theta_interp(i))
sin(theta_odom(i))	cos(theta_odom(i))	0	y_odom - 0.1*sin(theta_interp(i))
0	0	1	0
0	0	0	1

Two maps were created, a map using the noisy odometry measurements, and a map using the true odometry values. These maps are shown below in Figure 3:

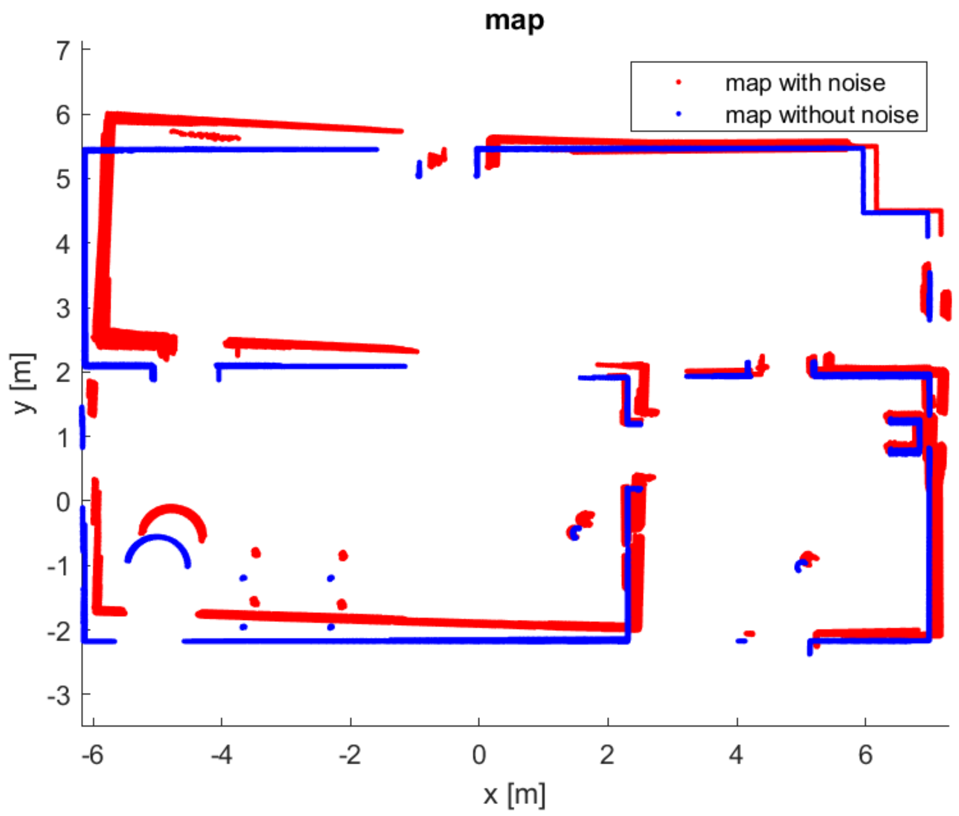

Figure 3: Part 3 generated map with and without noise modelling

The map created using noisy data is thicker than the map using pure data because the noise increases the uncertainty in the robot’s position and has a direct impact on the transformation of the sensor values back into the inertial frame. The thickness of the map increases as the robot travels because the uncertainty propagates. Measurements that the robot observes later on his journey will be less accurate than earlier measurements.

Appendix: MATLAB Code

```
% ======
% ass1.m
% ======
%
% 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.
%
% requires: basic Matlab, 'gazebo.mat'
%
% T D Barfoot, December 2015
%
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: code (noise-free) wheel odometry algorithm
% =======================================================
%
% Write an algorithm to estimate the pose of the robot throughout motion
% using the wheel odometry data (t_odom, v_odom, omega_odom) and assuming
% a differential-drive robot model.  Save your estimate in the variables
% (x_odom y_odom theta_odom) so that the comparison plots can be generated
% below.  See the plot 'ass1_q1_soln.png' for what your results should look
% like.
% variables to store wheel odometry pose estimates
numodom = size(t_odom,1);
x_odom = zeros(numodom,1);
y_odom = zeros(numodom,1);
theta_odom = zeros(numodom,1);

% set the initial wheel odometry pose to ground truth
x_odom(1) = x_true(1);
y_odom(1) = y_true(1);
theta_odom(1) = theta_true(1);

% ------insert your wheel odometry algorithm here-------
%for i=2:numodom
for i=2:numodom
    time_elapsed = t_odom(i) - t_odom(i-1);
    theta_odom(i) = theta_odom(i-1) + omega_odom(i-1) * time_elapsed;
    if theta_odom(i) > pi
       theta_odom(i) = theta_odom(i) - 2*pi;
    else
        if theta_odom(i) < -pi
            theta_odom(i) = theta_odom(i) + 2*pi;
        end
    end
    x_odom(i) = x_odom(i-1) + cos(theta_odom(i-1)) * v_odom(i-1) *
time_elapsed;
    y_odom(i) = y_odom(i-1) + sin(theta_odom(i-1)) * v_odom(i-1) *
time_elapsed;
end
% ------end of your wheel odometry algorithm-------

% plot the results for verification
figure(1)
clf;

subplot(2,2,1);
hold on;
plot(x_true,y_true,'b');
plot(x_odom, y_odom, 'r');
legend('true', 'odom');
xlabel('x [m]');
ylabel('y [m]');
title('path');
axis equal;

subplot(2,2,2);
hold on;
plot(t_true,theta_true,'b');
plot(t_odom,theta_odom,'r');
legend('true', 'odom');
xlabel('t [s]');
ylabel('theta [rad]');
title('heading');

subplot(2,2,3);
hold on;
pos_err = zeros(numodom,1);
```

```
for i=1:numodom
    pos_err(i) = sqrt((x_odom(i)-x_true(i))^2 + (y_odom(i)-y_true(i))^2);
end
plot(t_odom,pos_err,'b');
xlabel('t [s]');
ylabel('distance [m]');
title('position error (odom-true)');

subplot(2,2,4);
hold on;
theta_err = zeros(numodom,1);
for i=1:numodom
    phi = theta_odom(i) - theta_true(i);
    while phi > pi
        phi = phi - 2*pi;
    end
    while phi < -pi
        phi = phi + 2*pi;
    end
    theta_err(i) = phi;
end
plot(t_odom,theta_err,'b');
xlabel('t [s]');
ylabel('theta [rad]');
title('heading error (odom-true)');
print -dpng ass1_q1.png


% =================================================================
% Question 2: add noise to data and re-run wheel odometry algorithm
% =================================================================
%
% Now we're going to deliberately add some noise to the linear and
% angular velocities to simulate what real wheel odometry is like.  Copy
% your wheel odometry algorithm from above into the indicated place below
% to see what this does.  The below loops 100 times with different random
% noise.  See the plot 'ass1_q2_soln.pdf' for what your results should look
% like.

% save the original odometry variables for later use
v_odom_noisefree = v_odom;
omega_odom_noisefree = omega_odom;

% set up plot
figure(2);
clf;
hold on;

% loop over random trials
for n=1:100

    % add noise to wheel odometry measurements (yes, on purpose to see
effect)
    v_odom = v_odom_noisefree + 0.2*randn(numodom,1);
    omega_odom = omega_odom_noisefree + 0.04*randn(numodom,1);

    % ------insert your wheel odometry algorithm here-------
    for i=2:numodom
        time_elapsed = t_odom(i) - t_odom(i-1);
        theta_odom(i) = theta_odom(i-1) + omega_odom(i-1) * time_elapsed;
        if theta_odom(i) > pi
           theta_odom(i) = theta_odom(i) - 2*pi;
        else
            if theta_odom(i) < -pi
                theta_odom(i) = theta_odom(i) + 2*pi;
            end
        end
        x_odom(i) = x_odom(i-1) + cos(theta_odom(i-1)) * v_odom(i-1) *
time_elapsed;
        y_odom(i) = y_odom(i-1) + sin(theta_odom(i-1)) * v_odom(i-1) *
time_elapsed;
    end
    % ------end of your wheel odometry algorithm-------

    % add the results to the plot
    plot(x_odom, y_odom, 'r');
end

% plot ground truth on top and label
plot(x_true,y_true,'b');
xlabel('x [m]');
ylabel('y [m]');
title('path');
axis equal;
print -dpng ass1_q2.png


% =================================================================
% Question 3: build a map from noisy and noise-free wheel odometry
% =================================================================
%
% Now we're going to try to plot all the points from our laser scans in the
% robot's initial reference frame.  This will involve first figuring out
% how to plot the points in the current frame, then transforming them back
% to the initial frame and plotting them.  Do this for both the ground
% truth pose (blue) and also the last noisy odometry that you calculated in
% Question 2 (red).  At first even the map based on the ground truth may
% not look too good.  This is because the laser timestamps and odometry
% timestamps do not line up perfectly and you'll need to interpolate.  Even
% after this, two additional patches will make your map based on ground
% truth look as crisp as the one in 'ass1_q3_soln.png'.  The first patch is
% to only plot the laser scans if the angular velocity is less than
% 0.1 rad/s; this is because the timestamp interpolation errors have more
% of an effect when the robot is turning quickly.  The second patch is to
% account for the fact that the origin of the laser scans is about 10 cm
% behind the origin of the robot.  Once your ground truth map looks crisp,
% compare it to the one based on the odometry poses, which should be far
% less crisp, even with the two patches applied.

% set up plot
figure(3);
clf;
hold on;
```

```
% precalculate some quantities
npoints = size(y_laser,2);
k_angles = linspace(phi_min_laser, phi_max_laser, npoints);
k_cos_angles = cos(k_angles);
k_sin_angles = sin(k_angles);
x_map = linspace(1, 1036801);
y_map = linspace(1, 1036801);


for n=1:2
    current_point = 1;
    if n==1
        % interpolate the noisy odometry at the laser timestamps
        t_interp = linspace(t_odom(1),t_odom(numodom),numodom);
        x_interp = interp1(t_interp,x_odom,t_laser);
        y_interp = interp1(t_interp,y_odom,t_laser);
        theta_interp = interp1(t_interp,theta_odom,t_laser);
        omega_interp = interp1(t_interp,omega_odom,t_laser);
    else
        % interpolate the noise-free odometry 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);
    end

    % loop over laser scans
    for i=1:size(t_laser,1)
        % ------insert your point transformation algorithm here------
        if abs(omega_interp(i)) >= 0.1
            continue
        end
        for k=1:size(y_laser,2)
           % calculating where the point is in the robot frame
           k_range = y_laser(i, k);
           point_robot_frame = [
               k_range * k_cos_angles(k);
               k_range * k_sin_angles(k);
               0;
               1
               ];
           CIR = [
                cos(theta_interp(i)) -sin(theta_interp(i)) 0 x_interp(i) -
0.1*cos(theta_interp(i));
                sin(theta_interp(i)) cos(theta_interp(i)) 0 y_interp(i) -
0.1*sin(theta_interp(i));
                    0 0 1 0;
                    0 0 0 1
           ];
           point_intertial_frame = CIR * point_robot_frame;
           x_map(current_point) = point_intertial_frame(1);
           y_map(current_point) = point_intertial_frame(2);
           current_point = current_point + 1;
        end
        % ------end of your point transformation algorithm-------
    end
    if n==1
        plot(x_map, y_map, 'r.');
    else
        plot(x_map, y_map, 'b.');
    end

end
legend('map with noise', 'map without noise');
xlabel('x [m]');
ylabel('y [m]');
title('map');
axis equal;
print -dpng ass1_q3.png
```