ROB521 Assignment 1: Wheel Odometry and Mapping

Samuel Atkins | 1002951754

February 23, 2020

Table of Contents

[Part 1: Noise-free Wheel Odometry 2](#_Toc33276412)

[Part 2: Wheel Odometry with Added Noise 3](#_Toc33276413)

[Part 3: Building a Map 4](#_Toc33276414)

[Appendix: MATLAB Code 5](#_Toc33276415)

Table of Figures

[Figure 1: Part 1 true path and heading versus predicted path and heading 2](#_Toc33276013)

[Figure 2: Part 2 predicted path with added noise 3](#_Toc33276014)

[Figure 3: Part 3 generated map with and without noise modelling 4](#_Toc33276015)

# 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:

Equation : 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:

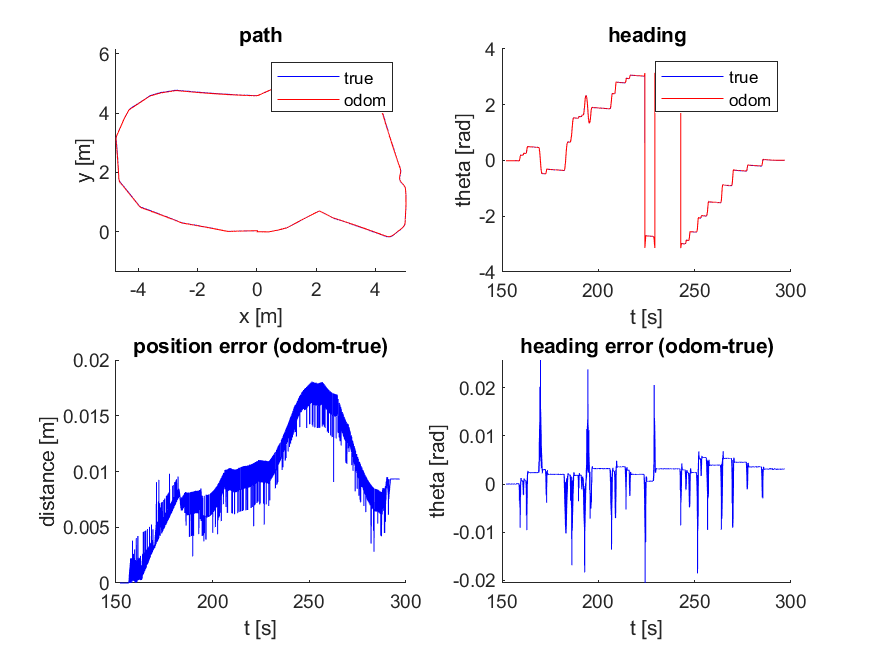


Figure : Part 1 true path and heading versus predicted path and heading

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:

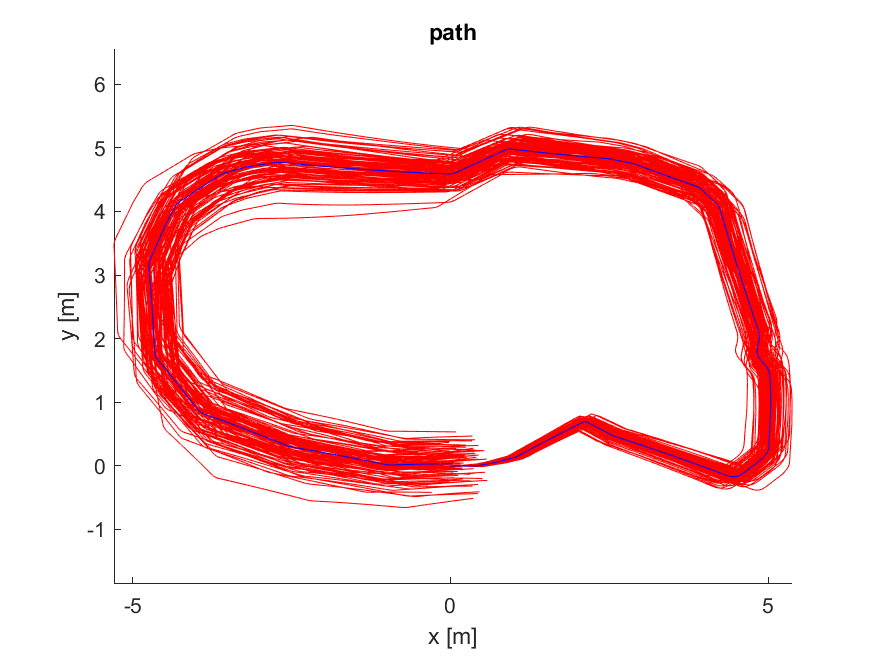


Figure : Part 2 predicted path with added noise

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 : 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:

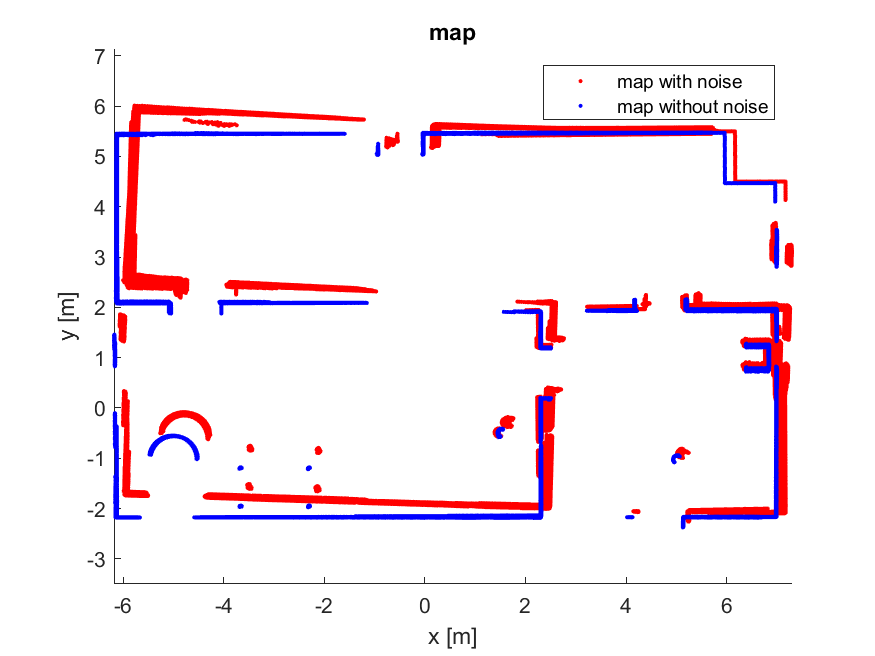


Figure : Part 3 generated map with and without noise modelling

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