

ASSIGNMENT 1

ROB521: Mobile Robotics and Perception
Jojo (Yizhi) Zhou, 1003002396

January 24, 2020

1 Noise-free Wheel Odometry

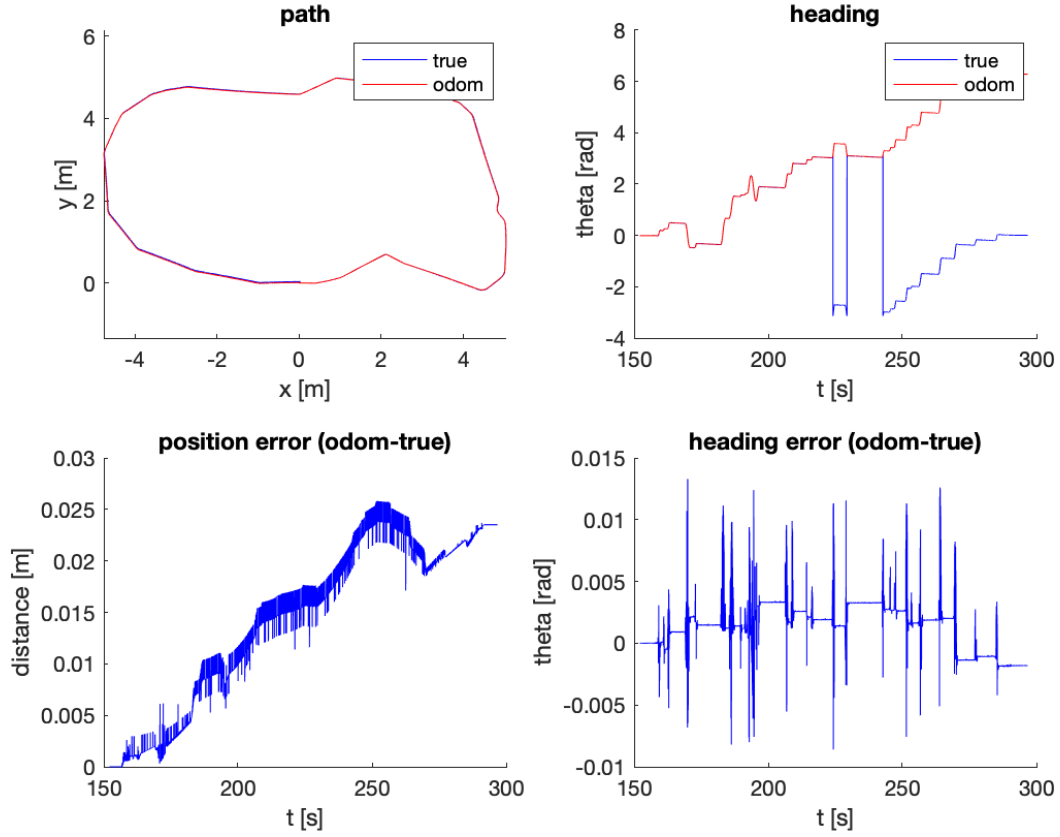


Figure 1: Output plot from to Question 1

The algorithm estimates the pose of robot using wheel odometry data, which is considered **noise-free** (i.e., no slipping and no measurement precision error in the encoder) in this part.

- **Position.** From the path plot, the ground truth and odometry are nearly identical. The position error ranging from 0 in the beginning to about 2.6 cm at worst and grows (roughly) proportionally. That being said, the error in centimeters is negligible compared to our scale of position in meters.
- **Heading.** The heading is quite accurate in the beginning as well, until around 230 seconds where there is a small period of offset. The same offset (i.e., $\sim 2\pi$) occurred after 250 as well. Since the offset is essentially a full round, the offset is unlikely to be caused by an error; instead, it is likely due to the way that ROS/Gazebo/Turtlebot is designed, which seems to favor smaller angle changes (i.e., increment by $\delta\theta$ instead of $\delta\theta - 2\pi$ if $|\delta\theta| < |\delta\theta - 2\pi|$). However, none of such offset (or any offset of an integer number of full circles) would have an impact when we determine the pose of robot, as θ and $\theta \pm 2\pi$ are practically equal to each other.

From the error plot, we can see that the 2π offset was indeed not taken into account. Instead, we get a bunch of small errors. Through comparison with the heading plot, we can see that these small errors occur at sudden direction changes - the larger the increment/decrement in heading, the larger the error. This totally makes sense as the robot might needs to overshoot a little bit, physically, when as it changes its heading.

2 Noisy Wheel Odometry

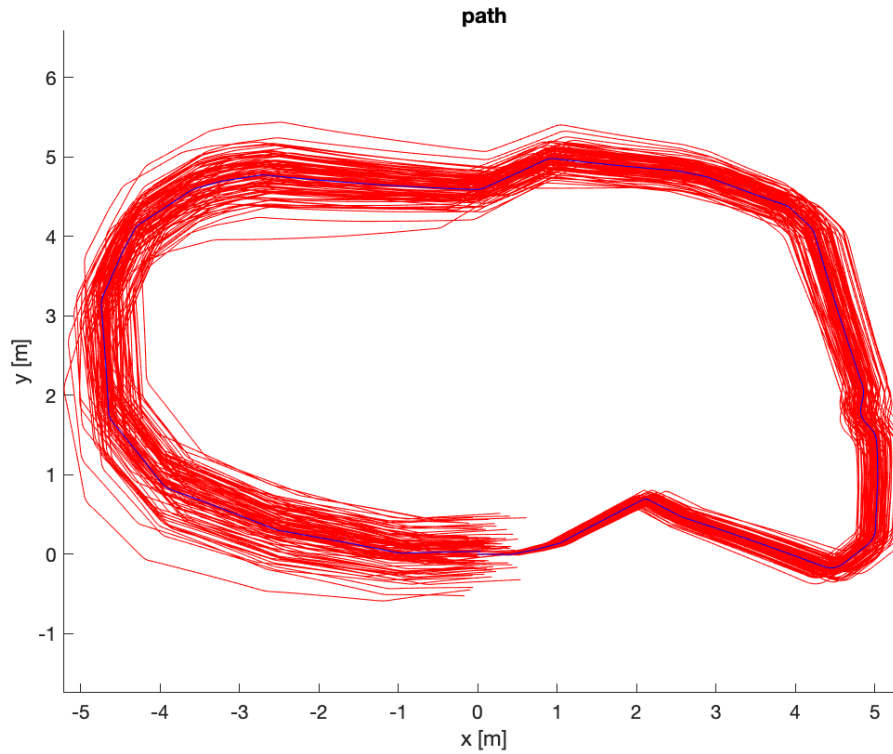


Figure 2: Output plot from to Question 2

The algorithm in this part remains the same as that in the previous part, with an addition of noise that simulates the real wheel odometry. We randomly sample 100 instances with from such noise and plot the 100 resulting paths in Figure 2. A few observations are worth being discussed:

- Obviously, the paths starts at the same point as the ground truth and continues to diverge away from it, while the general trend still follows the ground truth.
- Mathematically speaking, as `randn` retunes scalars drawn from the standard *normal distribution*, the random instances we generated have the noise-free velocities (i.e., same as ground truth) as mean values. Therefore, the expected path should indeed match the ground truth one, while the **error propagates**.
- Unlike part 1, this the result produced in this part doesn't exactly match the reference solution, which makes sense since the samples are drawn randomly.

3 Map from Odometry

For the noisy odometry data, the laser data gets more uncertain the longer the robot travels. This is expected as the odometry is a form of dead reckoning so the variance/uncertainty grows without bound. We can see growing errors in both the mean, as the odometry data is offset from the ground truth and sometime duplicated, and the variance, as the odometry landmarks are significantly thicker which indicates higher uncertainty. While the two patches for angular velocity and laser position offset was

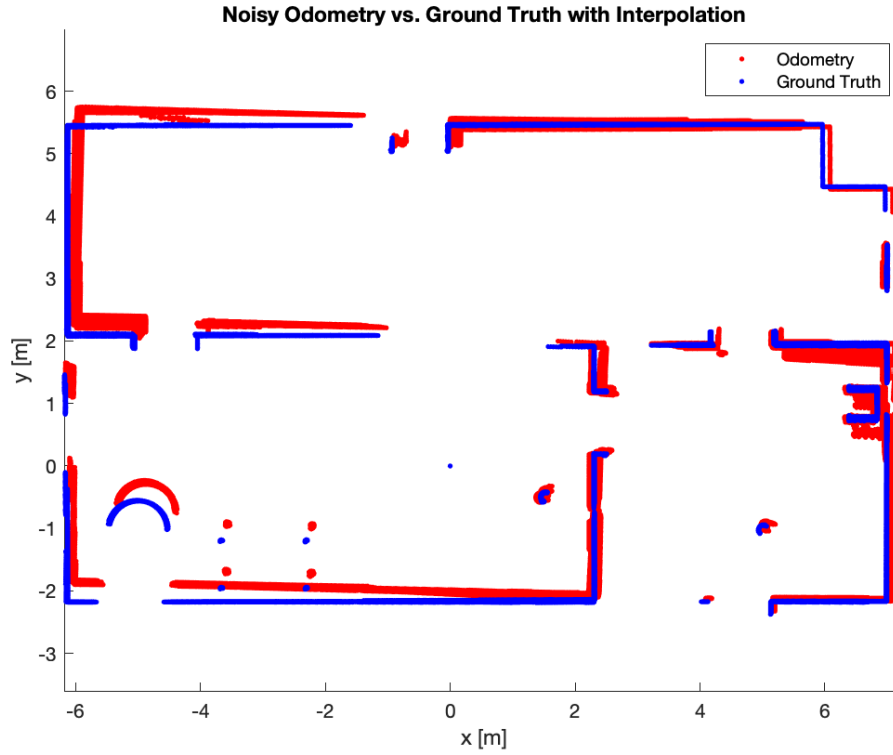


Figure 3: Output plot from to Question 3

applied, the resulting solution is still not as crisp as the sample solution. This does not appear to be simply a plotting error as the top-left corner is doubled in both plots, indicating some pose estimation errors in my algorithm.

Appendix: Source Code

```
1 % =====
2 % ass1.m
3 % =====
4 %
5 % This assignment will introduce you to the idea of estimating the motion
6 % of a mobile robot using wheel odometry, and then also using that wheel
7 % odometry to make a simple map. It uses a dataset previously gathered in
8 % a mobile robot simulation environment called Gazebo. Watch the video,
9 % 'gazebo.mp4' to visualize what the robot did, what its environment
10 % looks like, and what its sensor stream looks like.
11 %
12 % There are three questions to complete (5 marks each):
13 %
14 %     Question 1: code (noise-free) wheel odometry algorithm
15 %     Question 2: add noise to data and re-run wheel odometry algorithm
16 %     Question 3: build a map from ground truth and noisy wheel odometry
17 %
18 % Fill in the required sections of this script with your code, run it to
19 % generate the requested plots, then paste the plots into a short report
20 % that includes a few comments about what you've observed. Append your
21 % version of this script to the report. Hand in the report as a PDF file.
22 %
23 % requires: basic Matlab, 'gazebo.mat'
24 %
25 % T D Barfoot, December 2015
26 %
27 clear all;
28
29 % set random seed for repeatability
30 rng(1);
31
32 % =====
33 % load the dataset from file
34 % =====
35 %
36 %     ground truth poses: t_true x_true y_true theta_true
37 %     odometry measurements: t_odom v_odom omega_odom
38 %     laser scans: t_laser y_laser
39 %     laser range limits: r_min_laser r_max_laser
40 %     laser angle limits: phi_min_laser phi_max_laser
41 %
42 load gazebo.mat;
43
44 % =====
45 % Question 1: code (noise-free) wheel odometry algorithm
46 % =====
47 %
```

```

48 % Write an algorithm to estimate the pose of the robot throughout motion
49 % using the wheel odometry data (t_odom, v_odom, omega_odom) and assuming
50 % a differential-drive robot model. Save your estimate in the variables
51 % (x_odom y_odom theta_odom) so that the comparison plots can be generated
52 % below. See the plot 'ass1_q1_soln.png' for what your results should look
53 % like.
54
55 % variables to store wheel odometry pose estimates
56 numodom = size(t_odom,1);
57 x_odom = zeros(numodom,1);
58 y_odom = zeros(numodom,1);
59 theta_odom = zeros(numodom,1);
60
61 % set the initial wheel odometry pose to ground truth
62 x_odom(1) = x_true(1);
63 y_odom(1) = y_true(1);
64 theta_odom(1) = theta_true(1);
65
66 % -----insert your wheel odometry algorithm here-----
67 for i=2:numodom
68     % Note: we update the ith as current using (i-1) as previous
69     h = t_odom(i) - t_odom(i-1);
70     theta_odom(i) = theta_odom(i-1) + omega_odom(i) * h;
71     x_odom(i) = x_odom(i-1) + v_odom(i) * cos(theta_odom(i)) * h;
72     y_odom(i) = y_odom(i-1) + v_odom(i) * sin(theta_odom(i)) * h;
73 end
74 % -----end of your wheel odometry algorithm-----
75
76 % plot the results for verification
77 figure(1)
78 clf;
79
80 subplot(2,2,1);
81 hold on;
82 plot(x_true,y_true,'b');
83 plot(x_odom, y_odom, 'r');
84 legend('true', 'odom');
85 xlabel('x [m]');
86 ylabel('y [m]');
87 title('path');
88 axis equal;
89
90 subplot(2,2,2);
91 hold on;
92 plot(t_true,theta_true,'b');
93 plot(t_odom,theta_odom,'r');
94 legend('true', 'odom');
95 xlabel('t [s]');
96 ylabel('theta [rad]');

```

```

97 title('heading');
98
99 subplot(2,2,3);
100 hold on;
101 pos_err = zeros(numodom,1);
102 for i=1:numodom
103     pos_err(i) = sqrt((x_odom(i)-x_true(i))^2 + (y_odom(i)-y_true(i))^2);
104 end
105 plot(t_odom,pos_err,'b');
106 xlabel('t [s]');
107 ylabel('distance [m]');
108 title('position error (odom-true)');
109
110 subplot(2,2,4);
111 hold on;
112 theta_err = zeros(numodom,1);
113 for i=1:numodom
114     phi = theta_odom(i) - theta_true(i);
115     while phi > pi
116         phi = phi - 2*pi;
117     end
118     while phi < -pi
119         phi = phi + 2*pi;
120     end
121     theta_err(i) = phi;
122 end
123 plot(t_odom,theta_err,'b');
124 xlabel('t [s]');
125 ylabel('theta [rad]');
126 title('heading error (odom-true)');
127 print -dpng ass1_q1.png
128
129 %%
130 % =====
131 % Question 2: add noise to data and re-run wheel odometry algorithm
132 % =====
133 %
134 % Now we're going to deliberately add some noise to the linear and
135 % angular velocities to simulate what real wheel odometry is like. Copy
136 % your wheel odometry algorithm from above into the indicated place below
137 % to see what this does. The below loops 100 times with different random
138 % noise. See the plot 'ass1_q2_soln.pdf' for what your results should look
139 % like.
140
141 % save the original odometry variables for later use
142 v_odom_noisefree = v_odom;
143 omega_odom_noisefree = omega_odom;
144
145 % set up plot

```

```

146 figure(2);
147 clf;
148 hold on;
149
150 % loop over random trials
151 for n=1:100
152
153     % add noise to wheel odometry measurements (yes, on purpose to see effect)
154     v_odom = v_odom_noisefree + 0.2*randn(numodom,1);
155     omega_odom = omega_odom_noisefree + 0.04*randn(numodom,1);
156
157     % -----insert your wheel odometry algorithm here-----
158     for i=2:numodom
159         % Same as Q1
160         h = t_odom(i) - t_odom(i-1);
161         theta_odom(i) = theta_odom(i-1) + omega_odom(i) * h;
162         x_odom(i) = x_odom(i-1) + v_odom(i) * cos(theta_odom(i)) * h;
163         y_odom(i) = y_odom(i-1) + v_odom(i) * sin(theta_odom(i)) * h;
164
165     end
166     % -----end of your wheel odometry algorithm-----
167
168     % add the results to the plot
169     plot(x_odom, y_odom, 'r');
170 end
171
172 % plot ground truth on top and label
173 plot(x_true,y_true,'b');
174 xlabel('x [m]');
175 ylabel('y [m]');
176 title('path');
177 axis equal;
178 print -dpng ass1_q2.png
179
180 %%
181 % =====
182 % Question 3: build a map from noisy and noise-free wheel odometry
183 % =====
184 %
185 % Now we're going to try to plot all the points from our laser scans in the
186 % robot's initial reference frame. This will involve first figuring out
187 % how to plot the points in the current frame, then transforming them back
188 % to the initial frame and plotting them. Do this for both the ground
189 % truth pose (blue) and also the last noisy odometry that you calculated in
190 % Question 2 (red). At first even the map based on the ground truth may
191 % not look too good. This is because the laser timestamps and odometry
192 % timestamps do not line up perfectly and you'll need to interpolate. Even
193 % after this, two additional patches will make your map based on ground
194 % truth look as crisp as the one in 'ass1_q3_soln.png'. The first patch is

```



```

195 % to only plot the laser scans if the angular velocity is less than
196 % 0.1 rad/s; this is because the timestamp interpolation errors have more
197 % of an effect when the robot is turning quickly. The second patch is to
198 % account for the fact that the origin of the laser scans is about 10 cm
199 % behind the origin of the robot. Once your ground truth map looks crisp,
200 % compare it to the one based on the odometry poses, which should be far
201 % less crisp, even with the two patches applied.
202
203 % set up plot
204 figure(3);
205 clf;
206 hold on;
207
208 % precalculate some quantities
209 npoints = size(y_laser,2);
210 angles = linspace(phi_min_laser, phi_max_laser, npoints);
211 cos_angles = cos(angles);
212 sin_angles = sin(angles);
213
214 % initialize some quantities for future use
215 one = ones(1, npoints);
216 x_i = zeros(1, size(y_laser,1)*size(y_laser,2));
217 y_i = zeros(1, size(y_laser,1)*size(y_laser,2));
218
219 for n=1:2
220
221     if n==1
222         % interpolate the noisy odometry at the laser timestamps
223         t_interp = linspace(t_odom(1),t_odom(numodom),numodom);
224         x_interp = interp1(t_interp,x_odom,t_laser);
225         y_interp = interp1(t_interp,y_odom,t_laser);
226         theta_interp = interp1(t_interp,theta_odom,t_laser);
227         omega_interp = interp1(t_interp,omega_odom,t_laser);
228     else
229         % interpolate the noise-free odometry at the laser timestamps
230         t_interp = linspace(t_true(1),t_true(numodom),numodom);
231         x_interp = interp1(t_interp,x_true,t_laser);
232         y_interp = interp1(t_interp,y_true,t_laser);
233         theta_interp = interp1(t_interp,theta_true,t_laser);
234         omega_interp = interp1(t_interp,omega_odom,t_laser);
235     end
236
237 % loop over laser scans
238 for i=1:size(t_laser,1)
239
240     % -----insert your point transformation algorithm here-----
241     % only plot the laser scans if angular velocity < 0.1 rad/s
242     if abs(omega_interp(i)) < 0.1
243         % Transform from laser frame -> vehicle frame

```

```

244     x_v = y_laser(i,:) .* cos_angles;
245     y_v = y_laser(i,:) .* sin_angles;
246
247     % Define homogeneous transformation H_{iv: vehicle -> inertial}
248     sin_ang = sin(theta_interp(i));
249     cos_ang = cos(theta_interp(i));
250     H_iv = [cos_ang -sin_ang x_interp(i)-0.1*cos_ang;
251             sin_ang  cos_ang y_interp(i)-0.1*sin_ang;
252             0         0         1];
253
254     % Transform from vehicle frame -> inertial frame
255     xy_i = H_iv*[x_v; y_v; one];
256
257     % Record into the plot
258     x_i((i-1)*npoints+1 : i*npoints) = xy_i(1, :);
259     y_i((i-1)*npoints+1 : i*npoints) = xy_i(2, :);
260 end
261 % -----end of your point transformation algorithm-----
262
263 end
264 if n == 1
265     plot(x_i, y_i, 'r.')
266 else
267     plot(x_i, y_i, 'b.')
268 end
269 end
270
271 % Plot laser data
272 xlabel('x [m]');
273 ylabel('y [m]');
274 legend('Odometry','Ground Truth')
275 title('Noisy Odometry vs. Ground Truth with Interpolation');
276
277 axis equal;
278 print -dpng ass1_q3.png

```