ASSIGNMENT 1

ROB521: Mobile Robotics and Perception

Jojo (Yizhi) Zhou, 1003002396

January 23, 2020

1 Noise-free Wheel Odometry

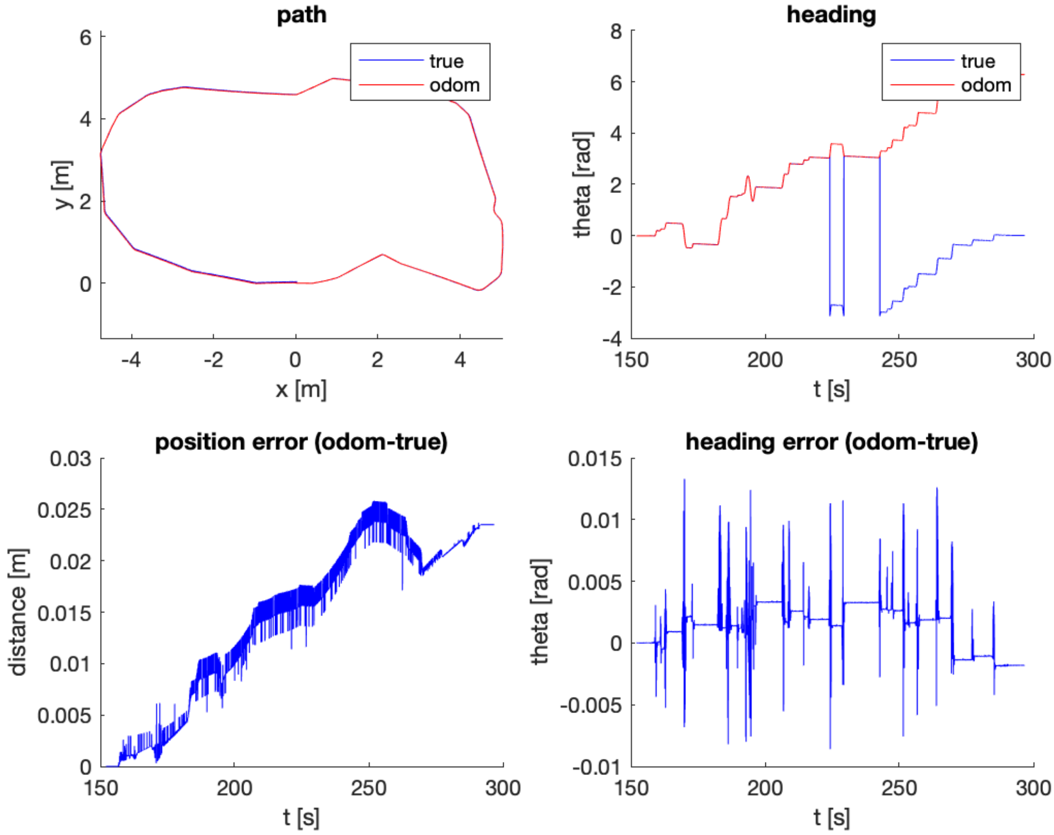

Figure 1: Output plot from to Question 1

The plot of the position is identical to the ground. As expected from noise free data, there is little to none uncertainty in the wheel encoder readings. There is very little offset from the true path taken, though the position error does still increase over time. Most of the error likely comes from the finite nature of the data as there is no noise or slippage in the encoder readings. The most notable error is the heading of the robot which diverges from the ground truth at the middle and end of the trajectory. This is likely due to the way the robot measures the ground truth heading. Since the odometry algorithm is incremental, it is impossible for the heading to suddenly jump by two pi. The measured, true heading on the other hand is effected by the specifications of the sensor or Gazebo software, allowing discrete jumps of two pi in the heading. Compared to the solution image, the position pretty much identical to the solution image. The most notable difference is the heading; where as my solution differs from the ground truth by two pi, the sample solution follows the true heading perfectly. This indicates a difference in the sample solution algorithm implementation, through for the purposes of the calculated trajectory this is not important.

2 Noisy Wheel Odometry

As we can see from the numerous odometry runs, the divergence of the odometry pose from the true path grows over distance travel. While the mean of the 100 runs is centered around the ground truth,

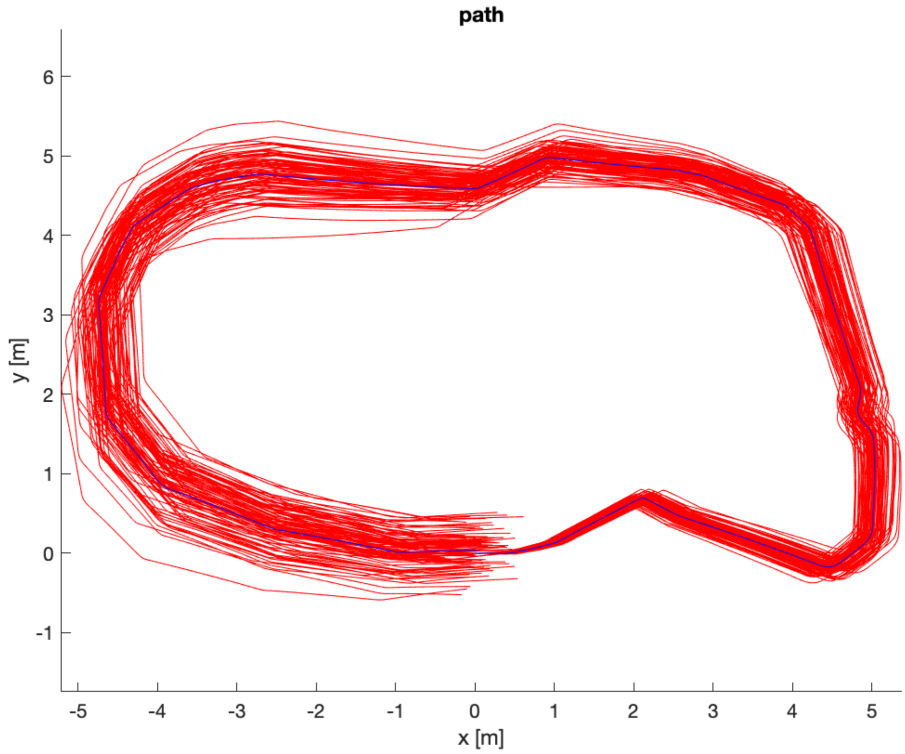

Figure 2: Output plot from to Question 2

the final position error for some of the paths is close to 1 meter, indicating high uncertainty. If we view the multiple runs as particles, we see the uncertainty grow without bound as the particles spread further apart and there is no way to tell which of the possible trajectories is the correct one. The image looks identical the sample solution image so no comments will be made.

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 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.

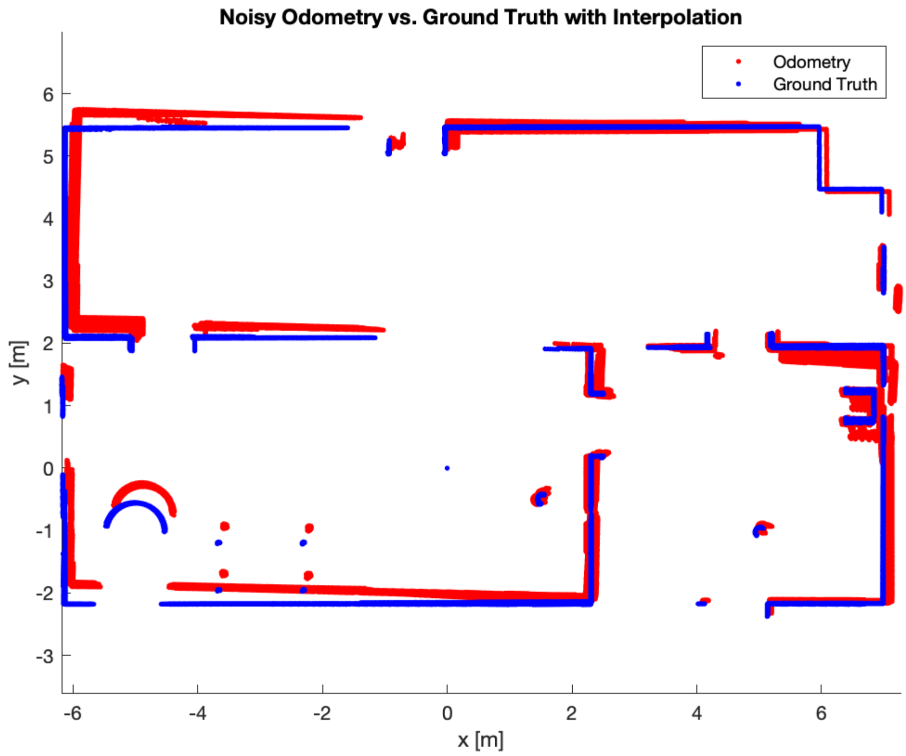

Figure 3: Output plot from to Question 3

Appendix: Source 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
  % Note: we update the ith as current using (i-1) as previous
    h = t_odom(i) - t_odom(i-1);
    theta_odom(i) = theta_odom(i-1) + omega_odom(i) * h;
    x_odom(i) = x_odom(i-1) + v_odom(i) * cos(theta_odom(i)) * h;
    y_odom(i) = y_odom(i-1) + v_odom(i) * sin(theta_odom(i)) * h;
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
      % Same as Q1
      h = t_odom(i) - t_odom(i-1);
      theta_odom(i) = theta_odom(i-1) + omega_odom(i) * h;
      x_odom(i) = x_odom(i-1) + v_odom(i) * cos(theta_odom(i)) * h;
      y_odom(i) = y_odom(i-1) + v_odom(i) * sin(theta_odom(i)) * h;

    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);
angles = linspace(phi_min_laser, phi_max_laser, npoints);
cos_angles = cos(angles);
sin_angles = sin(angles);

% initialize some quantities for future use
one = ones(1, npoints);
x_i = zeros(1, size(y_laser,1)*size(y_laser,2));
y_i = zeros(1, size(y_laser,1)*size(y_laser,2));

for n=1:2

    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------
        % only plot the laser scans if angular velocity < 0.1 rad/s
        if abs(omega_interp(i)) < 0.1
            %  Transform from laser frame -> vehicle frame
```

```
            x_v = y_laser(i,:) .* cos_angles;
            y_v = y_laser(i,:) .* sin_angles;

          % Define homogeneous transformation H_{iv: vehicle -> inertial}
          sin_ang = sin(theta_interp(i));
          cos_ang = cos(theta_interp(i));
            H_iv = [cos_ang -sin_ang x_interp(i)-0.1*cos_ang;
                  sin_ang  cos_ang y_interp(i)-0.1*sin_ang;
                  0      0     1];

            % Transform from vehicle frame -> inertial frame
            xy_i = H_iv*[x_v; y_v; one];

            % Record into the plot
            x_i((i-1)*npoints+1 : i*npoints) = xy_i(1, :);
            y_i((i-1)*npoints+1 : i*npoints) = xy_i(2, :);
        end
        % ------end of your point transformation algorithm-------

    end
    if n == 1
        plot(x_i, y_i, 'r.')
    else
        plot(x_i, y_i, 'b.')
    end
end

% Plot laser data
xlabel('x [m]');
ylabel('y [m]');
legend('Odometry','Ground Truth')
title('Noisy Odometry vs. Ground Truth with Interpolation');

axis equal;
print -dpng ass1_q3.png
```