ROB521: Mobile Robotics and Perception
Assignment #2: Wheel Odometry

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

The first question consisted of estimating the pose $(x_{odom}, y_{odom}, \theta_{odom})$ of a differential-drive robot using noise-free wheel odometry measurements $(t_{odom}, v_{odom}, \omega_{odom})$. The pose was estimated using the following equations:

$$\theta_{odom}(i) = \theta_{odom}(i-1) + [h \times \omega_{odom}(i)]$$
$$x_{odom}(i) = x_{odom}(i-1) + [h \times cos(\theta_{odom}(i)) \times v_{odom}(i))]$$
$$y_{odom}(i) = y_{odom}(i-1) + [h \times sin(\theta_{odom}(i)) \times v_{odom}(i))]$$

where $h = t_{odom}(i) - t_{odom}(i-1)$ and θ_{odom} was constrained within the $-\pi$ to π range.

Comparison plots were generated to compare the pose estimates and ground truth poses as shown in Figure 1.

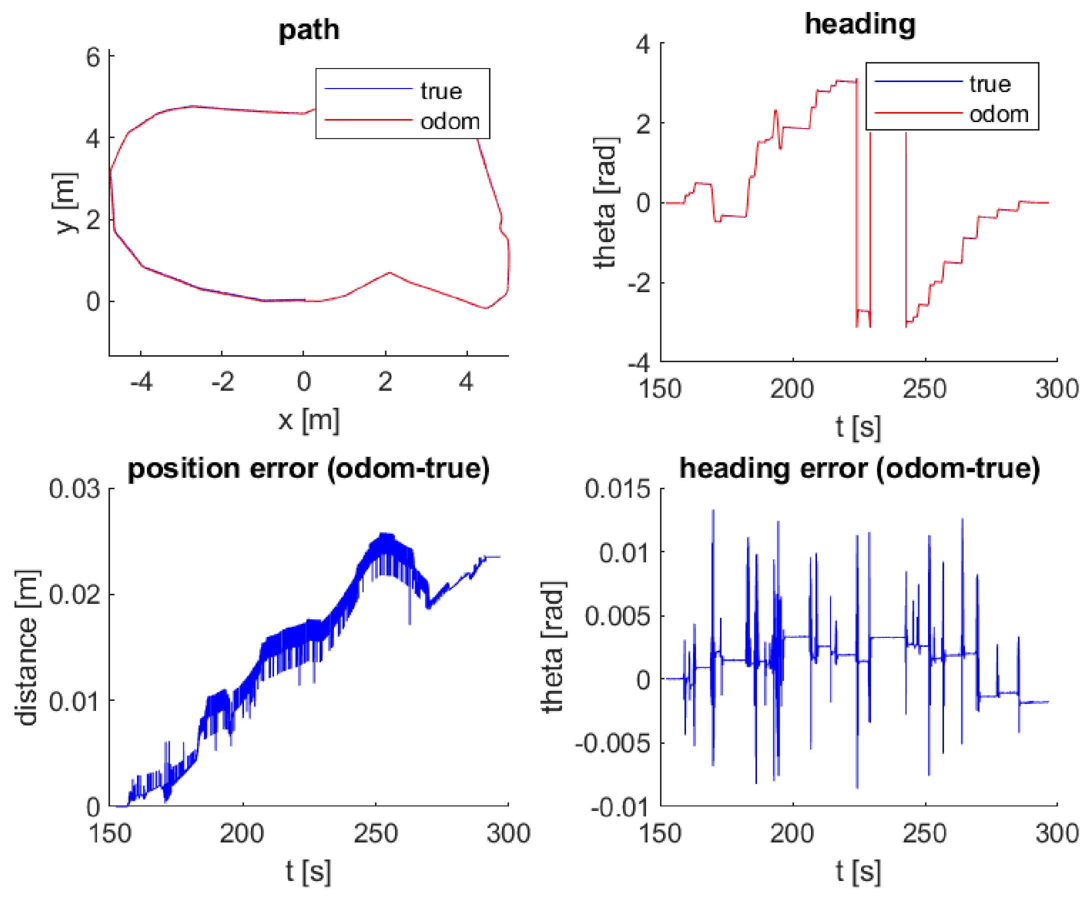

Figure 1: *Plots comparing ground truth poses and pose estimates determined using noise-free wheel odometry measurements*

From the top left and top right subplots in Figure 1, we can see that the estimated path and heading is the same as the true path and heading to the point where they are overlapping. The bottom subplots in Figure 1 show the position error remains less than 0.03 m (or less than 3 cm) and the heading error hovers around 0 radians. Evidently, the pose estimation is accurate because the odometry measurements used to estimate the pose were noise-free. In real life, wheel odometry data is corrupted with noise. It can also be observed that the position error seems to accumulate, whereas the heading error hovers around 0 radians.

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

For the second question, noise was deliberately added to the linear and angular velocities in an attempt to simulate what real wheel odometry data would be like. The same pose estimation code from question 1 was used on wheel odometry data that was corrupted with 100 different random noise simulations.

Figure 2 shows the path from ground truth poses in blue and the estimated poses obtained from noisy wheel odometry data in red.

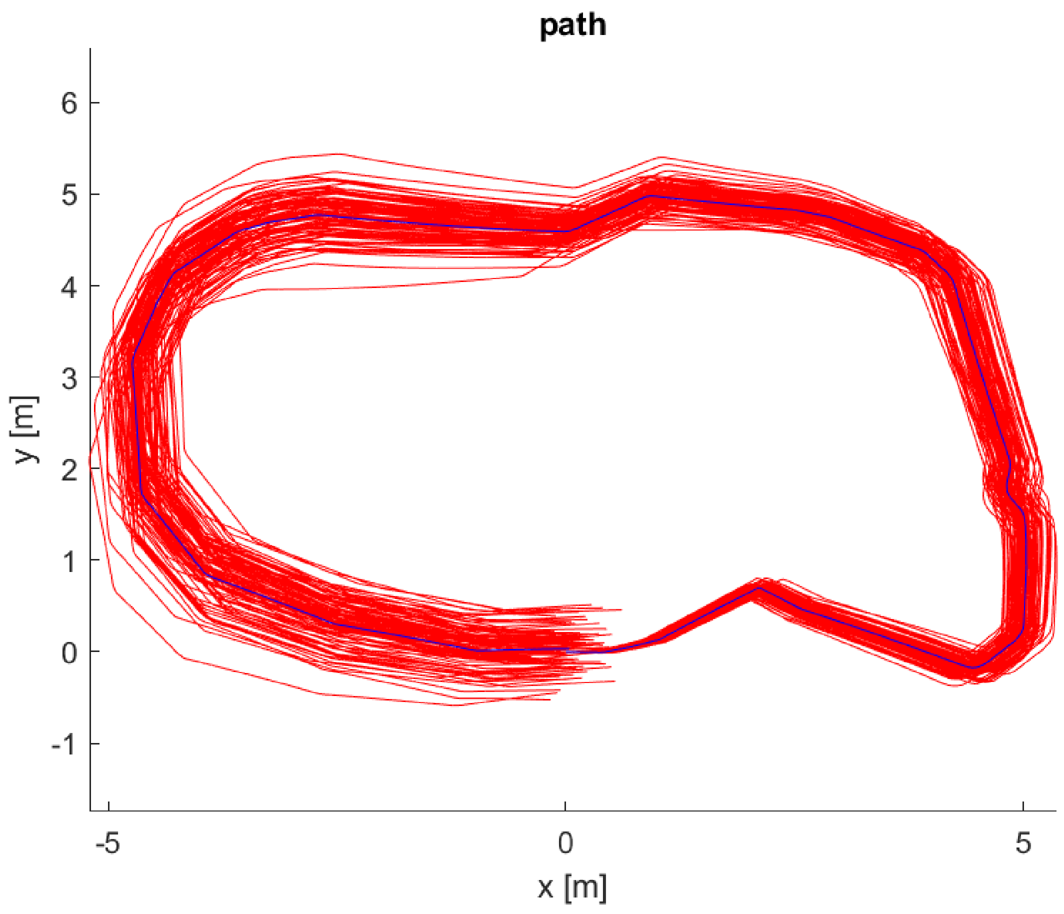

Figure 2: *Path from ground truth poses (blue) and estimated poses obtained from 100 different simulations of noisy wheel odometry data (red)*

From Figure 2, it is evident that the error between the true path in blue and the estimated paths in red increases as the robot moves along the path. At the starting of the path, the estimated poses closely resemble the true path, but towards the end of the path the estimates are further away and more scattered.

<u>Question 3: Build a map from ground truth and noisy wheel odometry</u>

In question 3, the laser scans were plotted in the robot's inertial frame to build a map of the environment the robot travels in.

Since the laser timestamps and odometry timestamps don't align well, they were interpolated (already given in the code). The laser scans were only plotted if the angular velocity was less than 0.1 rad/s since the timestamp interpolation errors have less effect when the robot turns quickly. First, the laser points were determined in the current frame as follows:

$$\begin{bmatrix} y_{laser}(i,:).*cos\_angles \\ y_{laser}(i,:).*sin\_angles \\ zeros(1, size(y_{laser}, 2)) \end{bmatrix}$$

where i represents each of the laser scans

These points were transformed to the inertial frame using the following rotation matrix, C and translation vector, r. The translation vector accounts for the fact that the origin of the laser scans is about 10 cm behind the origin of the robot.

$$C = \begin{bmatrix} cos(theta_{interp}(i)) & -sin(theta_{interp}(i)) & 0 \\ sin(theta_{interp}(i)) & cos(theta_{interp}(i)) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$r = \begin{bmatrix} x_{interp}(i) - 0.1 * cos(theta_{interp}(i)) \\ y_{interp}(i) - 0.1 * sin(theta_{interp}(i)) \\ 0 \end{bmatrix}$$

Figure 3 shows the resulting map. The blue map created from ground truth poses is thinner and more crisp than the red map produced from the noisy odometry data. From the provided video showing how the robot moves around the environment, it is visible that the robot begins in the bottom left room (closer to the right side of that room) and moves towards the bottom right room. The red map produced around the beginning of the robot's path aligns well with the ground truth blue map. However, as the robot progresses along the path, there are increasingly larger discrepancies between the red and blue maps since the errors between ground truth poses and estimated poses from wheel odometry data start building up. This is similar to what was observed in question 2 when plotting the path of the robot. For real world applications, this type of discrepancy will need to be accounted appropriately.

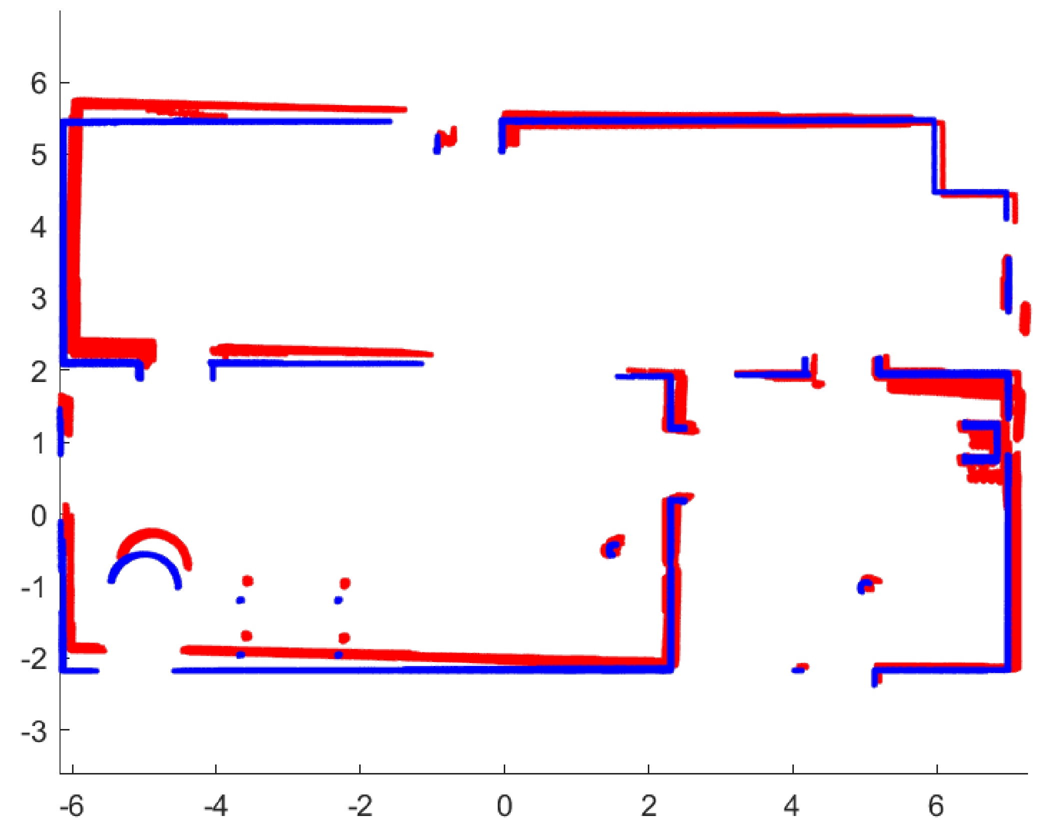

Figure 3: *Map built from ground truth poses (blue) and estimated poses using the last noisy wheel odometry data from question 2 (red)*

```
% ======
% ROB521_assignment2.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, 'ROB521_assignment2_gazebo_data.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 ROB521_assignment2_gazebo_data.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
    % determine time step and use equations from lecture
    h = t_odom(i) - t_odom(i-1);
    theta_odom(i) = theta_odom(i-1) + (h * omega_odom(i));

    % constrain the theta within the range
    if theta_odom(i) < -pi
        theta_odom(i) = theta_odom(i) + (2*pi);
    elseif theta_odom(i) > pi
        theta_odom(i) = theta_odom(i) - (2*pi);
    end

    x_odom(i) = x_odom(i-1) + (h * cos(theta_odom(i)) * v_odom(i));
    y_odom(i) = y_odom(i-1) + (h * sin(theta_odom(i)) * v_odom(i));
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
        % determine time step and use equations from lecture
        h = t_odom(i) - t_odom(i-1);
        theta_odom(i) = theta_odom(i-1) + (h * omega_odom(i));

        % constrain the theta within the range
        if theta_odom(i) < -pi
            theta_odom(i) = theta_odom(i) + (2*pi);
        elseif theta_odom(i) > pi
            theta_odom(i) = theta_odom(i) - (2*pi);
        end

        x_odom(i) = x_odom(i-1) + (h * cos(theta_odom(i)) * v_odom(i));
        y_odom(i) = y_odom(i-1) + (h * sin(theta_odom(i)) * v_odom(i));

    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);

for n=1:2
    map_x = [];
    map_y = [];
    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 laser scans if the angular velocity is less than 0.1 rad/s
        if abs(omega_interp(i)) < 0.1
            % define the transformation matrix to the account
            % for the fact that the origin of the laser scans is about 10 cm
            % behind the origin of the robot

            % rotation
            C = [cos(theta_interp(i)) -sin(theta_interp(i)) 0;
                sin(theta_interp(i)) cos(theta_interp(i)) 0;
                0 0 1];

            % translation
            r = [x_interp(i)-0.1*cos(theta_interp(i)); y_interp(i)-0.1*sin(theta_interp ↙
(i)); 0];

            % obtain points in the current frame
            pCurrent = [y_laser(i, :).*cos_angles; y_laser(i, :).*sin_angles; zeros(1, ↙
size(y_laser, 2))];

            % multiply points in robot frame by rotation matrix and add
            % translation vector to obtain points in intertial frame
            pInertial = (C * pCurrent) + r;

            % store for plotting
            map_x = [map_x; pInertial(1, :)'];
            map_y = [map_y; pInertial(2, :)'];
        end
        % ------end of your point transformation algorithm-------
    end
    if n==1
        plot(map_x, map_y, 'r.')
    else
        plot(map_x, map_y, 'b.')
    end
end

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