Alvin Pane

23 March 2022

ROB521 Assignment #2: Wheel Odometry and Mapping

Part 1: Noise-Free Wheel Odometry

The first part of this assignment required the implementation of an algorithm to estimate the pose of a robot throughout its motion using only the wheel odometry data collected at time-steps throughout its journey. The odometry data collected included time, velocity, and angular velocity at a specific point in time. A differential-drive robot model was considered in the implementation of the algorithm.

The algorithm assumes the initial starting position is equal to robot's ground truth position, essentially making the assumption that we know exactly where the robot is starting from. This assumption is not required, however. We could start with any guess about our initial position, although this would lead to error in the odometry path. It then uses an iterative approach, updating the estimate of the current time-step based on the estimate of the previous time-step using the following set of equations:

$$x(t) = x(t\text{-}1) + \Delta t * \cos(t\text{-}1) * v(t\text{-}1) \quad \textbf{(1)}$$
$$y(t) = y(t\text{-}1) + \Delta t * \sin(t\text{-}1) * v(t\text{-}1) \quad \textbf{(2)}$$
$$\theta(t) = \theta(t\text{-}1) + \Delta t * \omega(t\text{-}1) \quad \textbf{(3)}$$

Equations 1, 2, and 3 compute updates for the x-coordinate, y-coordinate, and theta pose at the current time-step, respectively. Δt is the time difference between the current time-step and the previous as measured by the odometry data. v(t-1) and ω(t-1) is the linear and angular velocity recorded at the previous time-step. These simple equations compute a new estimate of the current position based off the old position, the time elapsed, the robot heading, and its velocity in that direction. It is worth noting that all of the values calculated for θ(t) had to be mapped from the range [0 2π] to the range [-π π]. This was done using the MATLAB function WrapToPi(). This was necessary to map the angle to an appropriate robot heading and to account for a heading error produced by the fact that cosine takes the same value at -π and π. This was producing issues where when the robot was performing a complete turn, but the heading was not correctly recorded as per the sample solution. Figure 1 shows the MATLAB output for this algorithm when run on simulated robot data. Plotted is the true and odometry measured path vs. time, the robot heading vs. time, position error (odometry-true) vs. time, and heading error (odometry-true). We see that the odometry estimates of the path and heading are nearly identical, with very minimal error values in this noise-free case. The position error increases slightly over time as error begins to accumulate, while we do see much of this effect with the heading error.

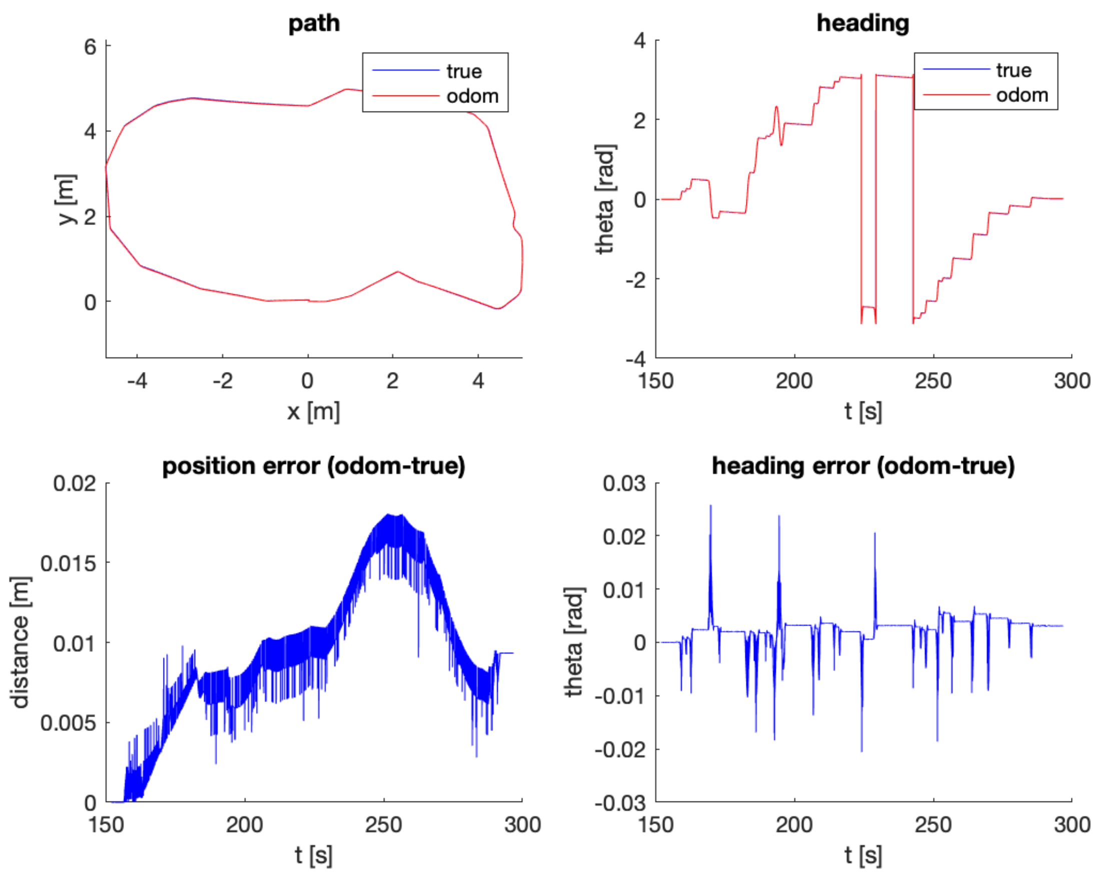


Figure 1: Outputs for Part 1

Part 2: Noisy Wheel Odometry

Part 2 builds off the algorithm introduced in Part 1, only this time injecting noise into the linear and angular velocities recorded by the odometry data. This injected noise simulates a real system where the sensor information will have some noise. 100 iterations of the algorithm from Part 1 are performed, with different random noise each time. The path as estimated by each odometry iteration (red) is plotted in Figure 2 against the ground truth path (blue). Looking at the results in the figure, we can see a drastic increase in position error, as the odometry paths begin to deviate greatly from the ground truth. We also see large effects from error-accumulation. At the start of the path the odometry estimates are very close to the ground truth, while at the end of the path they are quite far. This is in contrast to the noise-free case in Part 1 where the effect of error-accumulation was very minimal. The injection of noise is having a great impact on the accuracy of this odometry algorithm over time.

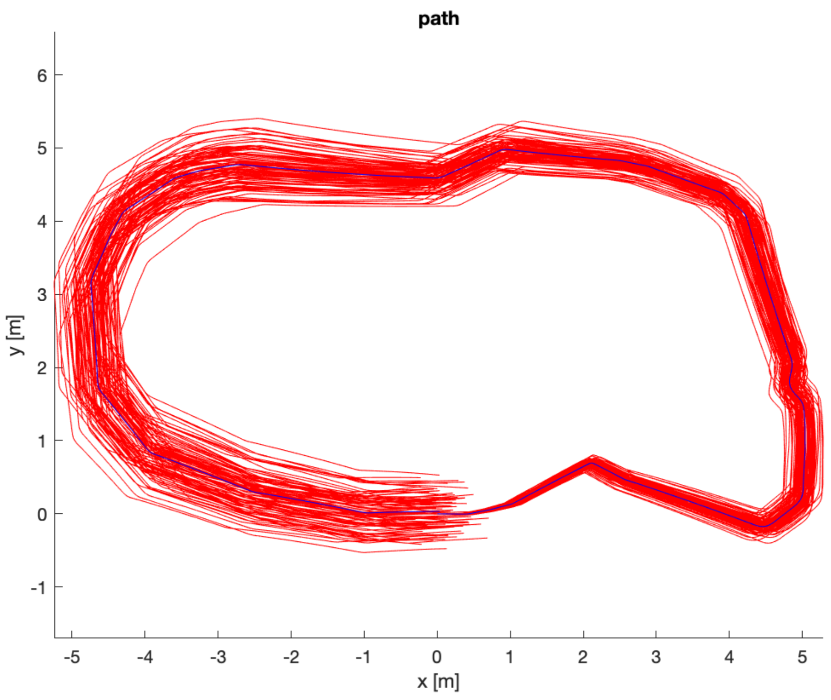


Figure 2: Output for Part 2

Part 3: Mapping from Noisy and Noise-free Odometry Data

In addition to the odometry data, the robot also collected LIDAR laser scan data. In Part 3, these laser scans were used, along with the noise-free and noisy odometry data, to produce maps of the robot surroundings. To do this, a few steps were required:

1. Perform an interpolation to account for the fact that the laser time-stamps and odometry time-stamps do not line up perfectly.
2. Determine the plotting of the laser scan points in the robot's current frame
3. Apply the homogenous transformation based on the translation and rotation from the odometry data to convert these points back to the robot's initial frame
4. Apply 2 patches to improve the crispness of the recorded map.
    a. Patch 1: Ignore laser scans collected when the robot's angular velocity was greater than 0.1 rad/s to remove the effects of time-step interpolation error when the robot is turning too quickly.
    b. Patch 2: Adjust the laser scans by 10cm to account for the fact that the true origin of these scans is 10cm behind the origin of the robot. This is done by adjusting the x and y coordinates by the appropriate component of the 10cm based on the robot angle.

Figure 3 shows the map (in red) as produced by the noisy odometry and laser scans. When compared to the ground truth (noise-free) odometry map (in blue), we see that the noise has indeed affected the accuracy of the mapping. The noisy map is nearly as crisp as the noise-free, but there is a significant shift due to the noise which may cause issues if attempting to use such a map for precise operations. We can also again that the mapping produced near the end of the robot path (the top right corner), is much more shifted than the mapping closer to the beginning of the path (bottom right). This could once again be evidence of the accumulated path error making an impact in the mapping of the laser scans.

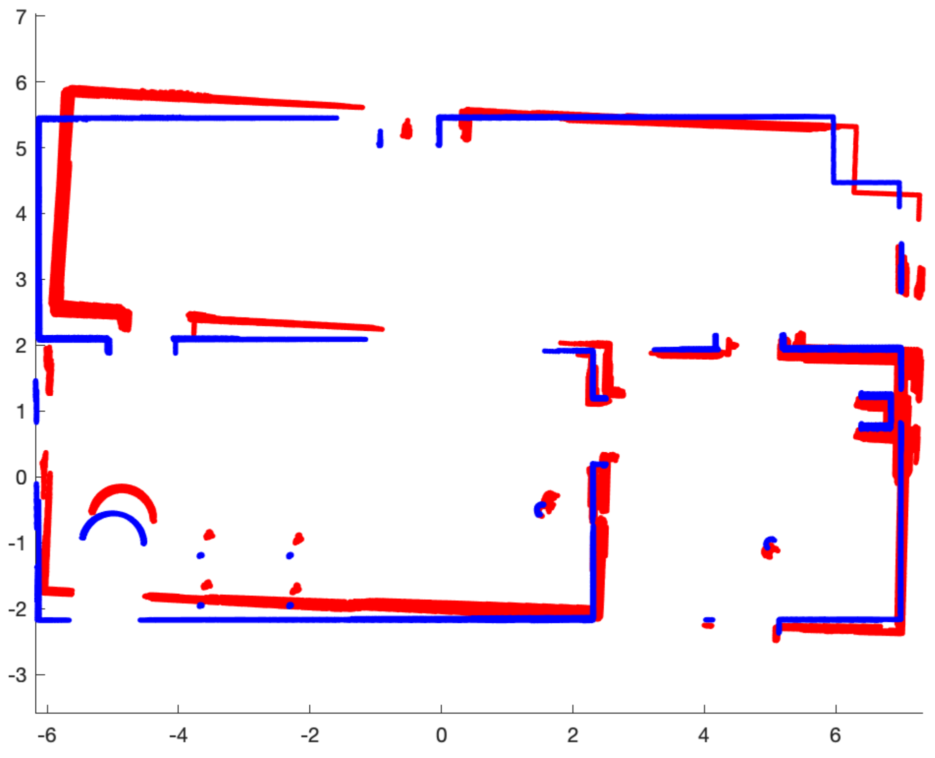
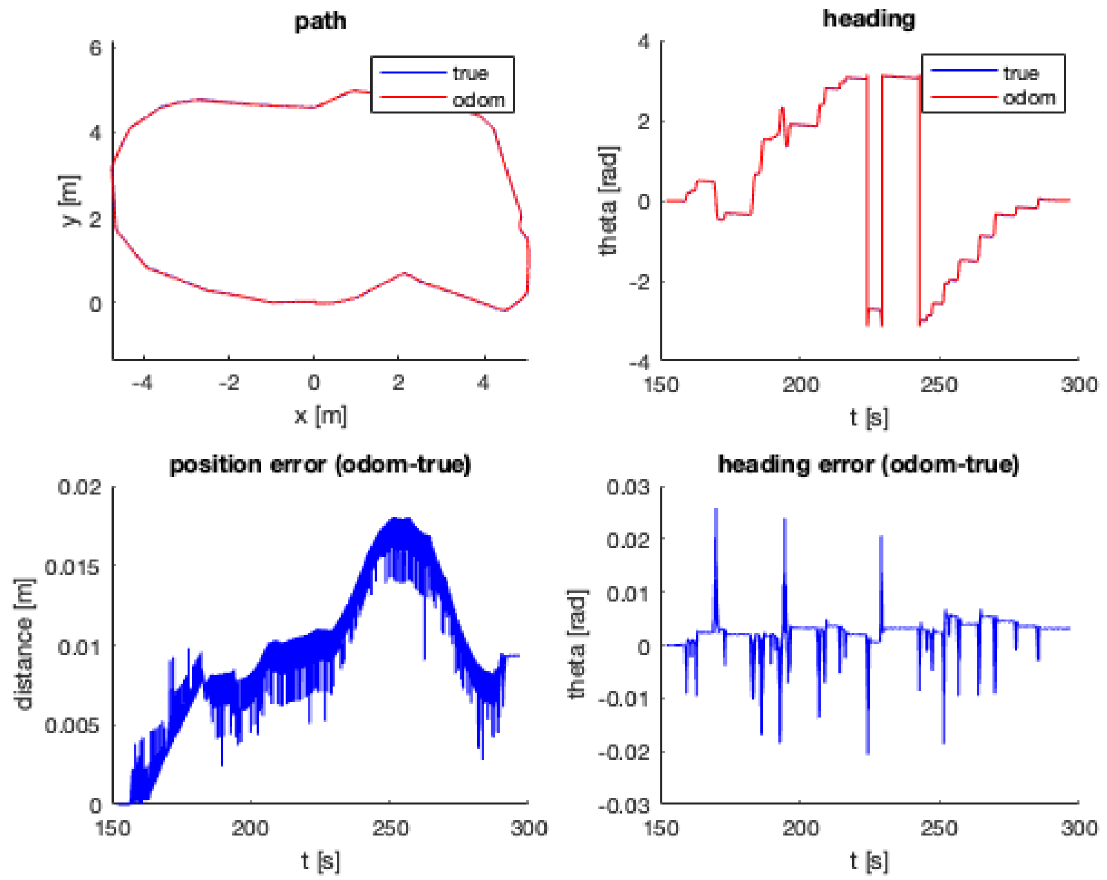


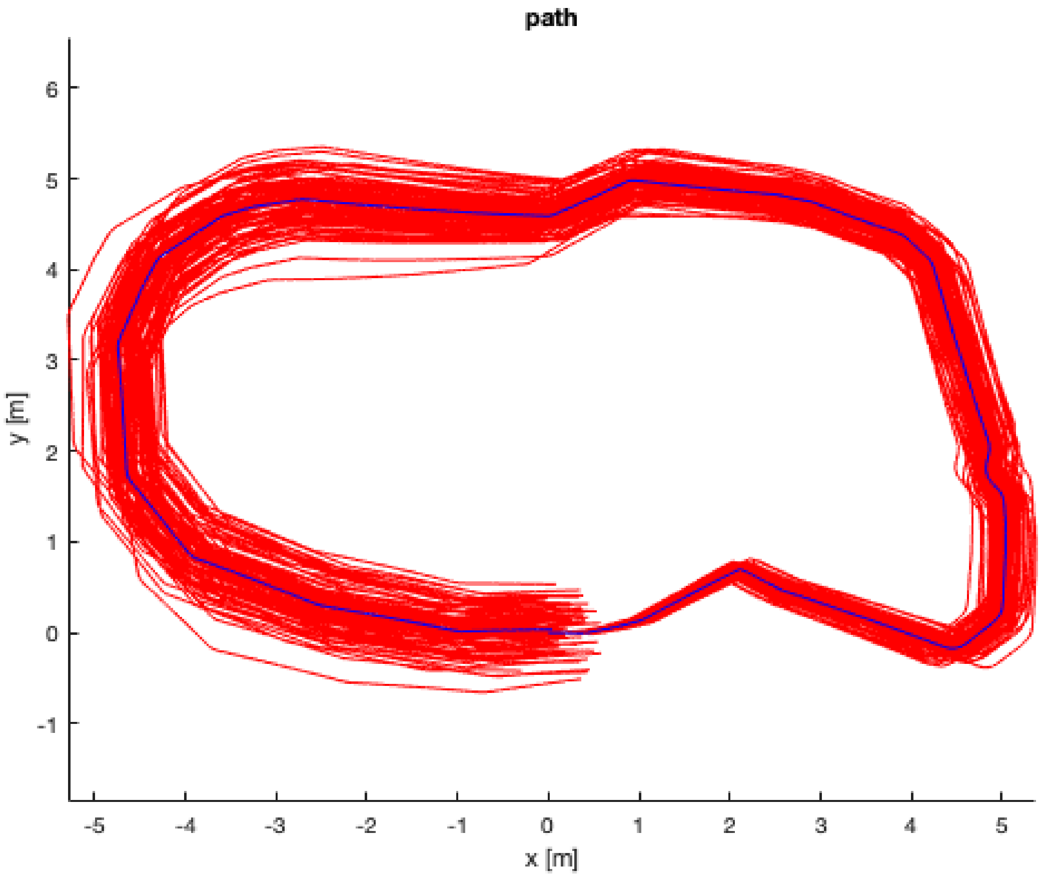
Figure 3: Map (Ground Truth and Odometry) of Robot Simulation Environment

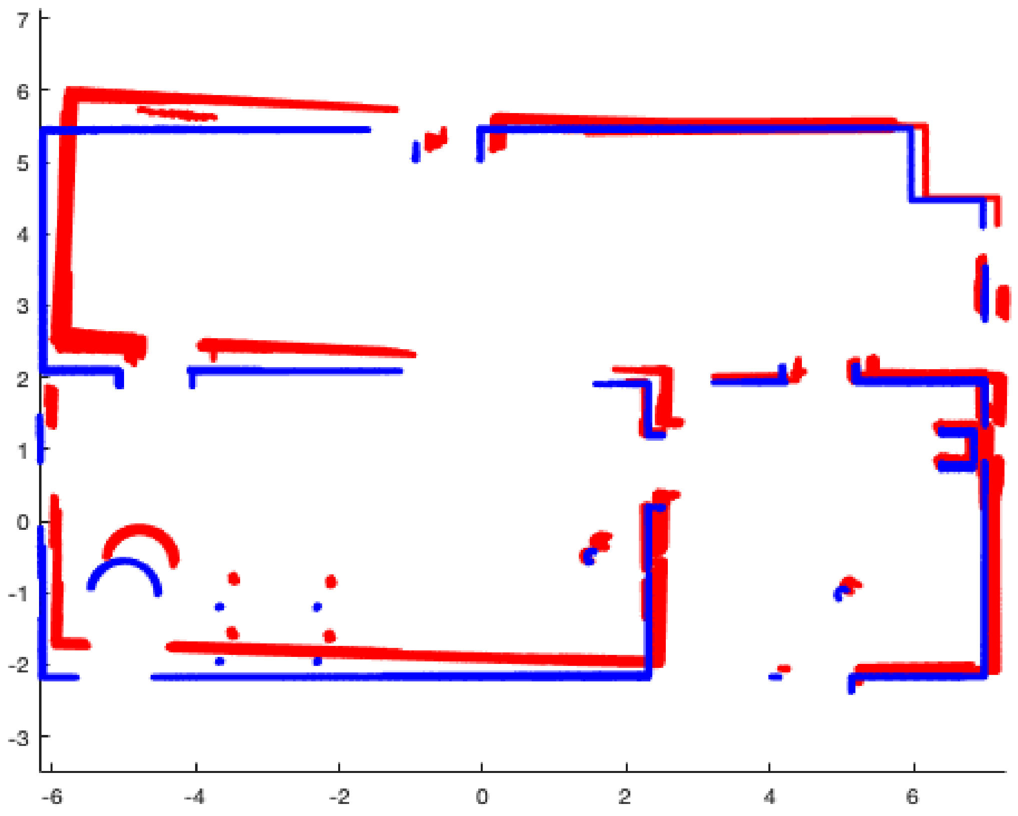
```
% ======
% ROB521_assignment2.m
% ======
%
% This assignment will introduce you to the idea of estimating the motion
% of a mobile robot using wheel odometry, and then also using that wheel
% odometry to make a simple map.  It uses a dataset previously gathered in
% a mobile robot simulation environment 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);
r = 66; %mm wheel radius
b= 287; %mm wheel separation,     data from lab3 datasheet
q = [x_odom' ; y_odom' ; theta_odom' ];

% ------insert your wheel odometry algorithm here-------
for i=2:numodom

    delta_t = t_odom(i) - t_odom(i-1);

    %compute the new x value
    x_odom(i) = x_odom(i-1) + delta_t*cos(theta_odom(i-1))*v_odom(i-1);
    %compute the new y value
    y_odom(i) = y_odom(i-1) + delta_t*sin(theta_odom(i-1))*v_odom(i-1);
    %compute the new theta value, wrap it to -pi pi interval to ensure
    %heading is correct
    theta_odom(i) = wrapToPi(theta_odom(i-1) + delta_t*omega_odom(i-1));


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.

load ROB521_assignment2_gazebo_data.mat;
% 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

        delta_t = t_odom(i) - t_odom(i-1);
        %compute the new x value
        x_odom(i) = x_odom(i-1) + delta_t*cos(theta_odom(i-1))*v_odom(i-1);
        %compute the new y value
        y_odom(i) = y_odom(i-1) + delta_t*sin(theta_odom(i-1))*v_odom(i-1);
        %compute the new theta value, wrap it to -pi pi interval to ensure
        %heading is correct
        theta_odom(i) = wrapToPi(theta_odom(i-1) + delta_t*omega_odom(i-1));
    end
    % ------end of your wheel odometry algorithm-------

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

    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------

        %apply 2 patches

        %The first patch is to only plot the laser scans if the angular
        %velocity is less than 0.1 rad/s;

        if abs(omega_interp(i)) < 0.1
            %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.

            %Perform transformation to base and adjust by 10cm (0.1m)
            x = ((cos(angles+theta_interp(i))).*y_laser(i,:))' +  x_interp(i)
 - 0.1*cos(theta_interp(i));
            y = ((sin(angles+theta_interp(i))).*y_laser(i,:))' +  y_interp(i)-
 0.1*sin(theta_interp(i));

            if n==2 %noise free map
                plot(x,y,'b.');
            else %noisy map
                plot(x,y,'r.');
            end

        end


        % ------end of your point transformation algorithm-------
    end
end

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




path
y [m]
x [m]
6
5
4
3
2
1
0
-1
-5
-4
-3
-2
-1
0
1
2
3
4
5



Published with MATLAB® R2021b