# ROB521 A2 - Wheel Odometry

*Ethan Rajah*

March 18, 2025

# 1 Question 1

This question required the assumption of noise free wheel odometry measurements to estimate the robot's pose over time. This simplifies the algorithm to a simple discrete integration of the wheel odometry, $u$ over time:

$$
\begin{aligned}
\dot{x}_{t+1} &= u_t \cos(\theta_t) \\
\frac{x_{t+1} - x_t}{dt} &= u_t \cos(\theta_t) \\
x_{t+1} &= x_t + u_t \cos(\theta_t)\, dt
\end{aligned}
$$

The same approach is taken for the $y$ and $\theta$ components of the robot pose, which aligns with the differential drive kinematics model represented with respect to the inertial frame from lecture. In the case where there is no noise in the sensor readings, the robot pose estimates are very accurate and closely follow the ground truth robot pose. This is seen in **Figure 1**, where the robot pose estimate is shown in red and the ground truth robot pose is shown in blue. The error between the odometry pose estimates and ground truth never exceed 0.02 meters in the $x$ and $y$ directions, and 0.025 radians in the $\theta$ direction, therefore demonstrating the accuracy of the algorithm when there is no noise.

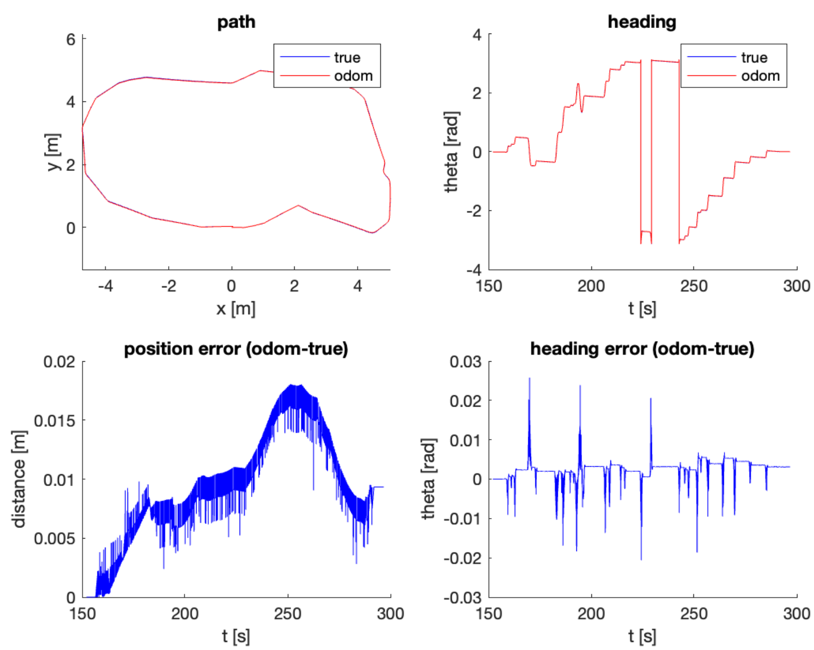


Figure 1: Noise-free wheel odometry robot pose estimates and error

# 2 Question 2

This question introduced gaussian noise to the wheel odometry measurements, which was then used to estimate the robot's pose over time. The purpose of this question was to demonstrate the degradation of the pose estimate accuracy when using the algorithm from Question 1 with noisy sensor readings. **Figure 2** shows the estimated robot pose path over time with noisy wheel odometry measurements for 100 random trials in red, and the ground truth robot pose in blue. Comparing the performance of the algorithm to Question 1, it is clear that the pose estimate error is significantly higher when noise is introduced. While the general path structure is still captured, detailed localization is missed, thus providing motivation for the mean and uncertainty pose estimate equations derived in lecture.

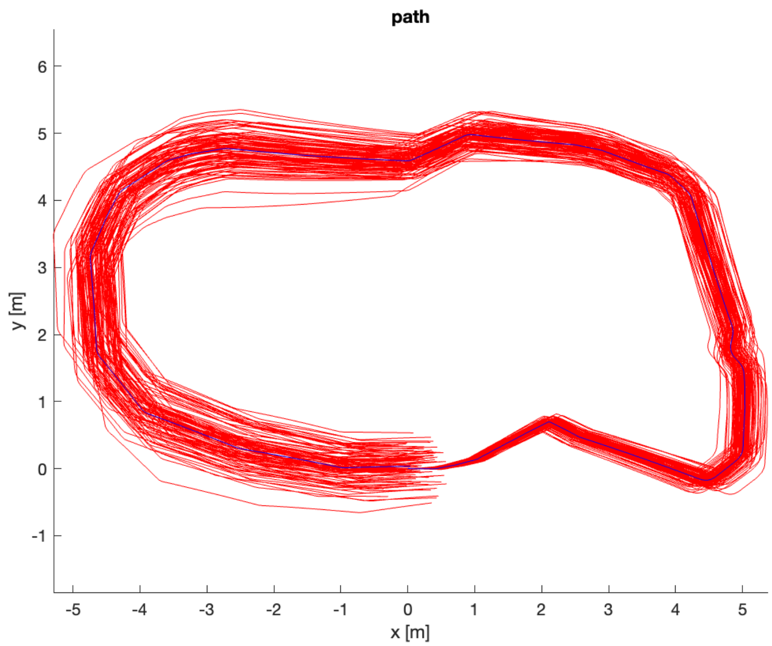


Figure 2: Comparison between noise-free and noisy wheel odometry robot pose estimates. The noisy wheel odometry robot pose estimates are shown in red, while the noise-free wheel odometry robot pose estimates are shown in blue.

# 3 Question 3

Question 3 extended the concepts of Question 2 by emphasizing the importance of considering sensor noise when using the data to build maps. The noise-free and noisy wheel odometry measurements are used with the LIDAR scan data to generate maps of the environment with respect to the initial robot pose. To get the scan data in the initial frame, the data is first transformed to the current robot pose using the fact that origin of the laser scans is about 10 cm behind the origin of the robot. Using the transformed $x$ and $y$ components of the laser data, the current robot pose from the wheel odometry is used to generate a transformation matrix to get the laser scan data with respect to the initial frame. More specifically, the current robot $\theta$ estimate is applied to the laser scan data through an elementary rotation matrix about the $z$ axis. The estimated $x$ and $y$ components of the robot pose are then used to translate the data to a position in the initial frame. The same process is done for the noise-free and noisy wheel odometry measurements, and the results are shown in **Figure 3**. From the figure it is clear that the introduction of noise into the system significantly degrades the quality of the map generated. While the general structure of the map is still captured, many details are positioned relatively far off from the ground truth map. Moreover, this demonstrates the importance of accounting for sensor noise when estimating the robot pose from wheel odometry measurements, as stochastic model representations can greatly improve estimates and map generation capabilities.

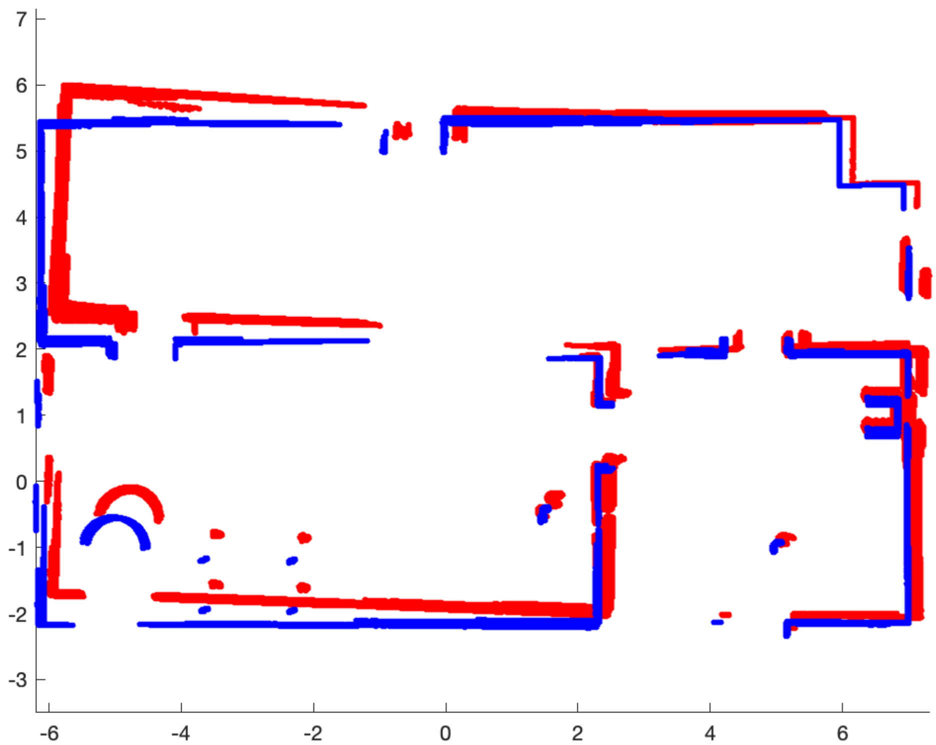


Figure 3: Map generation comparison between noise-free and noisy wheel odometry with respect to the initial robot pose. The noisy results are shown in red, while the noise-free results are shown in blue.

# 4 MATLAB Code

The MATLAB code for these implementations are provided here, as well as are attached in the submission.

```
% ======
% 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
    % For each odom measurement, calculate the change in x, y, and theta. Since this is noise-free, the equations are simply based on the numerical integration
    % of the velocities to get the position and heading.
    x_odom(i) = x_odom(i-1) + v_odom(i-1)*cos(theta_odom(i-1))*(t_odom(i)-t_odom(i-1));
    y_odom(i) = y_odom(i-1) + v_odom(i-1)*sin(theta_odom(i-1))*(t_odom(i)-t_odom(i-1));
    theta_odom(i) = theta_odom(i-1) + omega_odom(i-1)*(t_odom(i)-t_odom(i-1));

    % Ensure that the heading is between -pi and pi
    while theta_odom(i) > pi
        theta_odom(i) = theta_odom(i) - 2*pi;
    end
    while theta_odom(i) < -pi
        theta_odom(i) = theta_odom(i) + 2*pi;
    end
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
        % For each odom measurement, calculate the change in x, y, and theta. Since this is noise-free, the equations are simply based on the numerical integration
        % of the velocities to get the position and heading.
        x_odom(i) = x_odom(i-1) + v_odom(i-1)*cos(theta_odom(i-1))*(t_odom(i)-t_odom(i-1));
        y_odom(i) = y_odom(i-1) + v_odom(i-1)*sin(theta_odom(i-1))*(t_odom(i)-t_odom(i-1));
        theta_odom(i) = theta_odom(i-1) + omega_odom(i-1)*(t_odom(i)-t_odom(i-1));

        % Ensure that the heading is between -pi and pi
        while theta_odom(i) > pi
            theta_odom(i) = theta_odom(i) - 2*pi;
        end
        while theta_odom(i) < -pi
            theta_odom(i) = theta_odom(i) + 2*pi;
        end
    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

    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 ------

        % Laser scans and noisy odometry are now aligned. We can transform the laser scans into current robot frame based on the fact that the laser is
        % 10 cm behind the robot. We can then transform the points into the initial robot frame.
        % Only plot for low rotational velocities to avoid interpolation errors.
        if abs(omega_interp(i)) < 0.1
            % Transform laser scans into current robot frame
            laser_curr_robo_x = (y_laser(i,:)+0.1).*cos_angles;
            laser_curr_robo_y = (y_laser(i,:)+0.1).*sin_angles;

            % Transform current frame laser scans into initial robot frame (using principle rotation about z-axis)
            laser_initial_robo_x = laser_curr_robo_x.*cos(
```

```
                theta_interp(i)) - laser_curr_robo_y.*sin(
                theta_interp(i)) + x_interp(i);
            laser_initial_robo_y = laser_curr_robo_x.*sin(
                theta_interp(i)) + laser_curr_robo_y.*cos(
                theta_interp(i)) + y_interp(i);

            % Plot the points
            if n==1
                scatter(laser_initial_robo_x, laser_initial_robo_y,
                    10, 'r', "filled");
            else
                scatter(laser_initial_robo_x, laser_initial_robo_y,
                    10, 'b', "filled");
            end
        end

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

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