Exercise 3 and 4 – Matching observations to maps, Perception – Sensor fusion (Kalman filter)

Erik Nilsson, Julia Anderberg

March 2023

Abstract

This report is a summary of a series of experiments within the area of autonomous mobile robots and its capabilities of navigation. Key components for successful navigation is among others perception and localization. Perception is the robots ability to gather information about its surroundings from sensor measurements. In localization the robot derives the sensor measurement to estimate its position in the environment. In this report a method combining Dead Reckoning, Cox algorithm and Kalman filtering is presented. It was shown that the kalman filter can be used to combine two types of sensors and derive a better position estimate than any of the sensors could on their own.

1 Introduction

In autonomous mobile robots one of the key challenges is robust navigation. To enable successful navigation there are four key components that are required. First one is Perception meaning that the robot senses its surroundings. Second component is localization, the robot determines its own position within the environment. Cognition is the third component, which means that the robot interprets the data it received from its sensors to decide what actions to perform. The last and forth one is motion control, where the robot perform the actions decided necessary to achieve the desired trajectory. This report is a summary of the result from Exercise 3 & 4 within the course Intelligent Vehicles, Spring 2023. In the report the main focus is on perception and localization.

1.1 Perception & Localization

Robotic perception is one of the most important component of an autonomous system. It is crucial for the robot to gather knowledge and data about its environment to be able to make decisions, plan paths and operate in real-world environments. The key tasks in perception is to perform measurement through sensors to enable the robot to perceive, comprehend and reason about the surrounding environment. On-board sensors used for robots localization can be classified in two functional classes proprioceptive and exteroceptive sensors. Proprioceptive sensors, such as inertial measurement units (IMUs) and encoders measures values internal of the robot, for example speed and angle of the wheel. Localization methods using proprioceptive sensors is called Odometry or Dead reckoning. [1] A known issue with Odometry and Dead Reckoning when used on its own, integrated error and uncertainty in global position increases rapidly over time. Therefor there is a need for algorithms to optimize and minimize the errors in the estimation. In [2] the authors propose a method for combining a LiDAR(Light Detection And Ranging), Odometer and IMU together with Kalman Filter algorithm to obtain a real-time positioning system.

Exteroceptive sensors acquire measurement from the robots environment, such as LiDAR and cameras. Mobile robots are heavily dependant on exteroceptive sensors, a lot of work has been put in to the task of reducing errors. Systematic errors can be modeled and removed by the application of mathematical models, while random errors can destroy a measurement if they are too large compared to the signal [1].
In this report we will present two methods that improves localization, Kalman filter and Cox Algorithm[3]. The kalman filter is a method of combining different sensors to get a better reading. In this report a LiDAR sensor and dead reckoning will be combined. The readings from the LiDAR will be extracted using the cox line-fit algorithm[3] and the dead reckoning model will be based off of Wang[4]. The equation and steps needed to implement the Cox algorithm is described in Section 3 as well as the equations for the Kalman filter. In section 4 the experiments performed and results are presented.

2 Method

2.1 Unit conversion

In exercise 1 a gps receiver is used to estimate the position, speed and heading of a vehicle. The gps is not used again in the later exercises but the methods to extract speed and heading from position data are important for exercise 2. Equations 1,2 and 3 are used to calculate the distance, speed and heading between two points in a gps system. F_{lat} and F_{lon} are conversion factors for latitude and longitude, ϕ and λ are the respective coordinates and τ is the timestamp for the measurement.[5]

$$D = \sqrt{(F_{lat}(\phi_1 - \phi_2))^2 (F_{lon}(\lambda_1 - \lambda_2))^2} \tag{1}$$

$$v = D/(\tau_1 - \tau_2) \tag{2}$$

$$\theta = \begin{cases} 0^\circ = 360^\circ & \lambda_1 = \lambda_2 \ \& \ \phi_1 < \phi_2 \\ 180^\circ & \lambda_1 = \lambda_2 \ \& \ \phi_1 > \phi_2 \\ 90^\circ - tan^{-1}(\frac{F_{lat}(\phi_2 - \phi_1}{F_{lon}(\lambda_2 - \lambda_1)}) & \lambda_1 < \lambda_2 \\ 270^\circ - tan^{-1}(\frac{F_{lat}(\phi_2 - \phi_1}{F_{lon}(\lambda_2 - \lambda_1)}) & \lambda_1 > \lambda_2 \end{cases} \tag{3}$$

2.2 Dead reckoning for three wheeled robot

In exercise 2 methods for estimating a three wheeled robots position using dead reckoning was explored. The problem turned out to be two-fold. Firstly, a prediction had to be made of the robots pose. This was done by using given data of wheel speed, steering direction and sampling interval (equation 4). Secondly the uncertainty of the pose prediction also had to be calculated. The uncertainty calculation was done by using the law of error propagation (equation 5). The result of the calculation is a covariance matrix describing the uncertainty of the robots pose. To use the law of error propagation you need to calculate the jacobian of the kinematic model with regards to different sources of uncertainty, for example: uncertainty in position and uncertainty in measurements. Some of these jacobian matrices can become very large and complex, for this reason MATLAB's jacobian() function was of great help.

$$X_k = \begin{bmatrix} x_{k-1} + v * cos(\theta) * T * cos(\theta_{k-1} + \frac{v*sin(\theta)T}{2L}) \\ y_{k-1} + v * cos(\theta) * T * sin(\theta_{k-1} + \frac{v*sin(\theta)T}{2L}) \\ m\theta_{k-1} + \frac{v*sin(\theta)T}{L} \end{bmatrix} \tag{4}$$

$$\Sigma_{X_k} = J_{X_{k-1}} \Sigma_{X_{k-1}} J_{X_{k-1}}^T + J_{v\alpha T} \Sigma_{v\alpha T} J_{v\alpha t}^T \tag{5}$$

The kephera robot used in exercise 2 uses a differential drive configuration with wheel encoders, the encoder output needs to be converted to distance traveled and angle difference, a model for doing this and also calculating the covariance of the wheel encoder measurements is described in Location estimation and uncertainty analysis for mobile robots by Wang [4]. In exercise 2 three experiments were performed with three different kinematic models (equations 4, 6 and 7).
In exercise three and four only the snowhite robot was used so only its kinematic model was used (equation 4).

$$X_k = \begin{bmatrix} x_{k-1} + \Delta d * cos(\theta_{k-1} + \Delta\theta/2) \\ y_{k-1} + \Delta d * cos(\theta_{k-1} + \Delta\theta/2) \\ \theta_{k-1} + \Delta\theta \end{bmatrix} \tag{6}$$

$$X_k = \begin{bmatrix} x_{k-1} + \frac{\Delta r+\Delta l}{2} cos(\theta_{k-1} + \frac{\Delta r+\Delta l}{2b}) \\ y_{k-1} + \frac{\Delta r+\Delta l}{2} sin(\theta_{k-1} + \frac{\Delta r+\Delta l}{2b}) \\ \theta_{k-1} + \frac{\Delta r+\Delta l}{b} \end{bmatrix} \tag{7}$$

2.3 Cox line fit algorithm

The cox line fit algorithm was developed for the purpose of matching measured points to a set of infinite lines. In this report it is used to match points provided by a laser range sensor to a pre-determined map. More information about the method can be found in *Blanche-an experiment in guidance and navigation of an autonomous robot vehicle* by Cox[3].

2.3.1 Unit vectors

The lines in the map need to be represented by their orthogonal unit vectors calculated in equation 8, the distance to the line from origo is found by calculating the dot-product between the unit vector and any point on the line (9)

$$\begin{aligned} p_1 &= (x_1, y_1) \\ p_2 &= (x_2, y_2) \\ L &= p_2 - p_1 \\ u &= \frac{\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \cdot L^T}{||L||} \end{aligned} \tag{8}$$

$$r = u \cdot p_1 \tag{9}$$

2.3.2 Step 1: Translate and rotate datapoints

The datapoints from the laser range sensor comes as one array of distance measurements and one array with their corresponding angles, for every point a series of transforms are applied to move it to the world coordinate system. First the datapoints coordinates are converted from polar to cartesian coordinates(10).

$$\begin{aligned} x &= DIS_i \cdot cos(ANG_i) \\ y &= DIS_i \cdot sin(ANG_i) \end{aligned} \tag{10}$$

The next step is to transform the position of the point into the robots reference frame.

$$Rs = \begin{bmatrix} cos(s\theta) & -sin(s\theta) & sx \\ sin(s\theta) & cos(s\theta) & sy \\ 0 & 0 & 1 \end{bmatrix} \qquad X\_robot = R \cdot \begin{bmatrix} x & y & 1 \end{bmatrix}^T \tag{11}$$

$$Rw = \begin{bmatrix} cos(Ra + dda) & -sin(Ra + dda) & Rx + ddx \\ sin(Ra + dda) & cos(Ra + dda) & Ry + ddy \\ 0 & 0 & 1 \end{bmatrix} \qquad Ws = Rw \cdot \begin{bmatrix} X\_robot_1 & X\_robot_2 & 1 \end{bmatrix}^T \tag{12}$$

2.3.3 Step 2: Find targets for datapoints

Every line in the map needs to be assigned datapoints in order to know what error we are trying to minimize with the congruence, to do this the distance y_j is calculated by taking the dot-product of the lines unit vector u_j and the points location vector v_i.

$$y_i = u_j \cdot v_i \tag{13}$$

For every point, the line with the shortest distance y_i is chosen and set as target. Outliers also had to be removed in this step, since outliers will impact the result of the least square fit. Outliers were classified as having a distance to its closest line larger than the median distance of all the points to their closest line. The outliers were then removed from the dataset.

2.3.4 Step 3: Set up linear equation system

A linear equation system need to be set up to fit the datapoints to the map using the least square method(equation 14). "vm" in this case refers to the position of the robot. $x1$ is a column vector containing all x-components of the unit vectors for each points target, $x2$ is the same but for y-components. A becomes a $3xn$ matrix with a row for every datapoint and b becomes a 3x1 matrix containing how much the robots pose must change in x, y and θ to achieve a least square fit.

$$\begin{aligned} x_i1 &= u_i(1) \\ x_i2 &= u_i(2) \\ x_i3 &= u_i \cdot \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \cdot (v_i - vm) \\ A &= \begin{bmatrix} x1 & x2 & x3 \end{bmatrix} \\ b &= (A^T A)^{-1} A^T \cdot y \end{aligned} \tag{14}$$

2.3.5 Step 4: Add latest contribution to overall congruence

The contents of b are added to the robots pose for the next iteration, where points may be assigned to new lines and achieve a better fit.

2.3.6 Step 5: Check if the process has converged

To know if the iteration should be stopped we check weather b has converged:

$$\begin{aligned} \sqrt{b(1)^2 + b(2)^2} &< 5 \\ and & \\ |b(3)| &< 0.1 * \frac{\pi}{180} \end{aligned} \tag{15}$$

If this is the case the function terminates and returns the total congruence and the covariance matrix generated by the line fit. The covariance matrix is given by equation 16.

$$s2 \cdot (A^T A)^{-1} \tag{16}$$

$$s2 = \frac{(y - Ab)^T \cdot (y - Ab)}{n - 4} \tag{17}$$

2.4 Kalman filter

The Kalman filter is a method of taking a weighted average of two measurements, combining these to get a more accurate estimation of position. The weight of each measurement is proportional to the uncertainty of that measurement so that even measurements with great uncertainty may make a small contribution to the overall position estimate.[6]
We used the Kalman filter in two steps: first to receive a new position estimate (equation 18), second to receive a new covariance matrix representing the uncertainty in that position estimate(equation 19). It takes as input: the uncertainty of your position measurement ($\Sigma_{\hat{X}}$), the uncertainty of your position fix (Σ_{pf}), your position estimate according to dead reckoning ($\hat{X}$) and the position fix provided by the cox line fit algorithm ($\hat{X}_{pf}$).

$$\hat{X}^{+}(i) = \Sigma_{pf}(i)\left(\Sigma_{pf}(i) + \Sigma_{\hat{X}}(i)\right)^{-1}\hat{X}^{-}(i) + \Sigma_{\hat{X}}(i)\left(\Sigma_{pf}(i) + \Sigma_{\hat{X}}(i)\right)^{-1}\hat{X}_{pf}(i) \tag{18}$$

$$\Sigma^{+}_{\hat{X}}(i) = \left(\Sigma^{-1}_{pf}(i) + \Sigma^{-1}_{\hat{X}}(i)\right)^{-1} \tag{19}$$

3 Experiments and results

Five experiments were conducted in exercises three and four, some more advanced than others but none the less all useful for gaining an understanding of the program and the methods used therein

3.1 Run and understand code

The first experiment was just to run the program as provided to get some understanding of its structure and how it works. The program provided uses only dead reckoning for position estimation, but without calculating the uncertainty in its position. It doesn't take into account the laser measurements. Path quickly deviates from ground truth and has no hope of correcting itself.

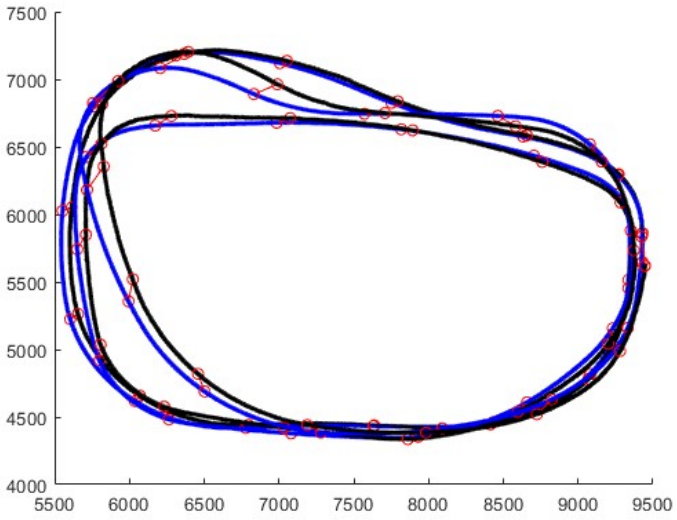


Figure 1: Predicted position of robot in blue and ground truth in black

3.2 Cox line-fit test

The second test uses only the position fix given by the cox lie-fit algorithm to update the position of the robot. This had varying results, sometimes the fix gives a position to the left of ground truth and sometimes to the right, as can be seen in figure (fig 2). This is since the cox line-fit algorithm is not perfect and neither is the data provided to it. It is very sensitive to outliers and even though we remove outliers we use a quite crude method and some might slip through.
Overall error looks slightly better than without, but as we can see the path is no longer smooth. Uncertainty of position fix is set as the total uncertainty after every fix and as we can see it increases between fixes due to the dead reckoning model now being implemented.(fig 3
We can also see the effect of the cox line-fit algorithm in how the laser measurements align with the walls in the map after the run. We can see the measurements fit into the corner after some rotation and despite outliers it has managed a good fit. The outliers in this picture could be due to some temporary obstacle on the floor that was not included in the map.

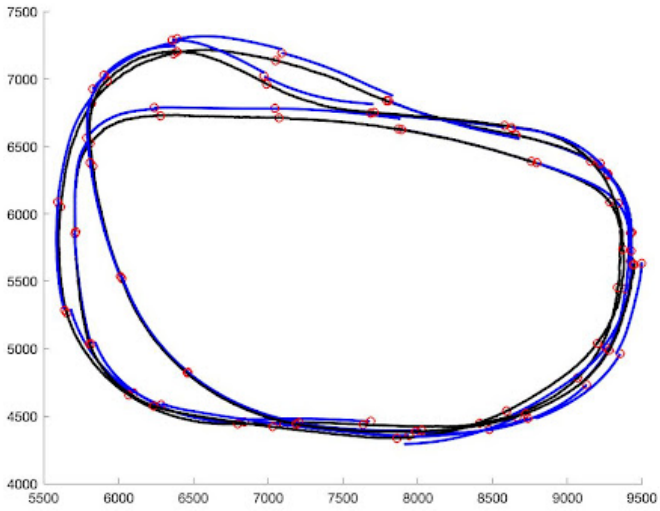
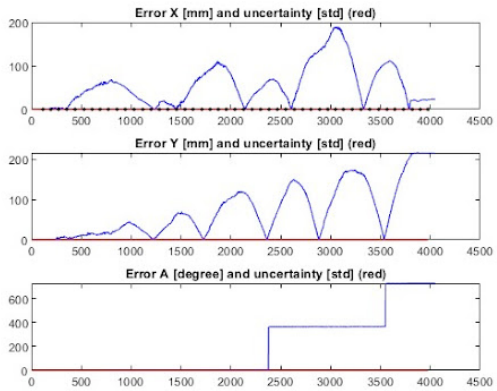
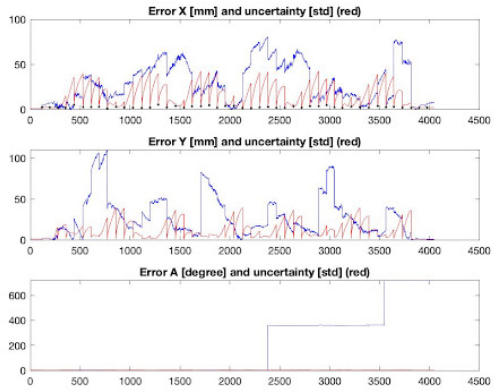


Figure 2: Here the cox line-fit algorithm position fixes are set as new position

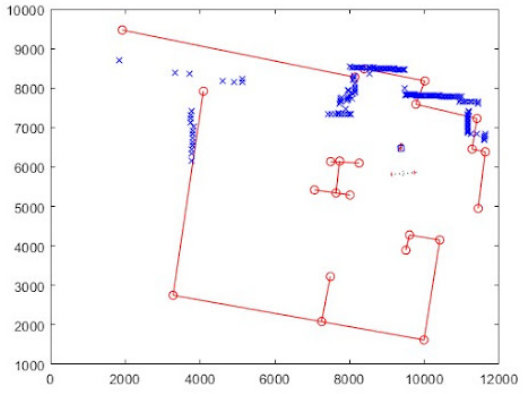


(a) Error and uncertainty in Experiment 1

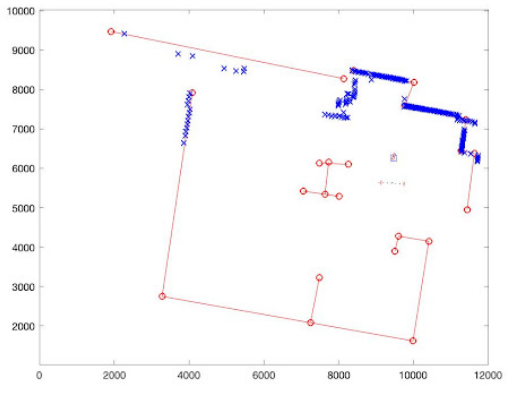


(b) Error and uncertainty in Experiment 2

Figure 3: Comparison of the errors and uncertainties in experiments 1 and 2



(a) Laser point cloud in experiment 1



(b) Laser point cloud in experiment 2

Figure 4: Comparison of the scan match between experiment 1 and 2

3.3 Simulation in two parts

This experiment contains two parts where the same experiment is run with two different sets of uncertainties. This test focuses only on the workings of the kalman filter so we need to simulate the position fix normally given by the cox line-fit algorithm. To do this we add a random($\mu = 0, \sigma = \theta$) variable to the ground truth position

and use that as our position fix and we create a covariance matrix for the position fix (equation 21). since MATLAB's randn() function returns normally distributed numbers with a mean of zero and standard deviation of 1 we simply had to multiply the random variable with the desired standard deviation to get the distributions we wanted to test with as in equation 20.[7]

$$\begin{aligned} \hat{x}_{pf} &= x_{true}(i) + randn() * \sigma_x \\ \hat{y}_{pf} &= y_{true}(i) + randn() * \sigma_y \\ \hat{\theta}_{pf} &= \theta_{true}(i) + randn() * \sigma_\theta \end{aligned} \tag{20}$$

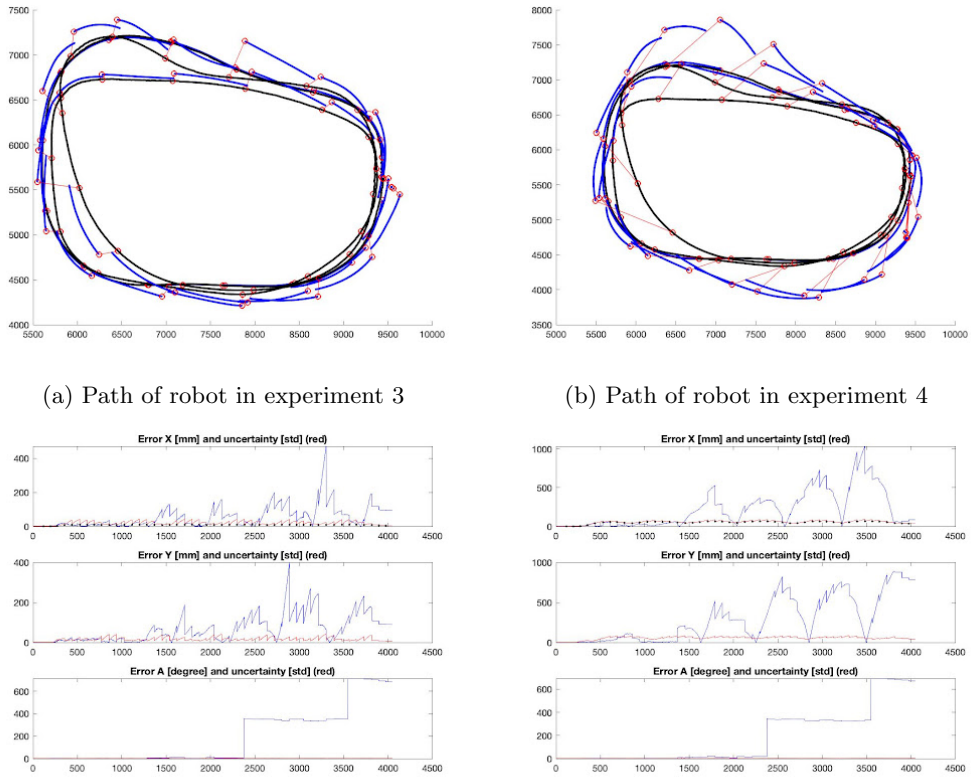
$$\Sigma_{pf} = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta \end{bmatrix} \tag{21}$$

3.3.1 $\sigma_x = 10, \sigma_y = 10, \sigma_\theta = 1$

For the first test the standard deviation in X and Y direction was set to 10 and for the heading it was set to 1. In experiment 3 the variance of the random position fixes were comparatively low so the kalman filter feels it can trust the random position fixes more than the dead reckoning in some cases as can be seen in figure 5a.

3.3.2 $\sigma_x = 100, \sigma_y = 100, \sigma_\theta = 3$

For the second test the standard deviation in X and Y direction was set to 100 and for the heading it was set to 3. This is significantly higher than what we can expect from the cox line-fit algorithm and as we can see in figure 5b the kalman filter returns positions much closer to what is estimated by the dead reckoning of the robot.



(a) Path of robot in experiment 3

(b) Path of robot in experiment 4

(c) Error and uncertainty in experiment 3

(d) Error and uncertainty in experiment 4

Figure 5

3.3.3 Comparison of two part experiment

From figure 5 we can see that when the uncertainty is lower, the simulated scan match is trusted more than the dead reckoning, so the estimated position of the robot changes by a lot every scan. When the uncertainty is high however the dead reckoning is trusted more and the new position estimates stay closer to the old estimates given by the dead reckoning.

3.4 Kalman filter: scan matching and dead reckoning

In the last experiment the kalman filter was used to fuse the position fix given by the cox line-fit algorithm and the dead reckoning of the robot. So far this has given the best result closest to ground truth.

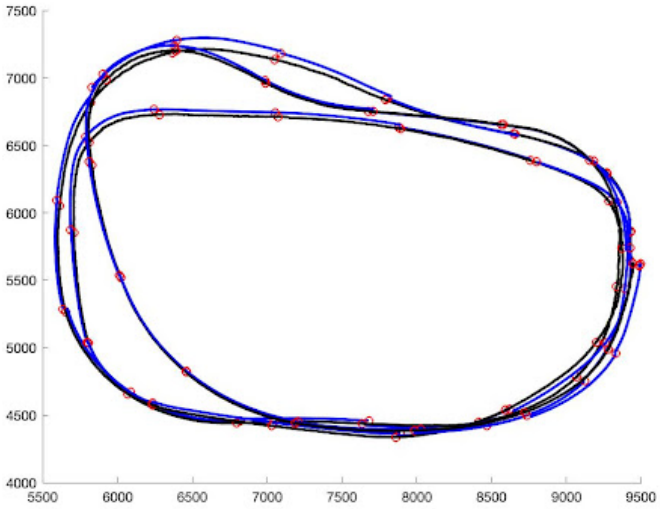


Figure 6: Using the kalman filter to fuse dead reckoning and LiDAR

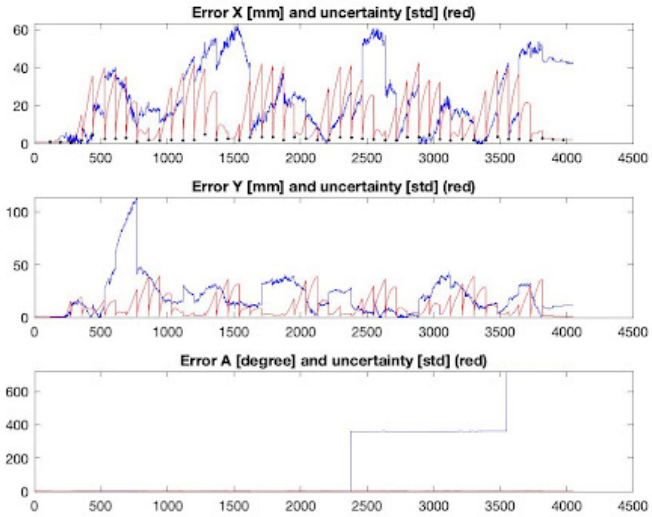


Figure 7: Error and uncertainty in experiment 5

4 Conclusions

In these exercises we have implemented an algorithm for matching LiDAR scans to a map and used the kalman filter to fuse sensor readings and get a better position estimate as a result. From the experiments we have conducted we can see that a LiDAR is a good complement for dead reckoning to reduce uncertainty and even improve the position estimate, given that the LiDAR's position estimate is implemented well and that it is fused optimally by way of kalman filter or other equivalent method. We have learned about the challenges in robot navigation and how to solve the problem of error propagation by using a LiDAR and pre-provided map of the robots environment. We have also learned how to combine different sensors capabilities and thus get a better estimate than we would if we only had one sensor. The cox line fit algorithm is not cited a lot, perhaps a more well established method could have better results.

References

[1] R Siegwart IRN. Autonomous Mobile Robots; 2004.

[2] Li Z, Su Z, Yang T. Design of Intelligent Mobile Robot Positioning Algorithm Based on IMU/Odometer/Lidar. In: 2019 International Conference on Sensing, Diagnostics, Prognostics, and Control (SDPC); 2019. p. 627-31.

[3] Cox IJ. Blanche-an experiment in guidance and navigation of an autonomous robot vehicle. IEEE Transactions on Robotics and Automation. 1991;7(2):193-204.

[4] Wang CM. Location estimation and uncertainty analysis for mobile robots. In: Proceedings. 1988 IEEE International Conference on Robotics and Automation; 1988. p. 1231-5.

[5] Adamchuk V. EC01-157 Precision Agriculture: Untangling the GPS Data String. 2001 01.

[6] Maybeck PS. Stochastic models, estimation, and control. vol. 141 of Mathematics in Science and Engineering; 1979.

[7] Random Numbers from Normal Distribution with Specific Mean and Variance;. Accessed: 2023-03-07. https://se.mathworks.com/help/matlab/math/random-numbers-with-specific-mean-and-variance.html.