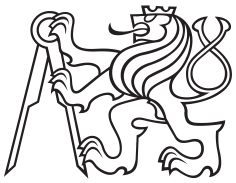
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering

LIDAR-Based Lane Tracking using Kalman Filtering and its Fusion with Camera-Based Lane Data

Daniel Veškrna

Supervisor: Nuri Kundak, MSc.
January 2024

Acknowledgements

I would like to thank my supervisor Nuri Kundak Msc. for his guidance and willingness throughout the making of this thesis. I would also like to thank Bc. Sana Othmanová and the Autonomous Driving and ADAS department of Porsche Engineering Services, s.r.o. for the opportunity and support.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university thesis.

In Prague on January 9, 2024 .....

Abstract

This thesis is devoted to the comparison of several lane estimation techniques with different underlying road geometry models with the use of publicly available LIDAR and camera data. For this purpose, a whole framework was developed for data processing and error evaluation. Knowledge of how the presented methods behave under specific circumstances and how they compare with each other is essential in the development of new advanced driver assistance systems.

The Kalman filter for lane parameters estimation and RANSAC algorithm were chosen for comparison, with first, second, and third degree polynomial road models. The last mentioned model, which is an approximation of a clothoid, showed the best results as the Kalman filter model. A proposed adaptive version of the Kalman filter capable of switching between models showed only minor improvements, but further testing is needed. The RANSAC alone performed poorly, but showed promising results in the presence of outliers. Especially as a pre-filter for the measurements when combined with the Kalman filter. The framework created can be used to compare other lane estimation techniques and contribute further to the development of driver assistance systems.

Keywords: lane estimation, Kalman filter, RANSAC, LIDAR

Supervisor: Nuri Kundak, MSc.

Abstrakt

Tato práce je věnována porovnání několika technik estimace jízdních pruhů s různými modely geometrie vozovky za využití veřejně dostupných dat z LIDARu a kamer. Za tímto účelem byl vyvinut celý framework pro zpracování dat a vyhodnocování chyb. Znalost toho, jak se prezentované metody chovají za konkrétních okolností a jaké jsou ve srovnání mezi sebou, je nezbytná při vývoji nových pokročilých asistenčních systémů pro řidiče.

Pro srovnání byl zvolen Kalmanův filtr pro estimaci parametrů jízních pruhů a algoritmus RANSAC s modely silnic polynomu prvního, druhého a třetího stupně. Poslední zmíněný model, který je aproximací klotoidy, vykazoval nejlepší výsledky jako model pro Kalmanův filtru. Navrhovaná adaptivní verze Kalmanova filtru schopná přepínat mezi modely přinesla pouze drobná vylepšení, ale je potřeba dalšího testování. RANSAC jako samotný si vedl špatně, ale ukázal slibné výsledky v přítomnosti vysokého šumu. A to zejména jako předfiltr pro měření v kombinaci s Kalmanovým filtrem. Vytvořený framework lze použít k porovnání jiných technik estimace jízdních čar a dále přispět k rozvoji asistenčních systémů řidiče.

Klíčová slova: estimace jízních pruhů, Kalmanův filtr, RANSAC, LIDAR

Překlad názvu: Sledování jízních pruhů na bázi LIDAR dat pomocí Kalmova filtrování a jeho fúze s kamera daty

Contents

Figures

Chapter 1

Introduction

As cars become smarter and smarter, more Advanced Driver Assistance Systems (ADAS) are being implemented. This is highly motivated by improved driving safety and efficiency, which many of the existing driver assistance features, such as forward collision warning or safe lane change, already provide [1]. It is a well-known fact that most car accidents are caused by human error. The advancement in autonomous vehicles with ADAS has a great potential to mitigate such mistakes [2]. Lane detection and tracking algorithms are a key part of several existing ADASs, such as lane departure warning. It detects and warns the driver when the car unintentionally crosses lane markings and can steer the car back into its road lane.

To detect lane markings, most cars today use several cameras, which are susceptible to extreme weather or poor lighting conditions. RADAR sensors cannot be used for this purpose because of low resolution, but are suitable for adaptive cruise control and collision mitigation [3, 4]. Ultrasonic sensors (USS) are mainly used for lane departure warning, parking assistance, and other simpler tasks. In recent years, Tesla has begun to remove RADAR and USS from its vehicles, relying solely on camera-based ADAS [5]. Elon Musk stated on X that "Humans drive with eyes and biological neural nets, so makes sense that cameras and silicon neural nets are only way to achieve generalized solution to self-driving." [6]. One could argue that we also use ears and touch to some lesser extent. It is a fact, however, that roads, signs, markings, etc., are designed for humans. But human senses are prone to error, and using technology not based on them, such as LIDAR, could mitigate our mistakes.

1.0.1 Aim of Thesis

This thesis aims to compare several means of lane tracking and estimation based on LIDAR data. Specifically to use Kalman filter (KF) of different types with several underlying physical models. Furthermore, to investigate the possibility of using Random Sample Consensus algorithm (RANSAC) to improve the line detection. The potential benefits of using measurements from both LIDAR and camera will be discussed as well. All data for testing comes from the publicly available PandaSet database by Hesai and Scale [7].

1.0.2 Layout of Thesis

There are 6 chapters in this work. Chapter 1 covers the above written introduction, motivation, and aim of the thesis, which is further discussed in parts of next chapter 2. It includes state-of-the-art overview of lane detection, tracking and estimation, general approach to tracking, a look into LIDAR and camera abilities, and lastly, theory of Kalman filter and RANSAC algorithms. Chapter 3, Methods and Implementation, starts with a description of the PandaSet and characteristics of the measured data, followed by lane tracking and estimation pipeline in this work. The core implementations of the latter mentioned algorithms are explained next. Results and discussion follow. The benefits of LIDAR-Camera integration are in chapter 4. In the last chapter 5, the final conclusions are drawn. In the very end of the thesis is the appendix with bibliography, implemented code, and project specification.

1.0.3 Personal Contributions

The author of this thesis wrote the algorithms to process the data from LIDAR and camera. He proposed some methods of how to use them for the tracking and estimation of road lines. He tested them and evaluated the obtained results presented in this work.

Chapter 2

State of Art

This chapter delves into current status of the lane detection and tracking field. Broad overview with key approaches to this problem is discussed first, followed by general steps of how it is solved. Since this work focuses on using LIDAR as the main source of measurement data and cameras being commonly used the most, advantages and disadvantages of LIDAR and camera in context of ADAS are briefly mentioned next. Last part of this chapter is devoted to theory of two main methods used in this work, Kalman filter and RANSAC, discussing their use in ADAS as well.

2.1 Lane Detection and Tracking Overview

In a recent study from 2021 [8], the authors provide a comprehensive review of current approaches to lane detection and tracking algorithms. As they point out, there are not many studies that provide an overview of the findings in this area. There are three basic methods for lane detection and tracking:

- Features-Based Approach (Image and Sensor-Based Lane Detection and Tracking)
- Model-Based Approach (Robust Lane Detection and Tracking)
- Learning-Based Approach (Predictive Controller Lane Detection and Tracking)

Each of these will be further discussed, and the following claims are based on their observations.

2.1.1 Features-Based Approach

This methodology relies on the integration of sensors and camera outputs for decision making in lane detection and tracking. Image frames are pre-processed, and specific lane detection algorithms are applied. The decision-making process for lane tracking is guided by sensor values. Several studies have implemented various methods, including inverse perspective mapping, kinematic-based fault-tolerant mechanisms, the Hough transform, and Kalman filters. For example, Kuo et al. implemented a vision-based lane keeping system using inverse perspective mapping, lane scope feature detection, and lane markings reconstruction [9]. The performance of these approaches varies under different conditions, such as sunlight and rain. Challenges include reduced effectiveness in tunnels and specific environmental scenarios.

2.1.2 Model-Based Approach

The Model-Based Approach employs global road models to fit low-level features, emphasizing robustness against illumination. Geometric parameters are utilized for effective lane detection. These must be chosen appropriately, as they are very sensitive to changes in road shapes. Noteworthy studies have utilized hierarchical agglomerative clustering, adaptive thresholds, and RANSAC algorithms. These approaches address challenges such as false-lane detection and strive for improvement, particularly in nighttime scenarios.

2.1.3 Learning-Based Approach

The Learning-Based Approach involves predictive controllers for lane detection and tracking. Reinforcement learning, deep learning, and probabilistic models are integral components. Notable implementations include reinforcement learning-based lane change controllers and deep learning for object detection. Real-time probabilistic and deterministic prediction of lane changing is explored, often integrating LIDAR and camera input for enhanced accuracy.

2.1.4 General Observations

In vision-based systems, image smoothing at the initial lane detection and tracking stage plays a pivotal role in enhancing system performance. External disturbances, including weather conditions, vision quality, shadow, blazing, and internal disturbances, such as narrow or wide lane markings, contribute the most to algorithm performance drops.

Model-based approaches, which focus on robust lane detection and tracking, exhibit superior performance under various environmental conditions, with camera quality significantly influencing lane marking determination. The choice of filter, particularly the prevalent use of the Kalman filter, significantly affects the algorithm performance.

Reinforcement learning with model predictive control emerges as a promising choice to mitigate false-lane detection.

The majority of researchers (>90 %) rely on custom datasets, utilizing various camera types such as monocular, stereo, and infrared, where stereo cameras demonstrate superior performance. Frequent calibration remains essential for accurate decision-making in a complex environment.

One of the greatest challenges in current ADAS is the substantial impact of changes in environmental and weather conditions on system performance. Lane markers may be occluded during overtakes, and abrupt changes in illumination, such as emerging from a tunnel, affect image quality and system performance. The results unsurprisingly indicate the highest lane detection and tracking efficiency under dry and light rain conditions. Heavy rain significantly impacts the efficiency of lane marking detection and unclear or degraded lane markings contribute to poorer system performance. IMU (Inertia Measurement Unit) and GPS were shown to be able to improve RADAR and LIDAR distance measurement performance in these scenarios.

Addressing aforementioned challenges and leveraging advanced technologies such as deep learning and reinforcement learning present opportunities for future research and improvement.

2.2 General Approach to Lane Tracking

The individual methods mentioned in the previous section differ in the techniques they employ, but the general guideline for lane tracking and estimation can be summarized in several key steps.

1. Data measurement
2. Data preprocessing
3. Lane point detection
4. Tracking and estimation

Perhaps only the learning-based approach is different in sense that everything is computed using machine learning techniques (e.g., neural network). There these steps are not that useful, per say, but this is out of the scope of this work.

In the first step, all the necessary data are collected. For lane detection, there are basically only two types of sensors used that can capture the individual road lines, camera and LIDAR. Camera data come in the form of images with depth information necessary for the next steps. The output from the LIDAR is in the form of a point cloud, a 3D space filled with points associated with intensity of reflection.

The second step involves preprocessing the raw data. Here extraction of the region of interest (ROI) in front of the car (EGO vehicle) is done, as well as ground plane segmentation, filtering out everything other than the road itself.

Then comes the differentiating between the lanes and the ground or the road. There are many techniques to do so. Detecting lanes from camera images involves algorithms from computer vision domain, lane labeling, and others depending on real-time or offline processing. LIDAR exploits its ability to detect reflected light intensity, finding peaks in the detection that white or other lighter colors emit. The sliding window technique is a common approach to find the consecutive points belonging to an individual line, even if gaps are present.

In the last step, lane points are used as input to different tracking and estimation algorithms depending on the approach. It usually involves some type of lane fitting and smoothing out the results. Detection itself can be further improved by discarding outliers in the first place.

2.3 LIDAR vs. Camera in Autonomous Driving

Autonomous driving is a rapidly evolving field and the choice of sensors for environmental perception plays a pivotal role in the development of reliable autonomous systems. Among the array of sensors employed, LIDAR and cameras stand out as key technologies, each offering distinct advantages and facing unique challenges.

2.3.1 LIDAR: Precision in 3D Spatial Resolution

LIDAR, LIght Detection and Ranging, or Laser Imaging, Detection and Ranging, is renowned for its capacity to provide real-time, high-definition 3D graphics of the surrounding environment. By emitting pulse modulated light and measuring the time difference between emitted and reflected light, LIDAR calculates accurate distances and captures intricate details. This sensor excels in fast response, long detection distances, and high angular resolution.

However, challenges persist. Point cloud data obtained by LIDAR systems may contain noise due to factors such as acceleration, deceleration, and changes in driving direction [10]. Moreover, the high cost of LIDAR technology poses a barrier to mass production and widespread adoption.

2.3.2 Cameras: Cost-Effectiveness with Environmental Sensitivity

Cameras offer a cost-effective solution for environmental perception in autonomous vehicles. Even with their low price, they come with high resolution, fast speed, and the ability to recognize colors, making them a mainstream

vision sensor. They can also provide depth information in the right configuration. Cameras are widely deployed in various applications, including forward collision warnings, lane departure warnings, and traffic sign recognition systems.

Despite their advantages, cameras use natural light, which comes with its caveats, unlike LIDAR that uses its own light source. They are sensitive to adverse weather conditions, changes in illumination, and may struggle to identify distant objects in static images [11]. The potential for decreased recognition accuracy under challenging conditions remains a concern.

2.4 Kalman Filter

The Kalman filter is a recursive algorithm that estimates hidden states of a dynamic system from a series of noisy measurements. It can also predict the future state of the system based on past estimates. Originally introduced in the 1960s by Rudolf. E. Kálmán, the Kalman filter has become a cornerstone in various fields such as control theory, signal processing, navigation systems, and robotics. Since then, several versions of KF emerged, notably the Extended KF and its iterated type or Unscented KF, each aiming to improve performance in different scenarios.

2.4.1 Mathematical Foundation of Kalman filter

The Kalman filter is grounded in the state-space representation of dynamic systems. The state-space representation is often expressed as

$$\begin{aligned} x_{k+1} &= F_k x_k + G_k u_k + v_k \\ z_k &= H_k x_k + w_k \end{aligned} \tag{2.1}$$

where:

- x_k is the state vector at time k,
- u_k is the control input,
- z_k is the measurement vector,

- F, G, and H are state transition, input transition, and observation matrices, respectively,
- v_k and w_k are process and measurement noise, respectively.

There are two sets of equations that are being recursively calculated in a *predict-correct* loop (see figure 2.1). They have been given many names, depending on the field where applied. Here they will be referred to as *state prediction* and *measurement update*.

State prediction step:

$$\begin{aligned} \hat{x}_{k+1|k} &= F_k \hat{x}_{k|k} + G_k u_k \\ P_{k+1|k} &= F_k P_{k|k} F_k^T + Q_k \end{aligned} \tag{2.2}$$

Measurement update step:

$$\begin{aligned} S_{k+1} &= H_{k+1} P_{k+1|k} H_{k+1}^T + R_{k+1} \\ K_{k+1} &= P_{k+1|k} H_{k+1}^T S_{k+1}^{-1} \\ \hat{x}_{k+1|k+1} &= \hat{x}_{k+1|k} + K_{k+1}(z_{k+1} - H_{k+1}\hat{x}_{k+1|k}) \\ P_{k+1|k+1} &= P_{k+1} - K_{k+1} S_{k+1} K_{k+1}^T \end{aligned} \tag{2.3}$$

where:

- $\hat{x}_{k+1|k}$ is the state prediction,
- $P_{k+1|k}$ is the state prediction covariance,
- $Q = cov(v)$ is the process noise covariance,
- S is the innovation covariance,
- K is the Kalman (filter) gain,
- z_{k+1} is the measurement,
- $\hat{x}_{k+1|k+1}$ is the updated state estimate,
- $P_{k+1|k+1}$ is the updated state covariance,
- $R = cov(w)$ is the measurement noise covariance.

After initialization at $\hat{x}_{0|0}$, $P_{0|0}$ the state is predicted based on the underlying state-space model, together with the covariance matrix (2.2). This is sometimes referred to as state and covariance extrapolation at time update step.

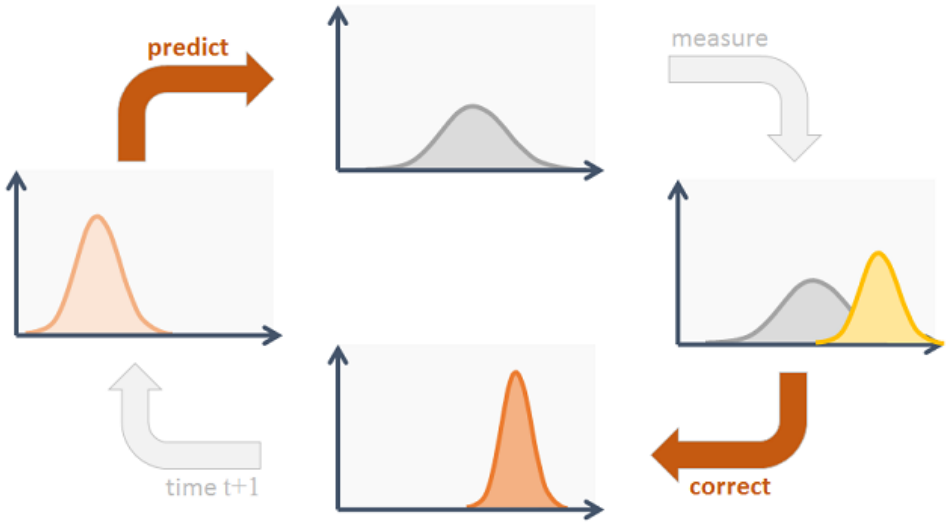


Figure 2.1: Predict-correct loop of a Kalman filter. Visualized on Gaussians, the KF first predicts next state from provided state transition, incorporates the measurement, corrects the state estimate, and repeats (from Object Tracking: Kalman Filter with Ease available at `https://www.codeproject.com`).

Essentially, the KF predicts the behavior of the system at the next time step and also provides the uncertainty of the prediction.

In the next update step (2.3), after the measurement is received, the KF corrects the prediction and the uncertainty of the current state. As a result, it provides a new state of the system based on a combination of some physical model updated by a measurement at that time, infused with their respective uncertainty. This is then repeated at some sampling time rate until the measurement stops.

This recursive nature allows the filter to continuously update its estimate as new measurements become available, making it well suited for real-time applications. Under the assumptions of Gaussian initial state and all the noise, the KF is the **optimal minimum mean square error state estimator**. If the random variables are not Gaussian and if some assumptions are lowered, the KF is the **best linear MMSE state estimator** [12]. This makes it widely applicable in situations where accurate state estimation is crucial. However, when either the system model or the measurement model is non-linear, the KF cannot handle it. This is when its extended version comes in.

2.4.2 Extended Kalman Filter

The extended Kalman Filter (EKF) is an extension of the Kalman filter designed to handle nonlinear system dynamics and measurements. Similar to the KF, it relies on the state-space representation. The dynamics of the system is expressed as follows:

$$\begin{aligned} x_{k+1} &= f(k, x_k, u_k) + v_k \\ z_k &= h(k, x_k) + w_k \end{aligned} \tag{2.4}$$

where:

- x_k is the state vector at time k,
- u_k is the control input,
- z_k is the measurement vector,
- $f(\cdot)$ represents the non-linear state transition function,
- $h(\cdot)$ represents the non-linear measurement function,
- v_k and w_k are process and measurement noise, respectively.

The logic behind and the steps remain almost the same as in the case of KF. The difference is in obtaining the predicted state $\hat{x}_{k+1|k}$. The nonlinear functions in 2.4 need to be linearized using Taylor series expansion around the latest estimate $\hat{x}_{k|k}$. Omitting the third order and higher order terms (HOT) yields second-order EKF, omitting also the second term yields first-order EKF. The latter case will be described.

$$x_{k+1} = f(k, \hat{x}_{k|k}) + F_k(x_k - \hat{x}_{k|k}) + HOT + v_k \tag{2.5}$$

where

$$F_k = \left.\frac{\partial f(k)}{\partial x}\right|_{x=\hat{x}_{k|k}} \tag{2.6}$$

is the Jacobian of f, evaluated at the latest state estimate. Similarly, the measurement is

$$z_{k+1} = h(k+1, \hat{x}_{k+1|k}) + H_{k+1}(x_{k+1} - \hat{x}_{k+1|k}) + HOT + w_{k+1} \tag{2.7}$$

where

$$H_{k+1} = \left.\frac{\partial h(k+1)}{\partial x}\right|_{x=\hat{x}_{k+1|k}} \tag{2.8}$$

is the Jacobian of h, evaluated at the latest state estimate. The difference in algorithm is in incorporating the state and measurement functions and corresponding Jacobians.

State prediction step:

$$\begin{aligned} \hat{x}_{k+1|k} &= f(k, \hat{x}_{k|k}, u_k) \\ P_{k+1|k} &= F_k P_{k|k} F_k^T + Q_k \end{aligned} \tag{2.9}$$

Measurement update step:

$$\begin{aligned} S_{k+1} &= H_{k+1} P_{k+1|k} H_{k+1}^T + R_{k+1} \\ K_{k+1} &= P_{k+1|k} H_{k+1}^T S_{k+1}^{-1} \\ \hat{x}_{k+1|k+1} &= \hat{x}_{k+1|k} + K_{k+1}[z_{k+1} - h(k+1, \hat{x}_{k+1|k})] \\ P_{k+1|k+1} &= P_{k+1} - K_{k+1} S_{k+1} K_{k+1}^T \end{aligned} \tag{2.10}$$

The accuracy of EKF predictions depends on the quality of the linearization. In general, any non-linear transformation will introduce some bias. Calculations based on the Taylor expansion will omit the HOT terms, inevitably introducing error. The second-order EKF will perform better, but some error will still persist. Another bias is introduced, when calculating Jacobians (or Hessians in the second-order case). The evaluation is done at the estimated or predicted state, because the exact state is unavailable. However, if the initial noises and errors are not too large, the EKF performs well [12].

In situations with highly non-linear behavior, the inherent approximations can lead to divergence of the EKF, rendering the filter unusable. Another type of KF called Iterative Extended Kalman Filter (IEKF) is capable of handling these cases to some extent by iterating over the one measurement, until the difference between two consecutive predictions becomes smaller than some threshold.

2.4.3 Kalman Filters in Advanced Driver Assistance Systems

In the context of ADAS, the KF is particularly valuable for several reasons. Car sensor measurements are inherently noisy due to environmental factors, sensor imperfections, and other uncertainties. The KF excels at noise reduction, effectively separating the true signal from the noise. This leads to smoother and more accurate estimates of the system state. ADAS algorithms require predictions about the future state of the vehicle based on current and

past measurements. By incorporating a model of the dynamics of the system, the KF can predict the future state. This prediction aids in anticipating the vehicle's behavior and improving response times. The KF can also integrate data from different sensors with varying accuracy. It excels in sensor fusion by dynamically weighting sensor inputs based on their reliability. This ensures that more accurate sensors contribute more to the overall estimation. The KF is well suited for dynamic environments. Its recursive nature allows it to continuously update estimates, making it adaptable to changes in the vehicle's surroundings. Brokar et al. demonstrated how KF is able to ignore minor perturbations and smooth out the tracking.

In lane-keeping assist systems, KFs help predict the vehicle's future position and adjust steering interventions accordingly, considering factors like road curvature and vehicle dynamics. Collision avoidance systems use KFs to fuse data from radar and lidar sensors to accurately track the positions of surrounding vehicles, enabling timely collision warnings or interventions. Adaptive cruise control leverages the ability of KF to predict future positions of nearby vehicles, enabling smoother and more accurate speed adjustments to maintain a safe following distance. Overall, the lane tracking task is where the KF is still used the most [8].

2.5 RANSAC Algorithm

Random Sample Consensus (RANSAC) is an algorithm widely used for robust model fitting in the presence of outliers (i.e., data that significantly differ from other observations). Its primary objective is to estimate a model from a set of data points, even when a significant portion of the data is contaminated by outliers. This is particularly crucial in scenarios where traditional algorithms may fail due to the presence of noise or incorrect measurements. A pseudocode of RANSAC is described in algorithm 1.

The choice of the model depends on the problem at hand. It could be a line, a plane, a circle, or any other geometric shape, for example. The model is typically represented by a set of parameters M and the goal is to find the optimal values for these parameters M_{best}. S is a minimal subset of data points D required for the fit (e.g., 2 points to fit a line).

The error function evaluates how well the model explains each data point and helps to identify inliers and outliers. For example, find the distance from the data points to a fitted line that falls within the inlier threshold ϵ.

Algorithm 1: RANSAC

Data: data points D, model parameters M, inlier threshold ϵ, number of iterations num_iter
Result: best-fitting model M_{best}

```
M_best ← ∅;
num_inliers ← 0;
for i ← 1 to num_iter do
    Randomly select a minimal subset S ⊂ D;
    Fit a model M using S;
    inliers ← Find inliers by measuring the error between D and M;
    if number of inliers > max_inliers then
        M_best ← M;
        max_inliers ← inliers;
return M_best;
```

The model with most inliers (least outliers) is chosen as the best-fitting one. The threshold is based on a specific application requirement or experimental evaluation.

The number of iterations num_iter can be roughly determined by the desired probability of successfully finding the right solution p (commonly chosen as 0.99, i.e. 99 %) and by the probability of choosing an inlier in the data w (roughly estimated by the number of inliers / number of points, e.g. 0.85 meaning 85 % of inliers). By assuming that all S points are selected independently and w^S is the probability that all S are inliers, then $1 - w^S$ is the probability that at least one of the S points is an outlier. This to the power of N (iterations of the algorithm) is the probability of never selecting S inliers, which is the same as the probability of not finding the right solution, which can be expressed as

$$(1 - w^S)^N = 1 - p \,. \tag{2.11}$$

Taking logarithm of both sides results in desired outcome.

$$N = \frac{\log(1-p)}{\log(1-w^S)} \tag{2.12}$$

However, this is often not sufficient, and in practice the number of iterations is determined by adding the standard deviation of N or multiplying by it. Again, it depends on the situation, needed confidence, and time requirements.

$$num\_iter = N + \frac{\sqrt{1-w^S}}{w^S} \,. \tag{2.13}$$

Although RANSAC is a robust estimator [13], its performance depends on the appropriate setting of the above parameters. Additionally, there is no upper

bound on the time it takes to compute these parameters. Any limit to the number of iterations results in a nonoptimal solution. Moreover, the algorithm assumes that a sufficient number of inliers exist to accurately estimate the model. To handle these problems, optimal RANSAC was proposed [14].

2.5.1 RANSAC in Advanced Driver Assistance Systems

ADAS algorithms face challenges in processing sensor data, particularly in the presence of outliers, noise, and varying environmental conditions. RANSAC can address these challenges in several ways.

RANSAC robustly estimates the model parameters by iteratively fitting to subsets of data. This process inherently rejects outliers, contributing to more accurate object detection that can be compromised by erroneous measurements in the first place. The same applies for lane detection, where lane markings can be obscured, faded, or interrupted, for example, by shadows or debris on the road. For example, Lu et al. used Gaussian distribution RANSAC and achieved great results [15]. Other tasks, such as sensor calibration or object tracking, can benefit from the use of RANSAC as well.

Chapter 3

Methods and Implementation

This chapter gives a complete overview of the lane tracking and estimation methods and how they were implemented. In the first part, the general methodology is outlined, followed by a description of the database used for the testing. Then comes the implementation part, detailing the individual steps and methods, namely preprocessing, lane sorting, Kalman filter, RANSAC, and so on. The next section presents the results of this work and what is the methodology behind the error evaluation. The last part discusses the results in a broader context, highlighting the advantages, drawbacks, and future prospects.

3.1 Lane Tracking and Estimation Methodology

The workflow of lane detection and tracking presented in this work is shown in the diagram 3.1. The whole process can be divided into two sections, preprocessing and tracking and estimation.

Firstly, the PandaSet LIDAR point cloud is converted into a format that can be loaded into the MATLAB environment. There, the lane markings are separated from the rest of the points, using semantic segmentation labels. The transformation from the world coordinates into the EGO coordinates is applied. Next, any points outside the defined region of interest (ROI) are filtered, leaving only the lanes in front of the EGO vehicle. At this point, all the selected points belong to the same set, and there is no differentiation between

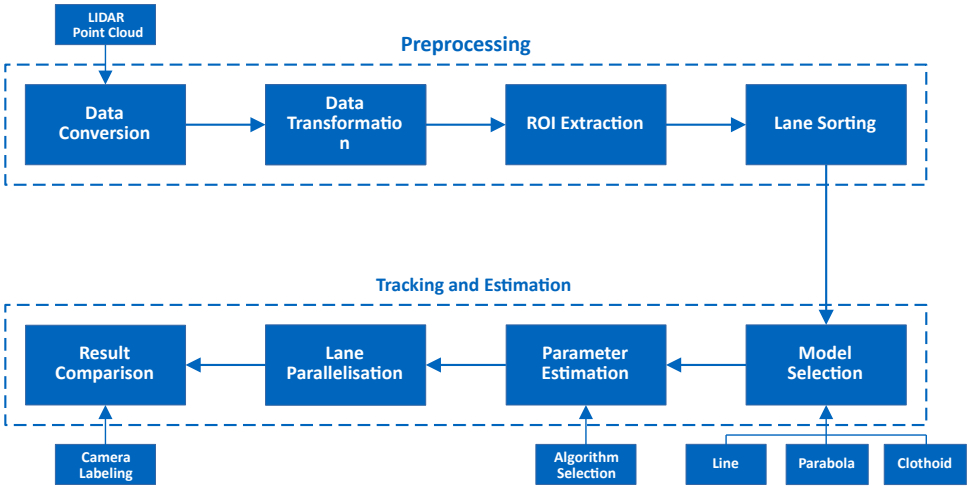


Figure 3.1: Steps of a pipeline presented in the thesis, comprising of two main parts, preprocessing and tracking and estimation.

the individual lines of the road. In the later stages, the algorithms require input in the form of a single road line to estimate its parameters. Therefore, lane sorting is performed next, depending on the data characteristics, either by window sliding technique, or via a simpler method. This concludes the preprocessing part.

Tracking and estimation come next, starting with the selection of the underlying physical model. This model represents the shape of the road line to be estimated using the following algorithms. Three models to compare were chosen, which essentially represent first-, second-, and third-degree polynomials. These are a line, a parabola, and a clothoid (i.e., Euler spilar; see figure 3.2). The clothoid model is widely used in literature as a shape that the road forms and is commonly simplified to a third-degree polynomial [16].

After choosing the model, an estimation of its parameters is performed. Two main algorithms were selected to do so, Kalman filter and RANSAC. There exist many strategies for approaching LIDAR-based lane tracking using the KF, but they usually focus on some additional external measurement. For example, Y. Zhang et al. used GPS / IMU system to build a curb prediction model with a tracking algorithm [17], or the lane marking detection method for localization within an HD map proposed by F. Ghallabi et al. [18].

The approach presented in this work aims to track the lane curvature parameters in the frame of the EGO vehicle only with the LIDAR itself. As it turns out, this requires only the standard KF (see section 3.3.3). Additionally, an adaptive KF algorithm is proposed. Since the car is at the origin, its distance to the estimated lines can be easily calculated later. Furthermore,

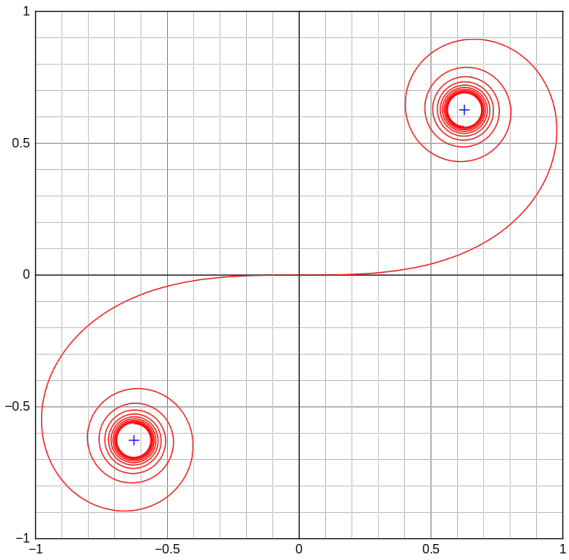


Figure 3.2: A double-end Euler spilar or clothoid is a curve with linearly changing curvature depending on the curve length. By AdiJapan - Own work, CC BY-SA 3.0, `https://commons.wikimedia.org/w/index.php?curid=22191870`

the accuracy of the estimation relies on the correct calibration of the LIDAR and not on GPS / IMU, which might not provide such accuracy as the LIDAR does or may not work under ground. M. Thuy et al. did something similar, claiming to use the EKF [16].

In addition, the RANSAC algorithm and its combination with the KF is investigated. Using the RANSAC alone for the lane tracking and estimation is not ideal due to its stochastic nature, but in a symbiosis with the KF might prove useful in certain driving scenarios.

After the parameter estimation, the fact that the individual road lines are parallel to each other can be used to make the estimate more robust. In the last step, the errors of individual algorithms can be evaluated and compared to each other. This requires the use of a third metric. For this purpose, camera data are used, and details of how the error is measured and how the algorithms are implemented are discussed in the following chapters.

3.2 Overview of PandaSet Database

PandaSet by Hesai and Scale AI is an open-source dataset for autonomous driving and machine learning use in mind. It combines Hesai's LIDAR sensors with Scale AI's data annotation. It features more than one hundred 8 second scenes with over 16,000 LIDAR sweeps and 48,000 camera images. Scale 3D sensor fusion segmentation provides a combination of cuboid and semantic segmentation with 28 annotation classes and 37 semantic segmentation labels. The EGO set-up features a mechanical spinning LIDAR, forward-facing LIDAR, 6 cameras, and on-board GPS and IMU. PandaSet aims to showcase complex urban driving scenarios in a variety of daytimes and lighting conditions, all on 2 routes in Silicon Valley [7].

This dataset was carefully chosen because it is

- **open-source**,
- **labeled**,
- **user-friendly**.

As stated before, Waykole et al. found in their large meta-analysis a very high ratio of custom data used for lane detection and tracking [8]. This poses a problem, since this approach results in something specifically tailored to each dataset. Replicating the results or comparing to other methods is afterwards very difficult, if not impossible. Nevertheless, many high quality publicly available datasets exist today, for example, Audi's A2D2 driving dataset, or K-Lane LIDAR lane data set. Some additional work is required to understand how to use them and acquire the data needed, but the benefits are clear. PandaSet is available for both academic and commercial use.

PandaSet comes with a complex label taxonomy done by combining both human work and smart tools. This results in consistently higher accuracy than if done independently. There are 37 classes of semantic segmentation, including lane markings, which is of greatest importance for this work. Dealing with raw data presents separate problems, such as ground segmentation and lane detection, and this thesis is mainly focused on tracking and estimation. Being able to get only points from LIDAR point cloud belonging to road lines presents a huge advantage.

Being user-friendly saves a lot of time when trying to incorporate such a

dataset into one's work. All the code in this thesis was developed in MATLAB environment, but finding publicly available data that could be loaded into MATLAB workspace was shown to be impossible. PandaSet has very straightforward tutorials in the devkit available on GitHub (`https://github.com/scaleapi/pandaset-devkit`), easily followed even with basic Python skills. Converting loaded data to some format readable to MATLAB is afterwards possible. A framework for such conversion was developed for this work and is described in the MATLAB live script tutorial available at thesis GitHub `https://gitlab.fel.cvut.cz/veskrdan/lidar-lane-estimation`.

3.2.1 Data Structure

Data of each sequence are made up of annotations, camera data, LIDAR data, and meta files. Annotations include cuboids and semantic segmentation, however, the latter is not included in every sequence. In the camera folder, there are images in JPG format from all six cameras around the car. Each camera has its own parameters included. These are camera intrinsics parameters (fx, fy, cx, cy), poses with coordinates (x, y, z) and heading (w, x, y, z) in the world frame, and lastly timestamps. The LIDAR folder includes point clouds in the world coordinates, poses, and timestamps with the same structure as in the camera case. The dataset also has GPS coordinates with its timestamps.

The fact that all the information is stored in the world frame is a bit inconvenient. Lane tracking and estimation are done in the EGO frame, but the PandaSet devkit has a tutorial on how to make the transformation. The method of projecting points from the world frame onto the camera images is shown as well. However, to do so with the obtained results from tracking, a sequence of transformations from EGO frame, back to the world frame, to camera frame, and then into the camera image needs to be performed. This introduces some error into the final estimate. Furthermore, the camera data come in the form of JPG images with no depth information. This makes comparing the results more difficult and is further discussed in section 3.4.1.

3.3 Implementation

In this part, details of how the aforementioned steps were implemented will be discussed. Mathematics behind the key models and transformations will be

shown, as well as an overview of the algorithms and their references in the code. To better understand the structure of the code and how to use it to replicate the results, a MATLAB live script *Tracking_and_Estimation_tutorial.mlx* is available at the GitHub page `https://gitlab.fel.cvut.cz/veskrdan/lidar-lane-estimation`.

3.3.1 Preprocessing

The PandaSet LIDAR point cloud is in PKL.GZ format, which is converted in Python *convertor.py* to a CSV file, together with the other parameters that are in JSON format. This can then be loaded into MATLAB and further processed (see figure 3.3).

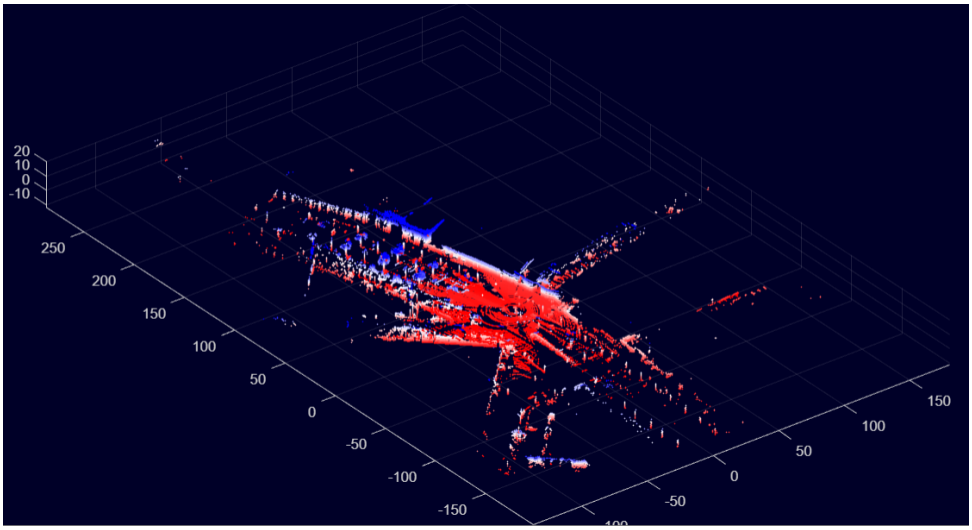


Figure 3.3: Raw point cloud example

The CSV files are loaded into tables in MATLAB and points that belong only to a label *Lane Line Marking* (specified in *classes.json*) are extracted (*label_extractor.m*, see figure 3.4). Here, the transformation from the world coordinates into the EGO coordinates is also done. The transformation matrix T is formed using the LIDAR heading as a quaternion to form a rotation matrix R:

$$\begin{bmatrix} h_w & h_x & h_y & h_z \end{bmatrix} \Longrightarrow R \tag{3.1}$$

Together with the LIDAR position, the transformation matrix can be assembled, and the point cloud can be converted to the EGO frame at the given

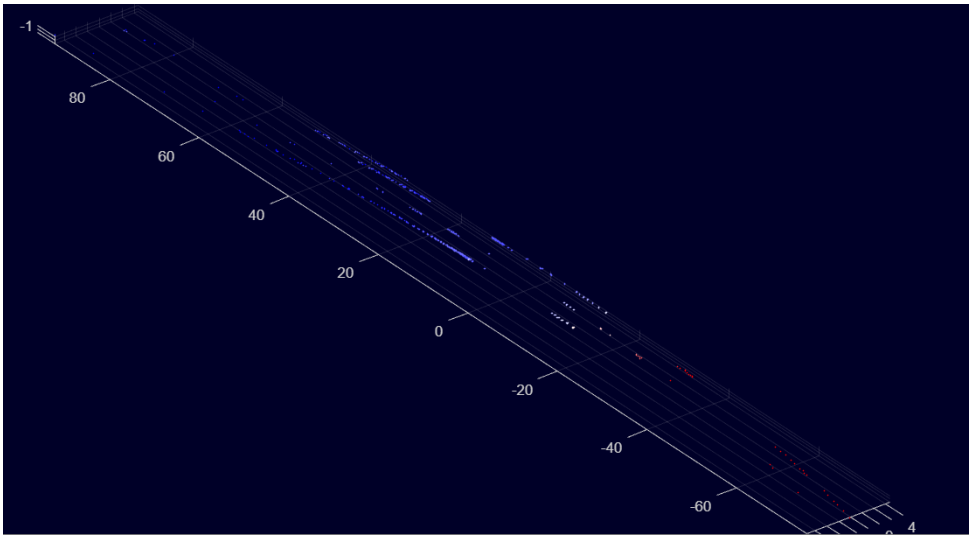


Figure 3.4: Point cloud of only the lane line markings.

timestamp. An example can be seen in figure 3.5.

$$T = \begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & p_x \\ R_{2,1} & R_{2,2} & R_{2,3} & p_y \\ R_{3,1} & R_{3,2} & R_{3,3} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{EGO} = inv(T) \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{WORLD} \tag{3.2}$$

Filtering points outside the ROI comes next (*extract_ROI*). Anything more than 8 meters away left and right, further than 60 meters in front of the EGO, and anything behind it is discarded. The resulting part of the road is directly in front of the EGO vehicle and crops the points that are at the very edge of the LIDAR detection capabilities (see figure 3.5).

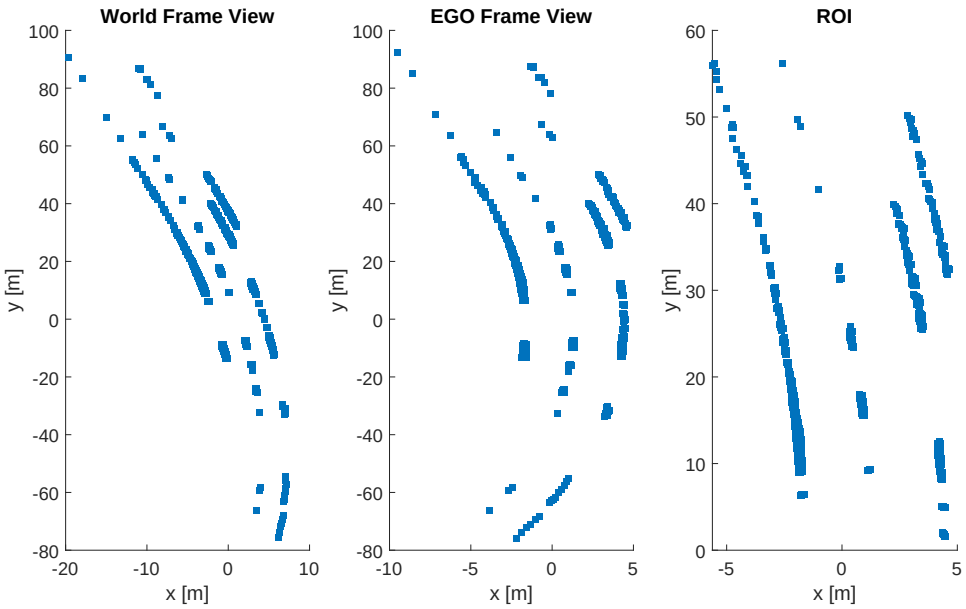


Figure 3.5: Transformation of point cloud sequence frame from the world coordinates (left) into the EGO coordinates (middle) and ROI extraction (right).

3.3.2 Lane Sorting

To differentiate between individual lines on the road, some kind of sorting algorithm must be applied. The set of points forming the lane markings, as can be seen in the figure 3.5, can then be categorized into *Line 1* - *Line 4*, for example. Each line itself can then be inputted as an input to the estimation algorithm in the next step.

For the simplest case of a straight road, such as a highway, the sorting can be performed simply by filtering by the x-axis. For example, in the figure 3.6, every point with $x < 0$ will belong to the left line and with $x > 0$ to the right line. This is indeed very simple and a quick way to prepare the points for estimation, but in practice unusable. It will fail for more than two lines, since determining the dividing line other than $x = 0$ is unreliable and the EGO vehicle can also shift. If any curvature is present, this approach will fail completely.

A common way to solve this problem is to use a technique called *window sliding*. The principle is to use a defined ROI (window) initialized at the beginning of the line to find any points within it and then slide the window in a specific way to find the next points on the line. Essentially, sliding along the line and finding any points along the way.

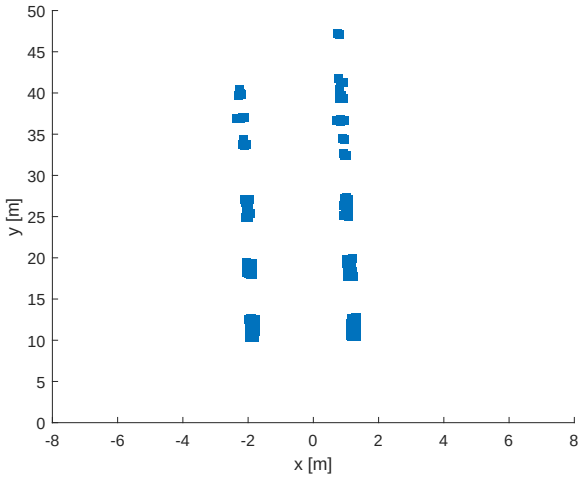


Figure 3.6: Example of straight road lines.

There are different types of window sliding technique. In a real-time scenario, after measuring the LIDAR point cloud, ground segmentation, and ROI extraction, the windows can be initialized at intensity peaks of the points detection and slide onward to find the individual lines. The principle for labeled lines here does not have to deal with all of these steps, but the sliding window technique remains the same. The pseudocode implemented is described in algorithm 2 (*sort_lines.m*).

The window resolution is set experimentally, depending on the road setting (for example, 1x1 meter). The number of windows is then determined by how many of them fit along the y-axis. If the point cloud is correctly filtered, initializing the window at the first found point in front of the EGO vehicle expects to find all points on that particular line. After the whole line is found, the next first point should belong to a new line. If any points within the window are found, the mean of them (essentially center of gravity) is calculated, and the window slides along the y-axis with res_Y and is shifted in the x-axis to the new center (figure 3.7 left).

If no points are found, the position of the next window needs to be predicted (figure 3.7 middle). This is done by fitting a polynomial through the centers of the previous windows. If there are not enough centers, i.e. at the beginning of the line, the new window is simply shifted only along the y-axis. After the gap is breached and the points are found again, the shifting is no longer predicted. When all of the windows are shifted, the new window is initialized at the next first point (figure 3.7 right). Through experimentation, it was found that predicting using the first-degree polynomial yielded better results. As the curvature of the road in the tested sequences is not exceptionally high (e.g., 90° turn), the higher degree prediction often missed the next points (see figure 3.8). The percentage of unsorted points for the sequence presented in

Algorithm 2: Sort Lines

Data: unsorted points U
Result: sorted points S

Init: $res_X, res_Y, num\_windows$;
$S \leftarrow \emptyset$;
while *not all lines are found or no points left* **do**
    Find the first point along y-axis $P \subset U$;
    window $\leftarrow$ Init. with center at P and size res_X, res_Y;
    line $\leftarrow \emptyset$;
    for *all windows in num_windows* **do**
        points $\leftarrow$ Find points inside the window;
        if *points found* **then**
            line $\leftarrow$ *points*;
            center $\leftarrow$ Find mean of *points*;
            window $\leftarrow$ Slide window to new *center*;
            Remove *points* from U
        else
            window $\leftarrow$ Predict new window position;
    $S \leftarrow$ *line*;
    Save and move to next line;

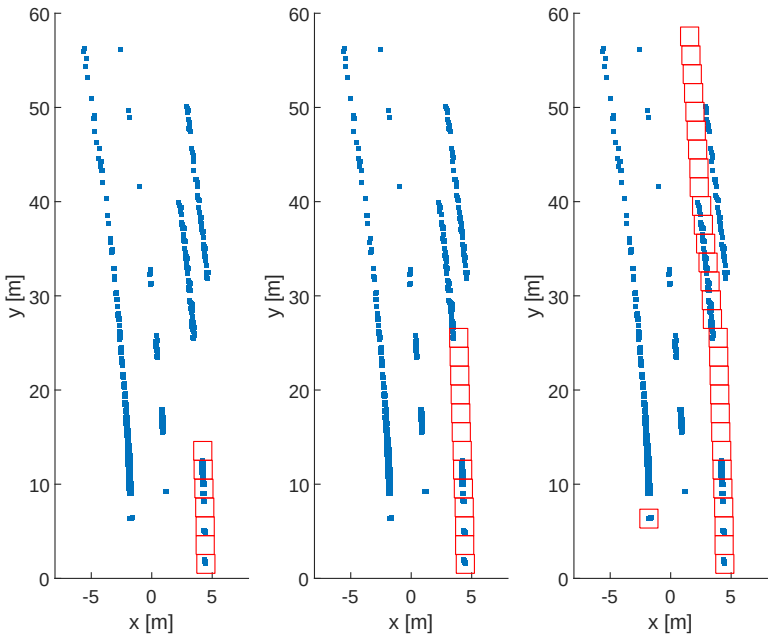


Figure 3.7: Example of implemented sliding window technique in action. On the left, the first window is initialized and slides along the means of the found points. If no points are found (middle), the new window shift is predicted. After the whole line is found, new window is initialized at the next first point (right).

the figure was 2.05 % for the second degree polynomial and 0.74 % for the first degree.

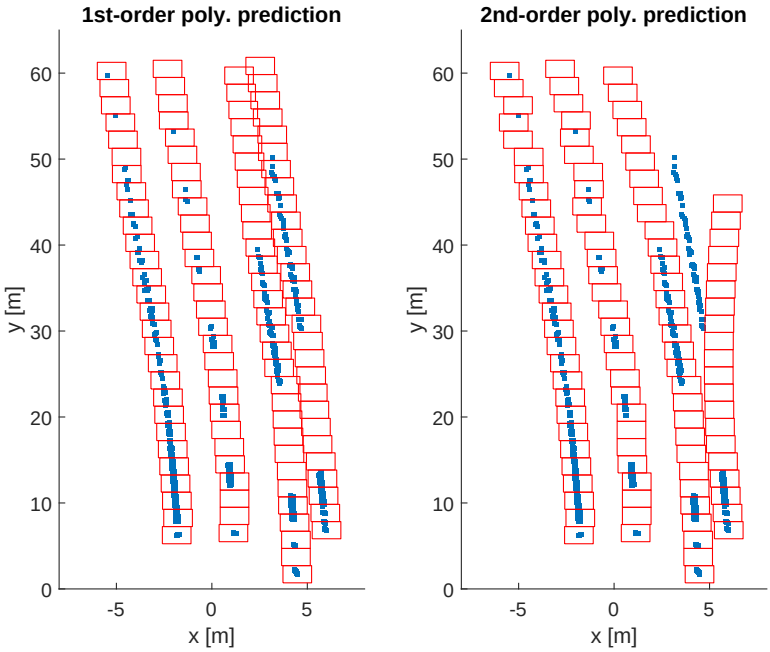


Figure 3.8: Difference between new window prediction based on first degree polynomial (left) and second degree polynomial (right). The first degree polynomial resulted in fewer errors when breaching the gaps in detection.

After the sorting is done, the points are divided into individual lines, as can be seen in the figure 3.9. The lines are also labeled, so that the points belong to the same line in each frame of the sequence.

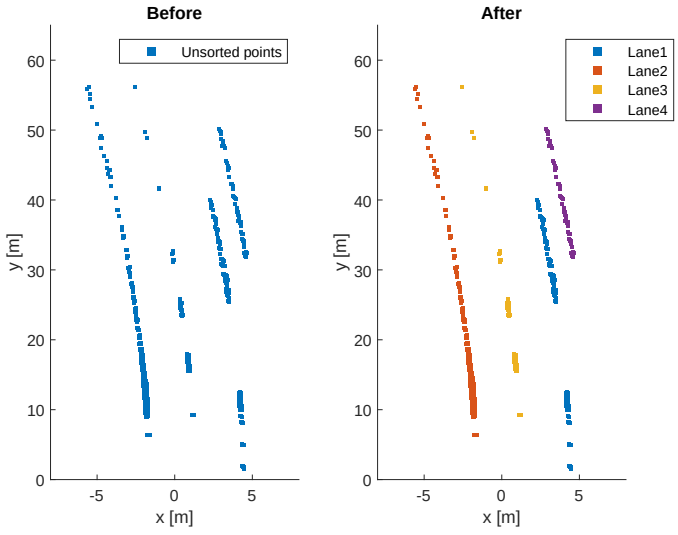


Figure 3.9: Result after the sliding windows technique sorting. The input to the algorithm is on the left, result on the right.

3.3.3 Kalman Filter

The implementation of the Kalman filter differs according to the selected lane model. Three models were chosen to compare with each other:

$$\begin{aligned} \textbf{Line:}\ & y = ax + b \\ \textbf{Parabola:}\ & y = ax^2 + bx + c \\ \textbf{Clothoid:}\ & y = y_0 + \frac{1}{2}c_0x^2 + \frac{1}{6}c_1x^3 \end{aligned} \tag{3.3}$$

The clothoid model is a third-order approximation of the original equation characterizing a clothoid [16]:

$$\begin{aligned} x(t) &= \int_0^t \cos\left(\frac{\pi}{2}u^2\right)\,du \\ y(t) &= \int_0^t \sin\left(\frac{\pi}{2}u^2\right)\,du \end{aligned} \tag{3.4}$$

Since the EGO vehicle is commonly presented as moving in the direction of the y-axis, the x and y in 3.3 will be switched from now on. The equations above represent the measurement model of the KF, where x,y are the LIDAR measurements, assumed to be independent of the state. The KF estimates constant parameters of the measurement models, which results in the following states s and transition matrices F:

$$\begin{aligned} \textbf{Line:}\ & F = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad s = \begin{bmatrix} a \\ b \end{bmatrix} \\ \textbf{Parabola:}\ & F = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad s = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \\ \textbf{Clothoid:}\ & F = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad s = \begin{bmatrix} x_0 \\ c_0 \\ c_1 \end{bmatrix} \end{aligned} \tag{3.5}$$

The states s in all of the measurement models are in linear relation, multiplied only by some coefficient after the measurement comes in. Since the measurement is $[y, x]$ and

$$\begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} y \\ ay + b \end{bmatrix} \tag{3.6}$$

The $y = y$ term can be omitted (similarly for all models). This leads to the following measurement matrices H dependent only on the measurement y:

$$\textbf{Line: } H = \begin{bmatrix} y & 1 \end{bmatrix}$$

$$\textbf{Parabola: } H = \begin{bmatrix} y^2 & y & 1 \end{bmatrix} \tag{3.7}$$

$$\textbf{Clothoid: } H = \begin{bmatrix} \frac{1}{6}y^3 & \frac{1}{2}y^2 & 1 \end{bmatrix}$$

The states were initialized at $s = 0$ and the state covariance P with the noise covariances Q, R were treated as 'tuning knobs', which means that their value was found experimentally (the real noise values of the models and LIDAR measurements are not known). The general guideline was to emphasize trust in the physical model over the measurement model, resulting in lower process noise values in the Q covariance matrix and higher measurement noise values in the R covariance matrix.

The complete KF equations were implemented according to 2.3 and 2.3 (*track_lines_KF.m*). The estimation algorithm for the entire sequence is described in pseudocode 3. Each sequence of PandaSet contains 80 frames of LIDAR point clouds (one frame is in figure 3.9, for example). After the initialization, the KF state is estimated for each line in the frame, resulting in polynomial coefficients of the chosen model and updated state covariance matrix. Each point of the line is a new measurement that corrects the predicted state. After all lines in the frame are estimated, the information is carried on to the next frame, further improving the estimation. An example from the run of the algorithm can be seen in figure 3.10. The advantages of curved models over the line model are obvious here. The improvement of the estimation over several frames is visible as the overall position of the estimates gets closer to the LIDAR points.

Algorithm 3: Kalman Filter Loop

```
Data: sorted lane data S, underlying model M
Result: estimated lines L
params ← Initialize KF parameters based on the model M;
for all frames in sequence do
    for all lines in S do
        for all points in a line do
            predict(params) ← Update params in prediction step;
            z ← Get new measurement from line points;
            correct(params, z) ← Update params in correction step;
        L ← Save estimated states from params;
```

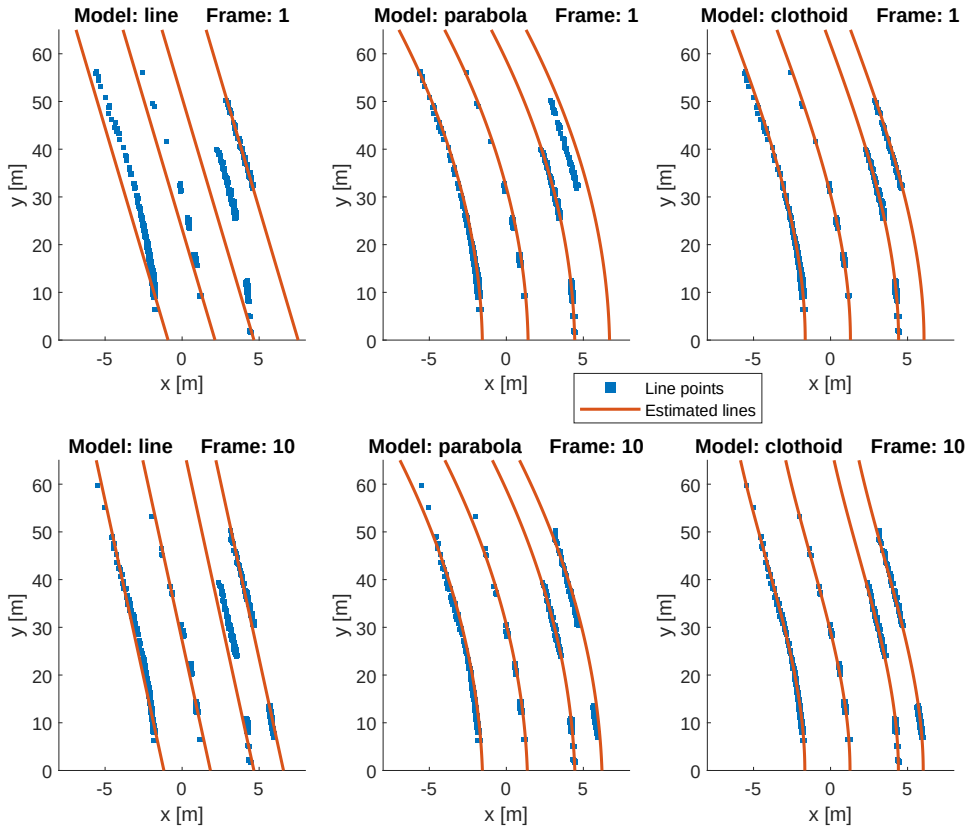


Figure 3.10: Example of the Kalman filter with each of the three models. The top row shows the results after just the first frame, the bottom row after ten frames.

Parallelisation

On a typical road, the driving lanes with the line markings are parallel to each other. As each line in the KF loop is estimated separately, none of them are parallel as a result. This can be corrected, further improving the estimates.

The best fitting line of each frame is chosen, and the rest is made parallel to it (*make_parallel.m*). This is done by finding the lowest RMSE (root mean square error) of a difference in the x displacement as follows:

$$\begin{aligned} \begin{bmatrix} a & b & c \end{bmatrix} &\leftarrow \text{result of KF loop} \\ \begin{bmatrix} x & y \end{bmatrix} &\leftarrow \text{line points from LIDAR} \\ \hat{x} &= polyval(a, b, c, y) \end{aligned} \tag{3.8}$$

The estimated line parameters in each frame are evaluated at the line points $[x, y]$ that form an estimated $\hat{x}$. The RMSE is then calculated, representing

a lateral displacement error per meter.

$$E_x = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{x} - x)^2} \tag{3.9}$$

The result is then weighted by adding a value inversely proportionate to the number of points on the line. If the best fit was made from a line made only from a few detections, it would make its score worse. When the best line is found, the parameters of the other lines are adjusted according to it, except for the coefficient responsible for the placement of the line on the x-axis. An example of how the estimated lines look before and after the process of parallelisation can be seen in figure 3.11.

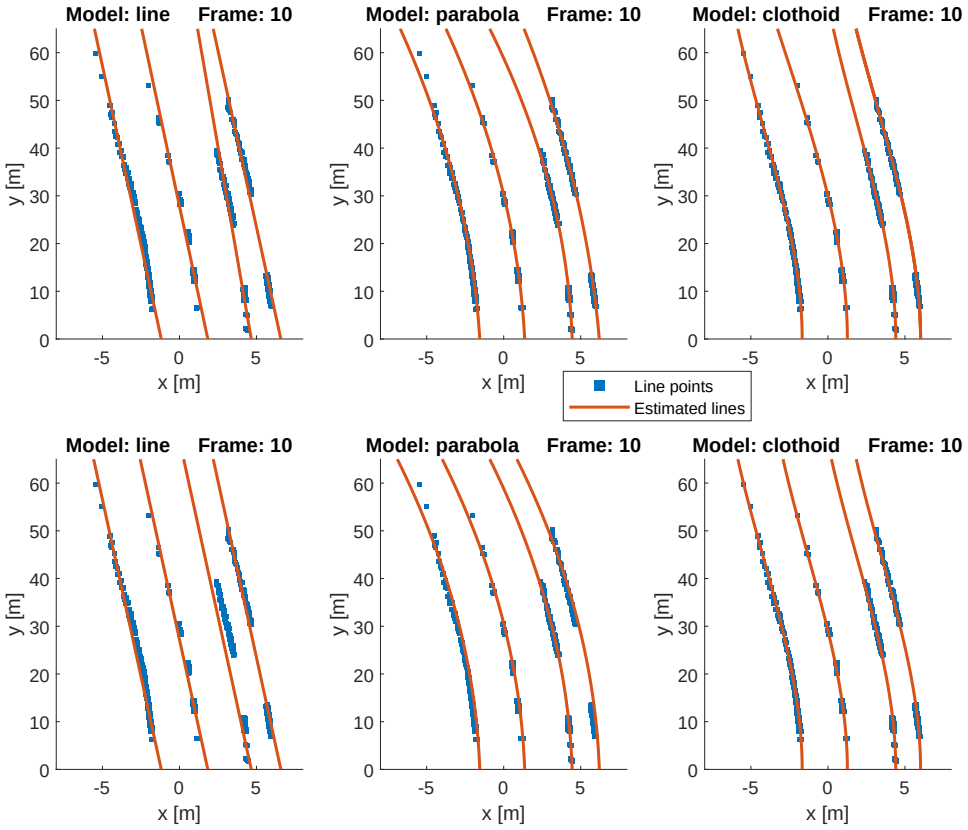


Figure 3.11: Example of the estimates before (top row) and after (bottom row) parallelisation.

Although this error is based on the measurement that goes into the estimate itself, it reflects the best estimated shape of all the lines. If a significant portion of one line is missing (e.g. obstructed by a different car), the estimate of it will be very poor. If a line measurement from a camera or other sensor was available, this error could be based on it, making the estimate even more robust. The potential benefits of this approach will be discussed in chapter 4.

3.3.4 RANSAC

The RANSAC algorithm loop is similar to the KF loop 3, except the inner most for-cycle is replaced by the function to perform the RANSAC (*track_lines_RANSAC.m*). Similarly as before, the algorithm depends on the model to be fitted.

The main idea is to solve a system of equations, which results in a polynomial that passes through the given set of points (i.e., polynomial interpolation). This is formulated by means of the Vandermonde matrix as follows.

$$Va = y$$
$$\begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^{n-1} \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} p(x_1) \\ p(x_2) \\ \vdots \\ p(x_m) \end{bmatrix} \tag{3.10}$$

where x and $y = p(x)$ are the points to be fitted, a is the vector of coefficients to be found and V is the Vandermonde matrix. Since the highest degree in the models is 3 and the maximum of points S is also 3 (the clothoid has $x^1 = 0$), the matrix will be small. Also, the points are chosen to be distinct, resulting in a square matrix with nonzero determinant, which is invertible. The solution is to find solving for coefficients a as

$$a = V^{-1}y \tag{3.11}$$

The equations for the three models to be solved are as follows

$$\textbf{Line:} \quad \begin{bmatrix} b \\ a \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \end{bmatrix}^{-1} \cdot \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

$$\textbf{Parabola:} \quad \begin{bmatrix} c \\ b \\ a \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{bmatrix}^{-1} \cdot \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \tag{3.12}$$

$$\textbf{Clothoid:} \quad \begin{bmatrix} x_0 \\ c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{2}x_1^2 & \frac{1}{6}x_1^3 \\ 1 & \frac{1}{2}x_2^2 & \frac{1}{6}x_2^3 \\ 1 & \frac{1}{2}x_3^2 & \frac{1}{6}x_3^3 \end{bmatrix}^{-1} \cdot \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

The number of iterations is decided via 2.12, 2.13 and depends on three variables. The probability of success p is set to 99 %. The minimum number of points S to fit the model is 2 or 3 for the line and parabola or clothoid. Determining the percentage of inliers ω is done through experimentation. The

average fitting error (as in 3.9) is displayed on a collor map in the figure 3.12. Since RANSAC is random, the error will be different each time. Therefore, there is a sample of 50 iterations for each outlier value (inliers (ω) = 1 - outliers). The average time it took for one run of the algorithm is displayed on the other axis. The test was performed at a constant threshold value on a single whole sequence.

As the number of inliers decreases (more outliers), the denominator term in 2.12 decreases, increasing the number of iterations. The average time it takes seems to increase exponentially, so a trade-off between the time and the error needs to be made. In this example with a first-degree polynomial fitting on a curved road, the 'sweet spot' seems to be at 70 % of outliers or 30 % of inliers and an average time of about 0.1 seconds. After that, the time increase seems to bring only diminishing returns in error.

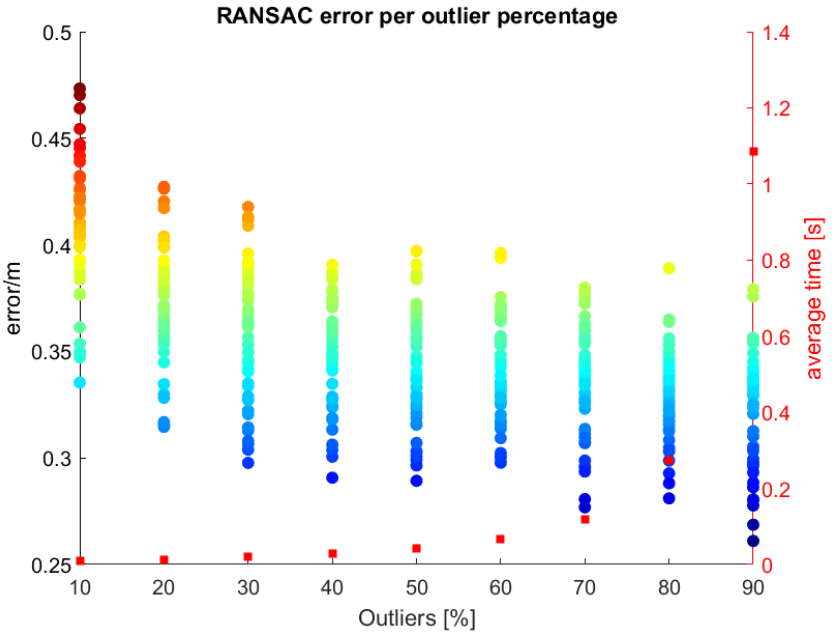


Figure 3.12: Example to showcase fitting error spread of 50 iterations with increasing percentage of outliers and an average time for one RANSAC cycle (red squares), at a constant threshold value.

The same principle applies to the threshold value. In this scenario and in any curve fitting in general, the threshold represents the Euclidian distance between the found polynomial and all of the points to be fitted. Any points below this value is regarded as an inlier, and the rest are outliers. Although this value does not influence the number of iterations, it has a direct correlation with the fitting error. This is clearly visible in the figure 3.13, where the best performance is at 0.1. Variations in average time are negligible. The test was done in the same manner but with a constant outlier percentage this time.

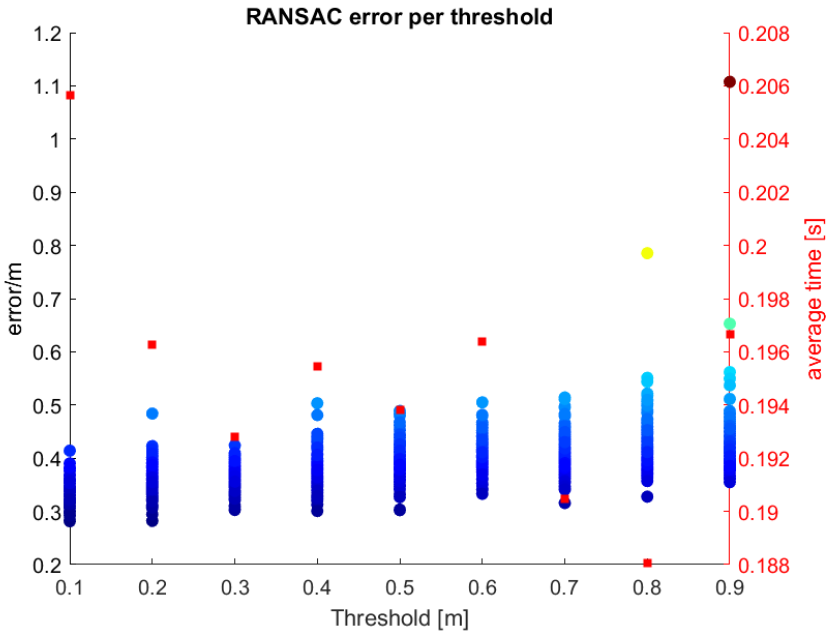


Figure 3.13: Example to showcase fitting error spread of 50 iterations with increasing threshold value and an average time for one RANSAC cycle (red squares), at a constant outlier percentage.

For the purpose of lane estimation, the RANSAC algorithm alone is very unreliable (see section 3.4). Even with the parallelisation step at the end, the randomness and higher computational requirements are detrimental. Furthermore, the LIDAR detections are the densest directly in front of the EGO vehicle, making the probability of choosing the random points from this part higher. This makes the RANSAC favor only the beginning of the lines (see figure 3.14). However, the ability to filter out unwanted points (outliers) is very valuable in some cases, which might prove useful in combination with the Kalman filter.

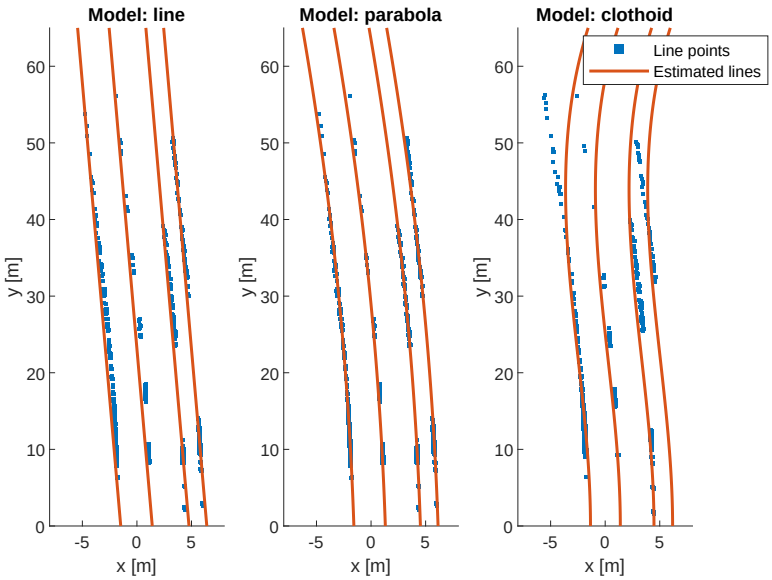


Figure 3.14: Example of the RANSAC algorithm with each of the three models.

3.3.5 RANSAC + Kalman Filter

Even though the Kalman filter is more robust for the purpose of lane estimation than the RANSAC, if a larger amount of outliers is present, it will include them in the estimation. The RANSAC can be used as a pre-filter to exclude any such unwanted data. For example, in case of entering a crossing (see figure 3.15, the KF (left) tries to include the lines from the other lines. The result after the paralellisation is not very precise. In contrast, the RANSAC (middle) performs much better in this scenario, as it completely ignores the other lines in the crossing.

The algorithm proposed combines the power of RANSAC and KF, taking the best out of both (*track_lines_RKF.m*). Before the measurement enters the KF loop, the current frame is passed through the RANSAC algorithm. Any points outside the RANSAC threshold are marked as outliers and discarded. The KF then receives only the filtered data as measurements. The resulting improved estimation can be seen in figure 3.15 (right).

3.3.6 Adaptive Kalman Filter

Road geometry is hardly ever constant, especially in an urban environment. Highways tend to be almost straight in a window of what a sensor can measure,

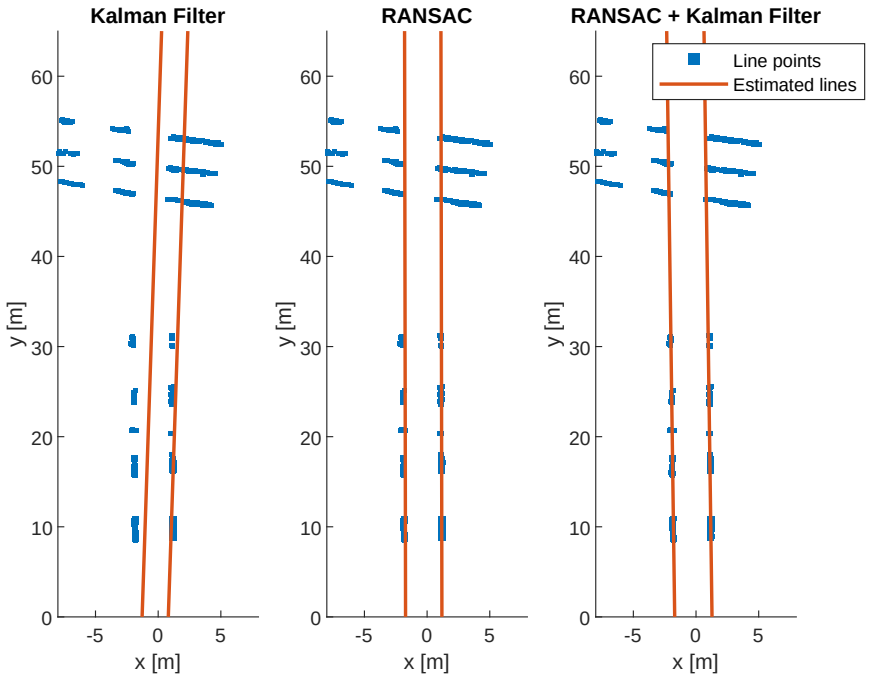


Figure 3.15: Example of the Kalman filter (left), RANSAC algorithm (middle), and their combination (right) in a high outlier count scenario.

but exits and other parts have a high curvature. Even though the clothoid or parabola model can be used to fit straight lines, a simple first-degree polynomial will be superior (figure 3.16). The ability to switch between several models depending on the shape of the road would be advantageous.

There are many methods on how to implement some sort of model switching. For example, the interactive multiple model (IMM) algorithm has shown to be a very robust way to detect a change and switch between different KFs. However, it is most suitable in situations with a high degree of maneuverability, such as target tracking, which the road environment is not [12]. There have been some studies proposing an adaptive Kalman filter solution, mostly relying on complex weighting factors to determine the correct model change [19].

The method proposed here is based on comparing the fitting error of several KF models running in the background (*track_lines_AKF.m*). Specifically, the lateral displacement error 3.9 of the three models with the weighting process used in the above methods. The model with the lowest error in any current frame is then chosen, penalized by the length of the line to favor the estimates based on the most measurement points. The final mode represents the best overall fitting model to all lines in each frame. Since the KF algorithm is very efficient, the computational speed is not very compromised even with 3 of them running in the background. An example of the mode switching on

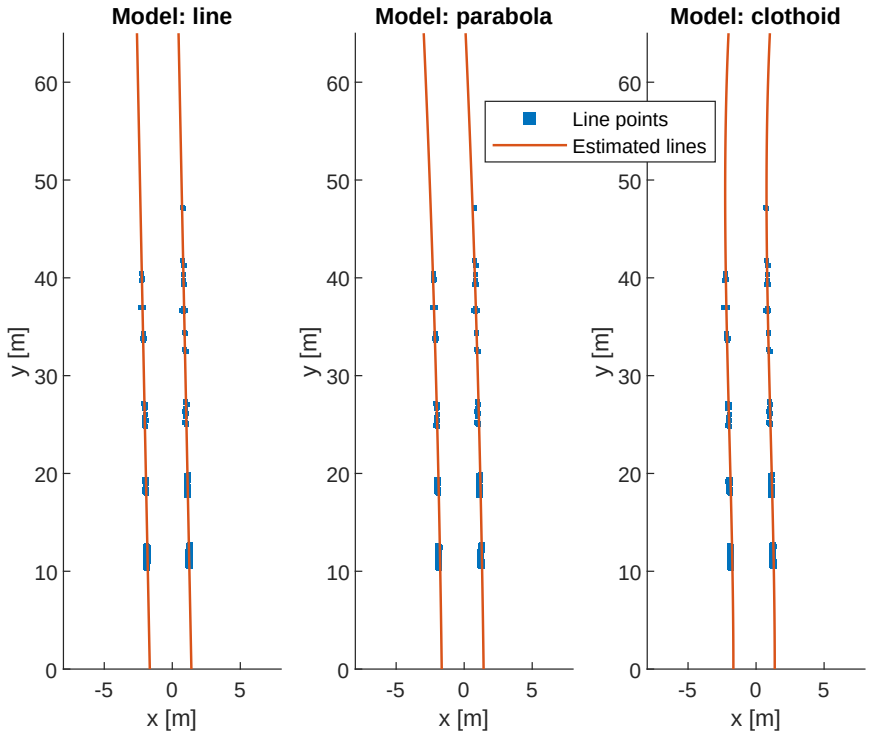


Figure 3.16: Difference between the three models on a straight road. Even with numerous detection points here, the performance of the line model better fits the data.

PandaSet sequence *013* (3 lines and a bike lane showcased in most examples above) can be seen in figure 3.17. The average error for each mode before the weighting process is on the right axis.

3.4 Results

Since this work aims to compare different lane estimation methods, this section will summarize the results of all the algorithms implemented and described above, focusing on their performance next to each other. To do so, two sequences from the PandaSet were chosen, sequence 013 and sequence 043.

Both sequences are from an urban environment, presenting an example of typical roads with imperfect line markings and traffic. Sequence 013 has a significant curvature of the road, heavy traffic obscuring the lines, and also a separate lane for bikes (see 3.9, 3.18). On the contrary, the sequence 043 is straight with only two dashed lines, but ends on a crossing where a lot of

Figure 3.17: Example of how modes switch in a span of one sequence. An average error of all lines is shown on the right axis.

outliers is present (see 3.15, 3.19).

Figure 3.18: One frame of PandaSet sequence 013.

More sequences were tested during development, but only two were chosen, since the camera images need to be labeled manually. This is a tedious and time-consuming process, but necessary to evaluate the results. Although there are some algorithms to automate the process, they are either not very accessible or not suitable for line markings.

The labeling was done in the MATLAB *Image Labeler* app, trying to choose

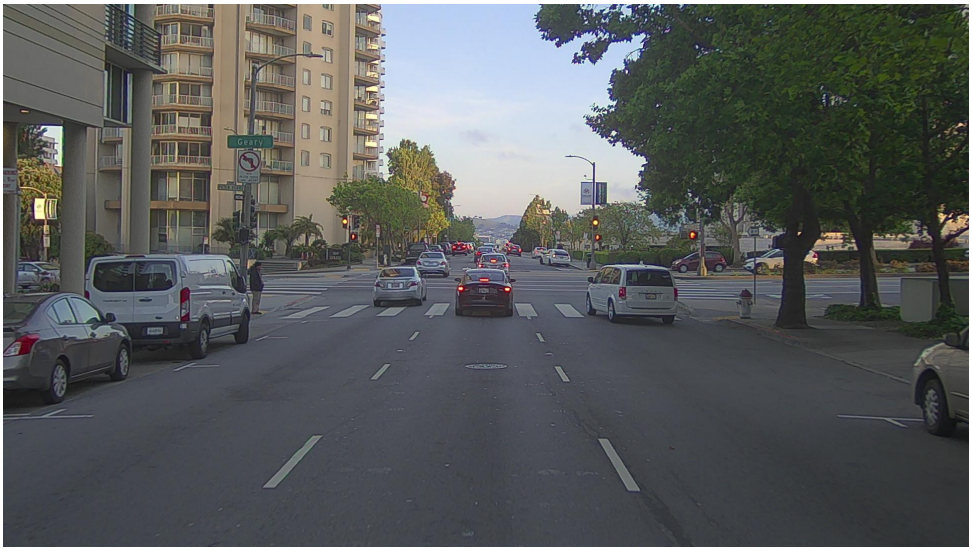


Figure 3.19: One frame of PandaSet sequence 043.

pixels that certainly belong to a line and are closest to the center of each line. This labeling is not perfect, but since all methods are tested against it equally, it does not pose a huge problem. The resulting PNG images with labeled pixels can be overlayed on top of the camera images or, more importantly, used to measure the error of the estimated lines (*extract_label_points.m*).

3.4.1 Error Estimation

The error of the final line estimates needs to be evaluated against a third variable not present in the estimate calculation itself. The only option in PandaSet is the camera images, which are not labeled. Moreover, the images come in JPG format with no depth information. This means that the error between the estimated line and the labeled line is measured in the image itself.

To do so, the estimated lines need to be projected into the camera image. The estimates in the EGO coordinates are projected back into the world frame, then to the camera frame, and finally into the image. Since the estimate gives out a polynomial that is evaluated into 2D points $[x, y]$, the third coordinate needs to be estimated as well. Using the MATLAB function *pcfitplane()*, a plane can be fitted through the lines of the processed LIDAR point cloud. From the plane equation and the evaluated polynomial, the z-coordinate can

be obtained.

$$\begin{aligned} ax + by + cz &= 0 \\ z &= \frac{-ax - by - d}{c} \end{aligned} \tag{3.13}$$

The transformation from world to EGO frame in 3.2 can be then done in reverse. In more practical terms, taking the rotation matrix R_w^e and the translation vector t_w^e from the world-to-EGO inverse transformation matrix T, where

$$P_e = R_w^e P_w + t_w^e \tag{3.14}$$

can be inverted to received points in the world frame.

$$P_w = (R_w^e)^T (P_e - t_w^e) \tag{3.15}$$

From the camera position and heading, the transformation matrix can be constructed as in 3.2. Points in the world frame can now be transformed into the 3D camera frame.

$$P_c = R_w^c P_w + t_w^c \tag{3.16}$$

To transform the 3D camera points into a 2D image, the camera intrinsics matrix is constructed.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

representing the internal parameters of the camera, focal length f and optical center c. The final step in projecting the 3D points onto the image plane is as follows.

$$\begin{aligned} P_c' &= K P_c \\ P_i &= \begin{bmatrix} x_c' \\ y_c' \end{bmatrix} \frac{1}{z_c'} \end{aligned} \tag{3.18}$$

where $P_c = [x_c \ y_c \ z_c]$ are the 3D camera points, $P_c' = [x_c' \ y_c' \ z_c']$, and $P_i = [x_i \ y_i]$ are pixels in the image frame.

The resulting projection can be seen in figure 3.20. It is apparent that there is a mismatch between the LIDAR points and the lines in the image. This could be due to wrong camera/LIDAR calibration, an error introduced during the transformations, or perhaps a skew factor that is usually part of the camera intrinsic matrix, which is not available in the PandaSet. However, this does not play a large role, since the main goal is to compare the methods and not to test them in the real world.

The final error can now be calculated as a reprojection error of each pixel of the labeled line to the nearest pixel of the estimated line. If done the other way, the result would favor the closer pixels from the labeling, which is prone

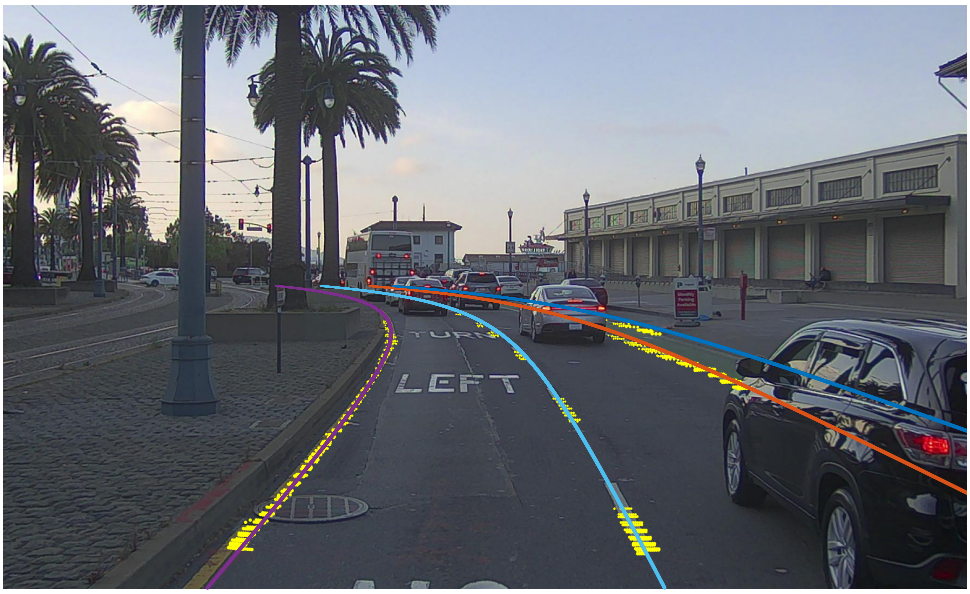


Figure 3.20: Projection of the estimated lines and LIDAR detections (yellow dots) onto the image.

to bias due to the manual labeling. The reprojection error in general is the average L2 norm of point correspondence errors expressed as

$$e_r = \frac{1}{N}\sum_{i=1}^{N} |p_i - q_i|_2 \tag{3.19}$$

where p_i are the observed feature points on the image plane and q_i are the predicted image plane locations of the 3D feature points when projected onto the image plane and distorted using the lens model parameters from the camera intrinsics. This is essentially what was done during the transformation above in the case of this work.

This would be enough to give some value to the error between the estimates, but the error units are pixels, which does not really tell anything meaningful. The reprojection error can be converted to a length error, if the working distance is known. However, the camera data do not feature depth information. Nevertheless, there is a workaround which introduces some bias, since it uses the depth from the estimates themselves. It will be shown that this introduced error is almost negligible, and the resulting error would have an actual value.

Using the pixel width of the image w_{px} and the horizontal focal length f_x, the field of view FOV can be calculated as follows.

$$FOV = 2\arctan\frac{w_{px}}{2f_x} \tag{3.20}$$

The approximate pixel angle α_p can then be found as

$$\alpha_p \approx \frac{FOV}{w_{px}} \tag{3.21}$$

If done separately for each pair of line label pixel and closest estimated pixel, it can be used to calculate an arc error e_a, which, when multiplied by the distance of each estimated pixel before it was transformed from 3D, will give the length error e_l.

$$\begin{aligned} e_a &= \alpha_p \cdot e_r \\ e_l &= d \cdot e_a \end{aligned} \tag{3.22}$$

Averaging the error per all pixels of a labeled line will essentially give an average lateral displacement error between the labeled lines and the estimated lines. With a standard deviation this gives a tangible value to the results.

To prove that the bias introduced by using the distance from the estimates is very low, a reprojection error of 8 pixels at 10 meters will be assumed. The length error with the numbers from the camera parameters is

$$\begin{aligned} FOV &= 2 \arctan \frac{1920}{2 \cdot 1970} = 0.91\ rad \\ \alpha_p &= \frac{0.91}{1920} = 0.47\ mrad \\ e_a &= 0.47 \cdot 8 = 3.8\ mrad \\ e_l &= 3.8\ mrad \cdot 10\ m = 3.8\ cm \end{aligned} \tag{3.23}$$

It is clear that even if the error from the distance introduction was, for example, 0.5 meters, the result would change about only 2 mm. This is very acceptable, considering that such a high error is very unlikely, and probably one order higher than it would be in the reality.

3.4.2 Error Results

The results of each sequence can be seen in figures 3.21 and 3.22. All the methods described in the chapters above are plotted in juxtaposition to see how they compare. The Kalman filters with respective models are in the bottom group of bars, the RANSAC is on top. The best-performing RANSAC model combined with the KF is displayed in the RANSAC group.

The numbers on top of the bars correspond to the error value for better clarity, with the standard deviation as the black line (error ± std). Each bar summarizes the average error between the estimated line and the labeled line of all lines of the entire sequence. In case of the sequence 013, that means the overall estimate error of 4 lines in 80 frames.

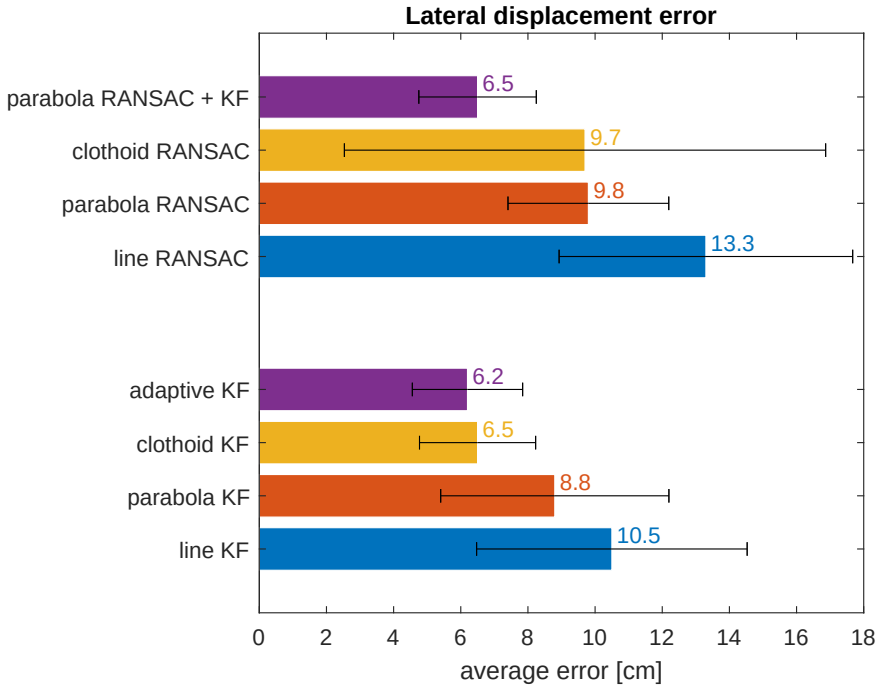


Figure 3.21: Lateral displacement error comparison between the individual methods tested on PandaSet sequence 013. Colored numbers correspond to the value of the bars. Black lines represent the standard deviation.

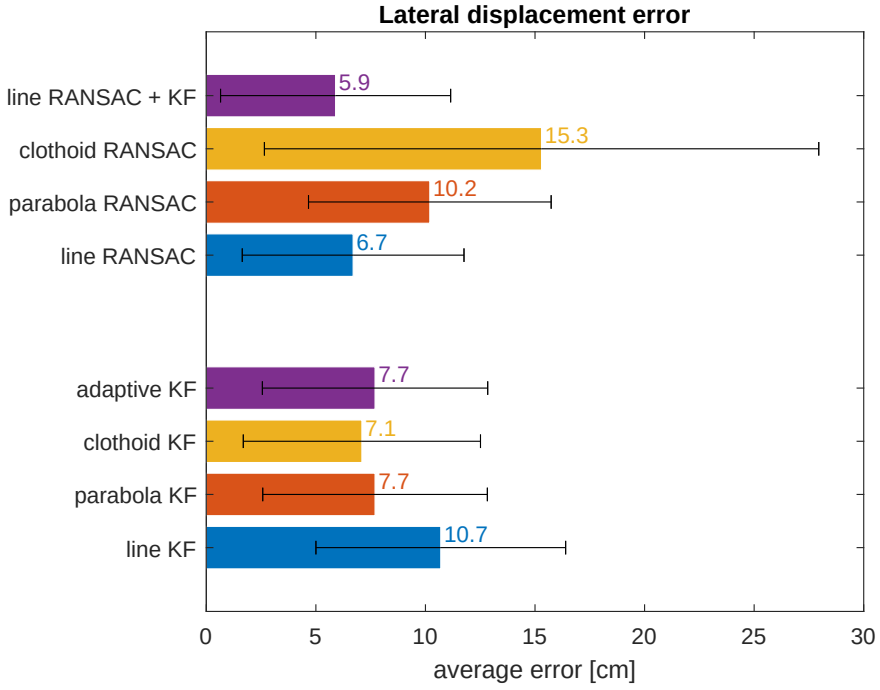


Figure 3.22: Lateral displacement error comparison between the individual methods tested on PandaSet sequence 043. Colored numbers correspond to the value of the bars. Black lines represent the standard deviation.

3.5 Discussion

The results of sequence 013 in figure 3.21 clearly shows the superiority of the Kalman filter over the RANSAC algorithm. Especially the clothoid model KF which is expected since this sequence has a quite high curvature. In comparison, the adaptive KF (AKF) performed marginally better, with improvement in error and standard deviation, (6.5 ± 1.7) cm and (6.2 ± 1.6) cm, respectively. Since the clothoid model is ahead of the others by far, the AKF was inclined to choose it more often, hence the rather small improvement. The PandaSet unfortunately does not feature any longer sequences, and since they do not overlap, they cannot be stitched together. Testing the AKF on a longer sequence with more abrupt changes in the curvature would be preferable, and might showcase its advantage over the other models even more.

The RANSAC alone performs much worse on this sequence than the KF. Due to the curvature, the line model is the worst overall, and the clothoid model offers almost no improvement over the parabola model. The high standard deviation and computation time seem to result from the more complex polynomial structure. Combining the clothoid KF with the better polynomial RANSAC as a prefilter does not result in any improvement over the KF alone. This is due because there are really no outliers in this sequence, and hence the measurement goes into the estimator unchanged.

The sequence 043 features straight lanes with dashed lines and has a much lower number of LIDAR detections than the other tested sequence. There is also a high change in the slope of the road. Moreover, there is a crossing in the end, which features a lot of outliers. These attributes together have some interesting consequences.

There are not many line markings detected by the LIDAR in about the first half of the sequence (compare figure 3.16 and 3.14). Most of them and around 30 m ahead of the EGO vehicle and the estimated lined are evaluated in the whole ROI region up to 65 m. This means that half of the line is being predicted. The slope of the road also changes, and since the estimation is done in the 2D plane, the estimates tend to be skewed to the center at the ends. This makes the clothoid and parabola models better, as they are able to curve at the tips, better predicting the line markings (see figure 3.22). The statements above also explained the high standard deviation, since the estimate is much more precise in front of the EGO vehicle.

The worse performance of the AKF can be explained by the fact that it

evaluates the score of how the model fits the data on the measurement. Since the measurements are not that dense in this case, and it cannot see how the curved models can predict the actual lines better, it can choose a worse performing model in the end. This approach could be extended by some more complex weighting factors or by a moving average of the switching criteria instead of checking for the switch in every frame.

The higher order RANSAC models are bad at estimating this straight sequence, especially the clothoid model with a very high standard deviation. On the contrary, the line model outperforms even the KF estimates, as the ability to discard the outliers really shines here. This is further demonstrated when the prefiltered data is used with the KF, resulting in the best overall error (5.9 ± 5.2) cm (line RANSAC + KF in figure 3.22).

The results show that the Kalman filter with the underlying clothoid model is capable of estimating the line markings in various scenarios. The adaptive Kalman filter can further improve the estimates, and the RANSAC brings high value in cases with a lot of noise.

The absence of any camera depth information in the PandaSet results in an inconvenient workaround in order to evaluate the estimated lines. However, it is unclear whether data from a stereo camera or similar sensor would drastically improve the precision, at least in the context of this work. The projection of the LIDAR data from the world frame onto the image plane according to the PandaSet tutorial shows a clear error between the two sensors (3.20). This contributes to the resulting error, and it is not certain where this inconsistency comes from. However, since the aim of this work is to compare several methods against the same data, the actual error value does not play that crucial role.

More data and testing are needed to further evaluate performance of these algorithms. However, the advantages of certain models are obvious. Relying on the labeled data limits the number of usable measurements, but testing the performance by other means is challenging. Having larger and easily accessible labeled datasets would be very beneficial not only for better evaluation of one's results, but also to be able to compare to methods developed by others. Further testing is needed to determine how the approach with plain KF would stack against some more complicated methods, such as learning-based algorithms. Especially, comparing whether the low computational benefits satisfy the performance of the KF.

As ADAS become more and more common in modern cars and advanced autopilots are starting to emerge, the broader the knowledge of various

methods applicable, the better. The results of this work and the prepared framework with an evaluation process can help expand this knowledge and can be used in the development of lane-keeping and other driving assistants.

Chapter 4

Integration of Camera and LIDAR Data for Enhanced Line Estimation

Line estimation is a critical aspect of autonomous navigation, providing crucial information on road geometry and facilitating safe vehicle control. Traditionally, vision-based methods relying solely on camera data have been employed for line estimation. However, challenges arise when faced with adverse conditions, such as varying lighting, occlusions, and the presence of objects in the camera view [20]. To overcome these limitations, the integration of LIDAR data into the line estimation process could improve accuracy and reliability.

The fusion of camera and LIDAR data leverages the strengths of each sensor modality, addressing their individual limitations discussed in previous chapters. While cameras excel at providing high-resolution images for semantic understanding, LIDAR sensors offer precise distance measurements, especially in complex three-dimensional environments. When these data sources are combined, a more comprehensive and accurate representation of the surroundings can be achieved, particularly in challenging scenarios where one sensor may struggle.

There are not many publications on this topic. In one of the newest studies, Narote et al. [18] mentions only one notable work, where the actual fusion is done and where the LIDAR lanes are detected using the cameras. They propose a new approach that involves a novel algorithm that fuses depth data from a 2D LIDAR and image data from a camera to remove noise from objects in the view and reliably detect lanes. The algorithm first identifies objects using 2D LIDAR, and a modified Bird's Eye View (BEV) image is

generated by converting object pixels to background pixels. Subsequently, line detection is performed on the modified BEV image, resulting in enhanced accuracy and robustness against objects in the camera view. No actual lane tracking and estimation is involved, but they plan to do so using a Kalman filter.

In this work, the results of line estimation based on the LIDAR data were compared against labeled line markings in the camera images. It was shown that there could be a significant difference between these two data sources. Even with a frequent calibration, there will always be some error, and mitigating this by any means should be a high priority for safety reasons. Robust algorithms, such as the Kamlan filter, exists today that can utilize measurements from more sensors and make an estimate with their noise taken into account.

There are many techniques for sensor fusion that have already been used in other fields that could be applicable to this problem. Testing several approaches, for example, synchronous and asynchronous fusion, estimating the states from each sensor separately or using one filter for everything, and so on, presents a very interesting topic for further investigation.

It is not within the scope of this work to test the algorithm performance in a real driving scenario. However, the integration of both LIDAR and the camera would most likely result in better line marking detection, further improving the estimation and tracking. More research is needed to prove the benefits of the LIDAR-camera fusion, but as the leading car manufacturers with advanced autopilots (such as Tesla) strive away from using other sensors than cameras, the motivation behind fades away. There is an ongoing development of LIDAR technology that should be better suited for cars, but commercial application is still in the future. Currently, LIDAR is still an expensive technology with many unknowns but with the potential to increase the overall safety and efficiency of autonomous driving technologies.

Chapter 5

Conclusion

As Advanced Driver Assistance Systems (ADAS) become an integral part of every modern car, the demand for even more autonomous functions increases. Parking assistants, lane departure warning, or collision avoidance is practically a given part of any new vehicle equipment, and the leading automotive software companies work intensely towards the first degree of full self-driving. Developing and comparing robust and safe methods for these systems was never so crucial. This work focused on one problem of this field, lane tracking and estimation.

The aim of this work is to select common algorithms and models in the literature and compare them with each other. Furthermore, testing them on a publicly available PandaSet database with LIDAR and camera measurements. To do so, a whole framework for data processing and evaluation was developed.

The Kalman filter (KF) and the RANSAC algorithm were chosen as the main instrument for the lane estimation. Each was tested with three underlying models representing the curvature of the line markings, namely a first-degree polynomial, a second-degree polynomial, and a clothoid, which is essentially a modified third-degree polynomial. Furthermore, an adaptive KF and a combination of RANSAC plus KF were proposed and compared.

The results showed that the KF with the clothoid model performed better than any other model or the RANSAC alone. The adaptive KF approach offered only minor advantages and both showed drawbacks in the presence of outliers. In this situation, the RANSAC algorithm was able to discard

them and in combination with the KF showed the best results. However, in no-outlier scenario it performed poorly and offered no improvement when used with the KF.

More testing of the presented methods is needed, as there is a lack of data to evaluate the results. However, the whole framework is prepared, which is capable of processing the LIDAR point clouds and evaluating the results against labeled camera images. This represents an important contribution, as most of the research is done on custom datasets.

Continuing this work, more estimation techniques could be compared. There are other types of KF with different combinations of models, or entirely different approaches. For example, the least-squares fitting, other RANSAC and polynomial fitting, and many more types of estimators. An interesting approach would be to expand the KF algorithm presented in this work by estimating the polynomial parameters individually with a constant-velocity and constant-acceleration model.

Appendix A

Bibliography

[1] M. Aly. Real time Detection of Lane Markers in Urban Streets. *In Proceedings of the 2008 IEEE Intelligent Vehicles Symposium*, page 7–12, June 2008.

[2] A. B. Hillel; R. Lerner; D. Levi; G. Raz. Recent progress in road and lane detection: A survey. 25:727–745, March 2014.

[3] V. K. Kukkala; J. Tunnell; S. Pasricha; T. Bradley. Advanced Driver-Assistance Systems: A Path toward Autonomous Vehicles. *In IEEE Consumer Electronics Magazine*, 7:18–25, 2018.

[4] S. Yenkanchi. Multi Sensor Data Fusion for Autonomous Vehicles. *University of Windsor*, 2016.

[5] Tesla Motors Inc. Tesla vision update: Replacing ultrasonic sensors with tesla vision. `https://www.tesla.com/en_eu/support/transitioning-tesla-vision`.

[6] Elon Musk. X. `https://twitter.com/elonmusk/status/1447588987317547014`.

[7] Hesai and Scale. PandaSet. `https://pandaset.org`.

[8] S. Waykole; N. Shiwakoti; P. Stasinopoulos. Review on lane detection and tracking algorithms of advanced driver assistance system. *Sustainability*, 13, 2021.

[9] C. Y. Kuom; Y. R. Lu; S. M. Yang. On the Image Sensor Processing for Lane Detection and Control in Vehicle Lane Keeping Systems. *Sensors*, 19, 2019.

[10] K. Geng; G. Dong; G. Yin; J. Hu. Deep dual-modal traffic objects instance segmentation method using camera and lidar data for autonomous driving. *Remote Sens*, 12:1–22, 2020.

[11] W. Wei; B. Shirinzadeh; R. Nowell; M. A. Ghafarian; T. Shen. Enhancing solid-state lidar mapping with a 2D spinning lidar in urban scenario slam on ground vehicles. *Sensors*, 21:1–18, 2021.

[12] Y. Bar-Shalom; X. R. Li; T. Kirubarajan. Estimation with Applications to Tracking and Navigation. *Wiley Interscience*, pages 207, 387, 476, 2001.

[13] Peter. J. Huber. Robust Statistics. *Wiley*, page 1, 1981 (republished in paperback, 2004).

[14] A. Hast; J. Nysjö; A Marchetti. Optimal RANSAC – Towards a Repeatable Algorithm for Finding the Optimal Set. *Journal of WSCG*, 21:21–30, 2013.

[15] Z. Lu; Y. Xu; X. Shan. A lane detection method based on the ridge detector and regional G-RANSAC. *Sensors*, 19, 2019.

[16] M. Thuy; F. P. León. Lane Detection and Tracking Based on Lidar Data, journal = Metrology and Measurement Systems, year = 2010, volume = 17, pages = 311-322.

[17] Y. Zhang; J. Wang; X. Wang; C. Li; L. Wang. A real-time curb detection and tracking method for UGVs by using a 3D-LIDAR sensor. *2015 IEEE Conference on Control Applications (CCA)*, pages 1020–1025, 2015.

[18] F. Ghallabi; F. Nashashibi; G. El-Haj-Shhade; M. A. Mittet. Lidar-based lane marking detection for vehicle positioning in an hd map. *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2209–2214, 2018.

[19] D. Khosla. Adaptive kalman filter approach for road geometry estimation. *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems*, 2:1145–1151, 2003.

[20] S. P. Narote; P. N. Bhujbal; A. S. Narote; D. M. Dhane. A review of recent advances in lane detection and departure warning system. *Pattern Recognition*, 73:216–234, 2018.

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Veškrna Daniel** Personal ID number: **483632**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

II. Master’s thesis details

Master’s thesis title in English:

LIDAR-Based Lane Tracking using Kalman Filtering and its Fusion with Camera-Based Lane Data

Master’s thesis title in Czech:

Sledování jízních pruhů na bázi LIDAR dat pomocí Kalmova filtrování a jeho fúze s kamera daty

Guidelines:

1) Problem introduction and state-of-art overview
2) Implementation of lane points fitting using RANSAC algorithm
3) Lane fitting of generated data and custom or publicly available LIDAR data
4) Implementaition of Extended Kalman Filter (EKF) for lane tracking
5) Other means of tracking and comparison to EKF
6) Proposal and implementation of fusion of lane data from different sensors (i.e. LIDAR + camera)

Bibliography / sources:

[1] Bar-Shalom, Y., Li, R. X., & Kirubarajan, T. Estimation with Applications to Tracking and Navigation (1st ed.). Wiley-Interscience 2001
[2] Thuy, M., & León, F. Lane Detection and Tracking Based on Lidar Data. Metrology and Measurement Systems, 17(3) 2010
[3] Ghallabi, F., Nashashibi, F. LIDAR-Based Lane Marking Detection For Vehicle Positioning in an HD Map. 21st International ITSC 2018
[4] Waykole, S., Shiwakoti, N. Review on Lane Detection and Tracking Algorithms of Advanced Driver Assistance System. Sustainability 2021
[5] Paek, D. H., Kong, S. H., & Wijaya, K. T. K-Lane: Lidar Lane Dataset and Benchmark for Urban Roads and Highways. 2022

Name and workplace of master’s thesis supervisor:

Kundak Nuri, MSc. Porsche Engineering Services, s.r.o., Radlická 714/113a 158 00 Praha 5

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **20.01.2023** Deadline for master's thesis submission: **09.01.2024**

Assignment valid until:
by the end of winter semester 2024/2025

Kundak Nuri, MSc.	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature