Lane Detection For Autonomous Car's High Precision Localization System

Pham Tuan Viet
Nguyen Dac Hoang Long
Pham Tien Hieu

Supervisor: Assoc. Prof. Phan Duy Hung

Bachelor of Artificial Intelligence
Hoa Lac campus - FPT University
2024

Acknowledgment

We gratefully acknowledge the support of Phenikaa-X Joint Stock Company for their funding, which facilitated the creation of an environment conducive to this research.

In this journey of academic pursuit, we owe a debt of gratitude to Assoc. Prof. Phan Duy Hung, whose unwavering dedication and enthusiastic guidance have been the cornerstone of our progress over the past four months. His invaluable insights, patience, and encouragement have played an integral role in shaping our ideas and refining our thesis. We are truly fortunate to have had such a committed mentor by our side.

We extend our heartfelt thanks to the esteemed lecturers at FPT University, particularly those within the Information Technology Specialization Department. Their expertise, support, and commitment to excellence have provided us with a solid foundation and the necessary skills to navigate the complexities of our research. Their dedication to nurturing intellectual curiosity has been a source of inspiration throughout this journey.

Furthermore, we express our deep appreciation to our families and friends, whose unwavering encouragement and unwavering belief in our abilities have been a constant source of strength. Their support, understanding, and valuable insights have enriched our academic endeavors and motivated us to strive for excellence. We are profoundly grateful for their enduring love and encouragement.

Together, these individuals have played an indispensable role in our academic journey, and for that, we are eternally grateful. Their support has been instrumental in the completion of this thesis, and we humbly acknowledge their contributions with heartfelt gratitude.

Abstract

Localization is a critical component for the safe and efficient operation of autonomous vehicles, as it enables precise positioning within the environment. In this paper, we present a novel approach to real-time visual-based localization for autonomous vehicles that combines the strengths of deep learning-based lane detection for robust perception with map matching algorithms for accurate localization against a priori map data. Through extensive experimentation and evaluation, our system achieves a remarkable mean Euclidean error of approximately 0.5 meters, demonstrating high precision in localization tasks. Moreover, our approach operates at an impressive frame rate of approximately 35 frames per second (fps), ensuring timely and responsive localization updates crucial for real-time decision-making in autonomous driving scenarios to out-perform previous works on visual-based localization systems. These results underscore the efficacy and feasibility of our method in addressing the visual-based localization challenges in autonomous vehicle navigation, paving the way for enhanced safety and reliability in autonomous driving systems.

Keywords: Lane Detection, Autonomous, Localization, Deep learning

Table of contents

List of tables

List of figures

List of abbreviations and acronyms

FPS:	Frame Per Second
NDT:	Normal Distributions Transform
ICP:	Iterative Closest Point
EKF:	Extended Kalman Filter
GNSS:	Global Navigation Satellite System
IMU:	Inertial Measurement Unit
LIDAR:	Light Detection and Ranging
CNN:	Convolutional Neural Networks
YOLO:	You Only Look Once Network
FPN:	Feature Pyramid Networks
IoU:	Intersection over Union
MSE:	Mean Square Error
ROI:	Region Of Interest
LaneATT	Lane Attention Detection Networks
PolyLaneNet	Polynomial Lane Detection Networks
MCL	Monte Carlo Localization

1. Introduction

1.1. Motivation

Localization plays a pivotal role in enabling the operational functionality of autonomous vehicles, serving as the cornerstone for safe navigation and decision-making in dynamic environments. Traditionally, localization systems rely on the fusion of data from a diverse array of sensors, including Global Navigation Satellite Systems (GNSS), Inertial Measurement Units (IMU), LIDAR (Light Detection and Ranging), Radar, and Cameras. While LIDAR offers high-precision localization, its adoption can be limited by cost constraints and slower scanning frequencies, prompting researchers to explore alternative camera-based localization methods.

Within camera-based localization, two prominent methodologies emerge: Visual Odometry and Map Matching. Each approach presents unique advantages and challenges, shaping the landscape of research and development in autonomous vehicle localization.

Visual Odometry methods [1, 2, 3, 4, 5], hold the advantage of versatility, capable of functioning across a wide range of environments. However, they are susceptible to drift over prolonged operation and often face challenges in achieving real-time processing within autonomous systems.

Conversely, Map Matching methods have gained prominence for their reliance on traditional lane detection algorithms followed by aligning detected lane lines with a predefined map. While these methods offer a structured approach, they struggle to effectively mitigate noise from real-world environments and the inherent vibrations present in camera systems.

In response to these challenges, this proposal introduces a novel mono-camera-based localization method. Central to this approach is the integration of a Deep Learning Lane Detection model, followed by map matching techniques. By harnessing the capabilities of deep learning, this method aims to overcome the limitations of traditional lane detection algorithms, offering improved robustness and accuracy in localization tasks.

Through the fusion of deep learning-based lane detection with map matching, this proposed method endeavors to provide a more reliable and precise localization solution for autonomous vehicles. By leveraging the strengths of both approaches, it seeks to navigate the complex landscape of real-world environments with enhanced efficacy, paving the way for advancements in autonomous vehicle localization research and practical implementation.

1.2. Related work

1.2.1. Lane Detection

Lane detection is a crucial aspect of computer vision and autonomous vehicle technology. It involves identifying and tracking the lanes on a road to assist in various applications such as lane-keeping assistance, lane departure warning, and autonomous driving. The primary goal of lane detection is to locate and delineate the lanes, typically represented as painted markings on the road surface. This task is essential for enabling vehicles to understand their position within the road and make informed decisions based on the detected lane boundaries. There are 3 main approaches to perform this task such as Segmentation-based, Anchor-based and Parameter-based. Each method has its own advantages and drawbacks.

With Segmentation-based methods, [6, 7, 8] is easy to implement and achieve pixel-wise accuracy but because of this, this method is not robust enough in real-life applications. Furthermore, this method doesn't understand the relationship between pixels of the same line and needs lots of post processing to get a smooth and identical lane line. One of the most famous segmentation-based methods is LaneNet [6]. This method treats lane detection as an instance segmentation problem, consisting of a lane segmentation branch and a lane embedding branch that can be trained end-to-end. The segmentation branch has two output classes which are background or lane. The lane embedding branch further disentangles the segmented lane pixels into different lane instances. Furthermore, combines the benefits of binary lane segmentation with a clustering loss function designed for one-shot instance segmentation.

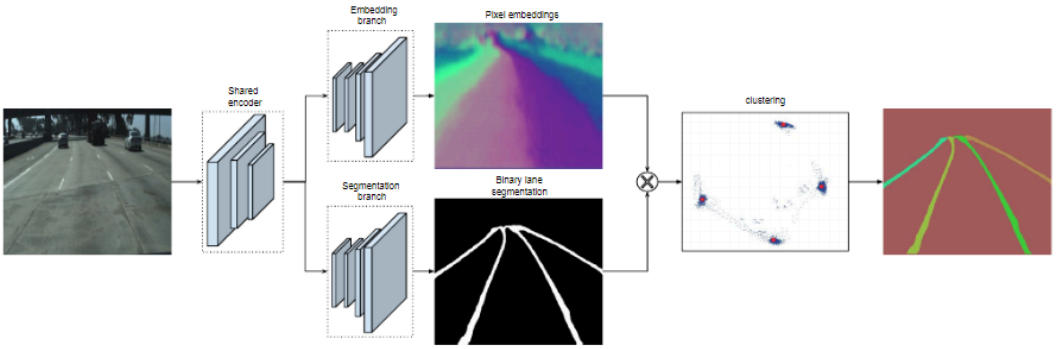

Figure 1: LaneNet architecture

Unlike the segmentation-based methods, Parameter-based methods [9, 10] don’t need much post processing and have a significantly low computational cost. But most of them need a predefined number of lanes and choose which polynomial order represents a line. And because data is usually a long-tail distribution, this method can be overfit with a straight line and can’t learn how curve lines represent. In PolyLaneNet [9], they use third order polynomial regression and each output

polynomial represents each lane marking in the image and corresponding confidence scores and vertical offset. They also predict the vertical position h of the horizon line, which helps to define the upper limit of the lane markings. But all of that can still be out-performed by Anchor-based methods.

Similar to famous object detection methods like YOLO, anchor-based method [11, 12] first predefined line anchor which was first introduced in LaneATT [11]. This line anchor contains (1) land and background probabilities. (2) the start points of the lane and the angle between the x-axis of the lane anchor (termed as x, y, theta). (3) The length of the lane. (4) The N offset points of that lane. This method depends hardly on the anchor generated but doesn't need much post processing and is more likely to get state of the art in 2D lane detection and can be used in real-time application. Above all of those methods, this work chooses CLRNet [12] as our baseline.

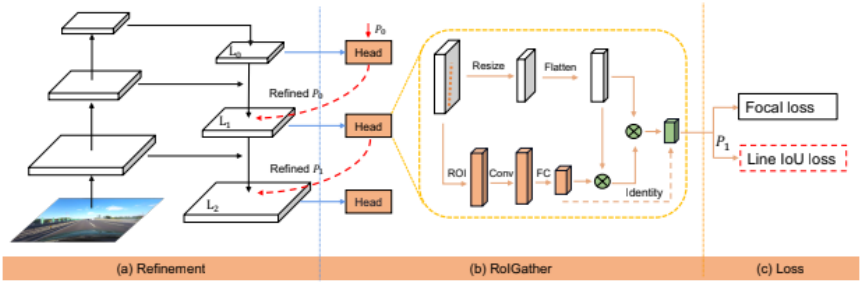

Figure 2: The architecture of CLRNet [12]

This method includes a CNN backbone then combines information from different features maps using FPN then each feature map is combined with its previous feature maps to ROIGather module which contains a cross attention between the prior anchor to the features map for global context of lane information. Then smaller features map will be used to further fine-tune larger features map. Finally, they propose Line IoU loss for lane detection, regressing the lane as the whole unit (Fig. 1)

Table 1. Benchmark of CLRNet and other models results in CULane Dataset

Model	**Evaluation Dataset**	**F1@50**	**GFLOPs**
LaneATT-ResNet18 [11]	CULane	75.13	9.3
LaneATT-ResNet34 [11]	CULane	76.68	18.0
LaneATT-ResNet122 [11]	CULane	77.02	70.5
CondLane-ResNet18 [13]	CULane	78.14	10.2
CondLane -ResNet34 [13]	CULane	78.74	19.6
CondLane -ResNet101 [13]	CULane	79.48	44.8
CLRNet-ResNet34 [12]	CULane	79.73	21.5
CLRNet-ResNet101 [12]	CULane	80.13	42.9
CLRNet-DLA34 [12]	**CULane**	**80.47**	**18.4**

CLRNet with DLA34 as backbone has proven its effectiveness over other models (shown in Table. 1). Thus, this work uses the CLRNet-DLA34 to detect lane lines for autonomous localization problems.

1.2.2. Visual-based Localization

Visual-based Localization is a method which uses camera image and other cheap sensors like GNSS and IMU to handle localization problems. This is already being used in self driving car localization, in David's method [14], First it generates map using OpenStreetMap data and apply ridge detector with camera image, then doing Iterative closest point (ICP) scan matching to corrected position with prebuilt map using OpenStreetMap data. But this method has several drawbacks such as it is not robust in real world environments and since this matching method doesn't take care about longitudinal data, as the result this will fail in high way scenarios.

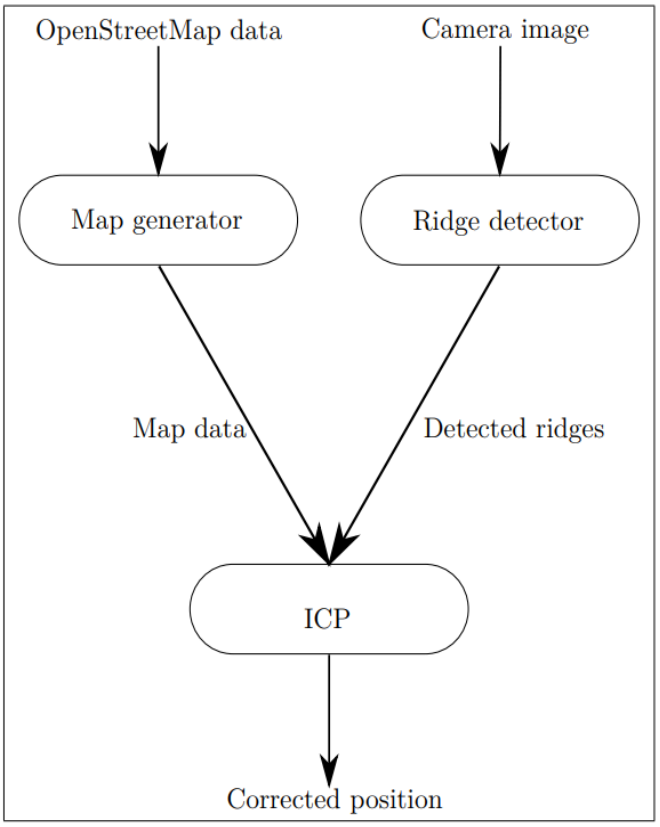

Figure 3: The three parts of the approach: map generation, ridge detection and comparison with ICP algorithm are articulated as shown by this figure.

After that, Sadli's method [15], using a similar method to first segment the lane using LaneNet [6] then estimate the ego's position relative to the median of the lane then correct it with map and GPS data. But once again, this method still has the same problem like previous work.

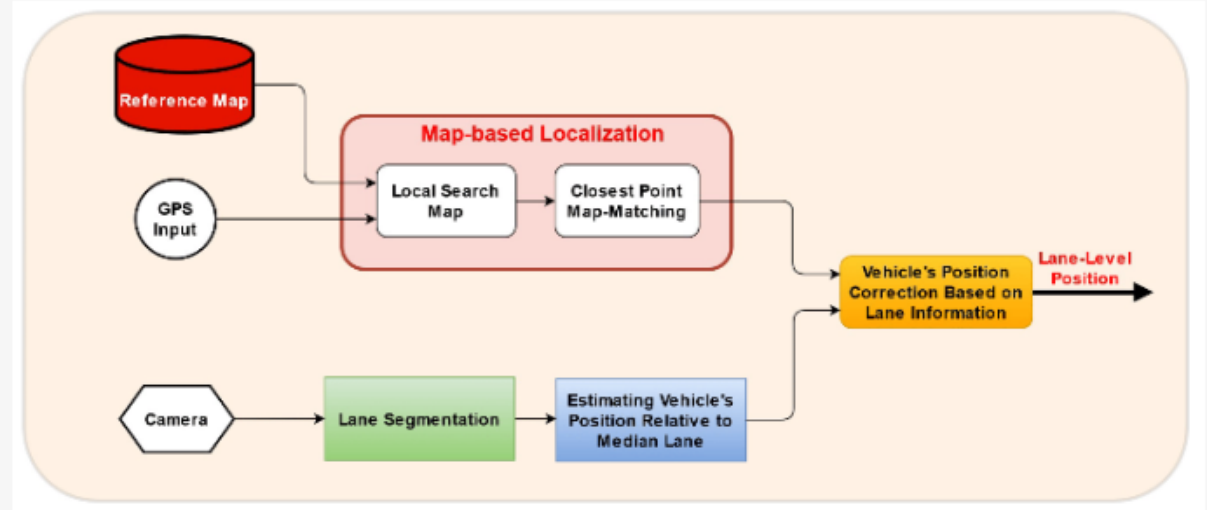

Figure 4: The pipeline of map-based localization

Recently, Yabloc [16] which is developed by Tier4 has been represented as an open source visual-based localization. Yabloc [16] uses particle filter or maybe known as monte-carlo localization to combine the matching between image and lanelet2, GPS, IMU to get the probability of the pose (Fig. 4). This method using a map built by lanelet2 and landmarks from the camera is a lane line. It is still using GPS to local search the relative position in lanelet2 map and then using particle filter to find the best particle fit by both GPS and camera measurements projected to a predefined map. Its basic principle is first using line segmentation detectors and graph segmentation from OpenCV. After getting the graph segmentation, it will find the most likely part is lane and filter all the line segmentation in that segment area as good line and other as bad line. Then transform these segments for each particle and determine the particle's weight by comparing them with the cost map generated from lanelet2.

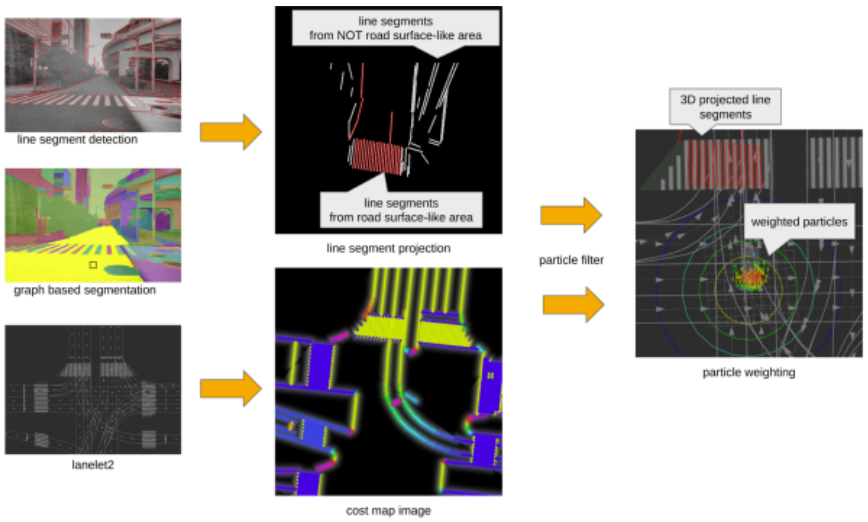

Figure 5: Yabloc [16] basic pipeline

After running some experiments of this method with Autoware and Awsim simulation, this lateral localization error is not a lane line error which makes the ego car hit the pavement. Due to the graph segmentation and search road marking can reduce wrong output which means this considering non road area is road area results as filter line segment worse. Without accurate road markings, this algorithm can’t match the cost map and the particle's weight is not distributed on a single lane anymore. In real world application, there are some other issues when the road markings are not present in real life or lanelet2 map, this will eventually fail. In this case, only GNSS is used which can be sure is centimeters level. And this method uses the opencv algorithm so it cannot be robust in real world environments. To handle this problem, we present a new method that innovates Yabloc [16] to get robust and low error visual-based localization.

1.2.3. Monte Carlo Localization (Particle Filters)

Monte Carlo Localization (MCL), often referred to as particle filters, serves as a robust method for enabling a robot to accurately determine its position within a known environment. At its core, MCL leverages a probabilistic framework that combines sensor measurements with the robot's motion model to estimate its location. Initially, a large number of particles are uniformly distributed across the map, effectively representing potential positions of the robot. As the robot moves, it continuously updates the positions of these particles based on its motion model, which predicts where the robot is likely to be next. Concurrently, sensor data is collected and used to evaluate the likelihood of each particle being the true position of the robot. This evaluation involves comparing the sensor readings to what the robot would expect to observe at each particle's location. Bayesian inference is then employed to assign weights to the particles, with those that are more consistent with the sensor measurements receiving higher weights. Conversely, particles that deviate significantly from the observed data are assigned lower weights. Through successive iterations of this process, the particle set undergoes resampling, where particles with higher weights are more likely to be replicated, while those with lower weights are pruned. This iterative refinement gradually narrows down the potential positions of the robot until convergence is achieved, providing a precise estimate of its location within the environment. MCL's versatility and effectiveness make it a cornerstone in robotics applications, facilitating tasks ranging from localization and navigation to mapping and exploration in dynamic and challenging environments.

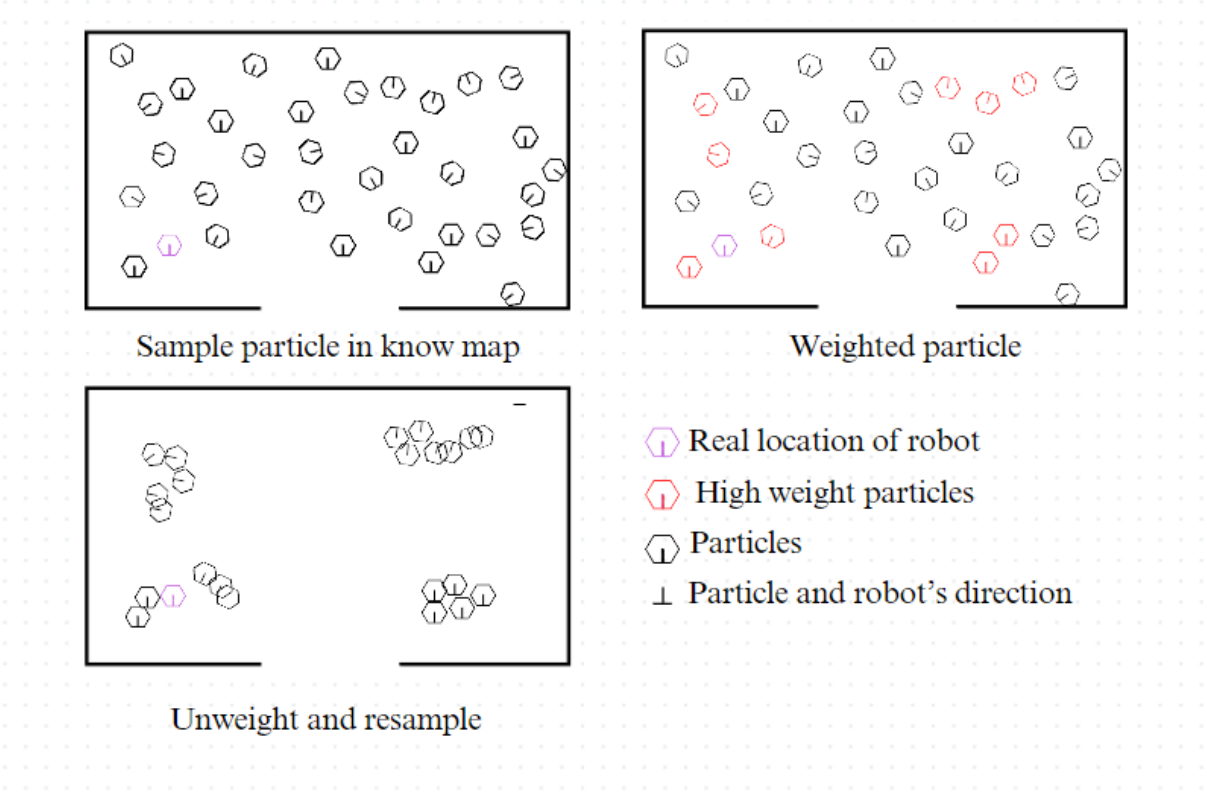

Figure 6: The probability densities and particle sets for one iteration of the algorithm.

To be more specific, for example, we randomly sample N particles in the know map with the formula $S_k = \{s^i_k; i=1\ldots N\}$ and following the steps below:

1. Each particle in S_k represents a possible location of the robot within the map.
2. From the set of particles S_{k-1} from the previous time step apply the motion model to each particle sik-1 to predict its next state. Specifically, we sample from a probability distribution $p(x_k|S^i_{k-1},u_{k-1})$, u_{k-1} is the input command from the previous step. It creates a new particle set S'_k, the set presents the prediction of the position of the robot in k.
3. Then we measure the weight of each samples in S'k by the weigh $w^i_k = p(z_k|S'^i_k)$ (z_k is the information that the sensor provides). The higher the weight, the more likely it is that the particle's position can be the robot's position. Then we resample the particles with high weight, turn them into new particles set S_k.
4. Repeating steps 2 and 3 recursively a large enough number of times, we will get the convergence of the particles, thereby getting the position of the robot.

2. Methodology

Innovating upon the foundational framework outlined in Yabloc [16], this study pioneers a novel approach to localization pose estimation by integrating a Particles Filter with advanced lane detection capabilities powered by deep learning techniques. This innovative fusion of sensor data with predefined lanelet2 maps offers a substantial improvement over traditional methods reliant solely on Opencv algorithms, ensuring heightened robustness across diverse environmental conditions.

However, while this method significantly enhances localization accuracy, it also recognizes the inherent limitations of relying solely on lane lines for position correction. Particularly in high-speed highway scenarios characterized by straight lanes, this approach may result in notable longitudinal errors. To surmount this challenge, the study introduces an IMU corrector mechanism designed to harness the rich data provided by Inertial Measurement Units (IMU).

This sophisticated approach represents a significant advancement in localization pose estimation, effectively bridging the gap between traditional sensor fusion techniques and emerging deep learning methodologies. By seamlessly integrating IMU data with the Particle Filter framework, the study not only enhances accuracy but also establishes a foundation for robust performance in complex real-world scenarios. Ultimately, this nuanced approach promises to revolutionize localization systems, offering unprecedented levels of reliability and precision across a wide range of applications and environments.

2.1. GNSS Corrector

With initial pose from GNSS, particles normal-distributed initialized and then fine-tune yaw angle by align vector map and segmentation mask. Assume $sigma$ as the standard deviation of particles's distribution, for each GNSS observation, GNSS Corrector computes the difference vector p_{dif} between the observed GNSS position and the mean position derived from the particle filter using Mahalanobis distance to make sure this is a valid observation instead of noise. The Mahalanobis distance is calculated as:

$$mal_{dis} = \sqrt{p_{dif}^{T} \cdot sigma^{-1} \cdot p_{dif}} \tag{1}$$

After being computed, this compare the mal_{dis} with a predefined threshold. If this exceeds the threshold, consider the GNSS observation as an outlier and reject it. Otherwise, accept this GNSS observation as valid. Next time, a further check is required to see if last mean pose and current mean pose are different to skip weighting due to almost the same position. Else, for each particle in the set of Particles Filter, the Euclidean distance between the particle's position and the observed pose is computed. This distance serves as a metric for evaluating the proximity of each particle to the

GNSS observation. The Euclidean distance between a particle $p(p_x, p_y)$ and the observed pose $po(po_x, po_y)$ is calculated as:

$$E(p, po) = \sqrt{(p_x - po_x)^2 + (p_y - po_y)^2} \quad (2)$$

Then, the weight are assigned based on this distance using a normal probability density function (PDF) with predefined parameters $flat^{gnss}_{radius}$, max^{gnss}_{radius}, min^{gnss}_{weight}, max^{gnss}_{weight} to control the range and the spread distance of the weight distribution:

$$d = max(0, min(|E(p, po)| - flat^{gnss}_{radius}, max^{gnss}_{radius}) \quad (3)$$

$$c = \frac{-\log log\,(min^{gnss}_{weight} / max^{gnss}_{weight})}{max^{gnss\ 2}_{radius}} \quad (4)$$

$$W_{pf} = max^{gnss}_{weight} \times e^{(-c \times d^2)} \quad (5)$$

2.2. Camera Corrector

This corrector uses a Deep Learning Lane Detection model to get the lane line and transform it from image coordinate to map coordinate point cloud then matching with lanelet2 predefined map. But this matching is lack of longitudinal information (Fig. 3)

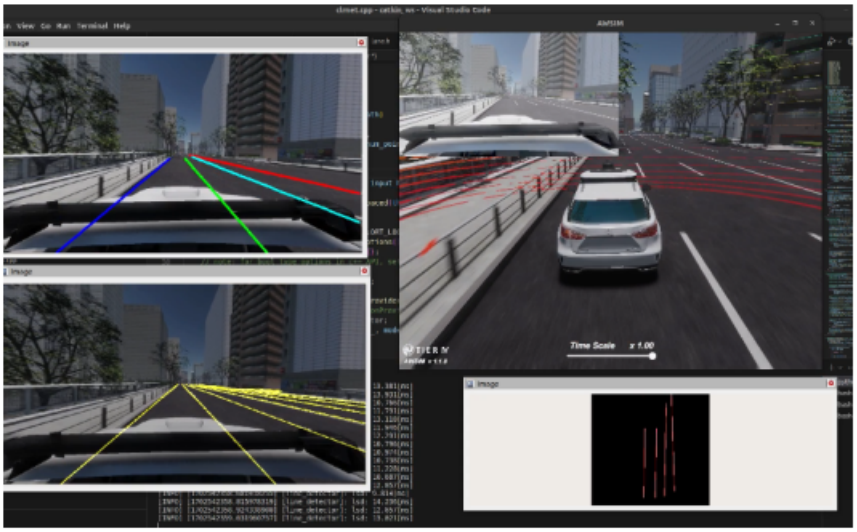

Figure 7: Camera Corrector Example

The deep learning model is designed to provide detailed lane line information, offering a comprehensive output that consists of a list of lane lines, each represented

by 72 discrete points. However, to further analyze and utilize this data, it is imperative to convert these pixel-based coordinates into a more universally applicable format. This involves a multi-step process to transform the pixel coordinates into world coordinates, making them compatible with broader spatial analysis and applications. Assume that homogenous world point coordinates is $World_p = [World_x, World_y, World_{z,}, 1]$, homogenous pixel coordinate is $Pixel_p = [Pixel_x, Pixel_y, 1]$, To initiate this conversion, the pixel coordinates are first transformed into film coordinates,then camera coordinates, and finally world coordinates using Intrinsic parameters is A, Extrinsic parameters contains Rotation is R and Translation is t:

$$Pixel_p = A[R|t]\, World_p \tag{6}$$

With is formular, 3D world coordinates can be recalculated. This is achieved by computing the bearing and distance from the camera center to the corresponding world point. The bearing is calculated as:

$$bearing = (R * A^{-1} * Pixel_p) \tag{7}$$

Following this, the distance from the camera center to the world point along the bearing vector is determined:

$$distance = -\frac{t_z}{bearing_z} \tag{8}$$

Using the bearing and distance values, the world coordinates $World_x$ and $World_y$ are computed as:

$$World_x = t_x + bearing_x * distance \tag{9}$$

$$World_y = t_y + bearing_y * distance \tag{10}$$

After compute $World_x$ and $World_y$, Assume that all line is in projected to a flat plane with z coordinate is equal to 0 which means $World_z$ is set to 0. Similar to Yabloc [16], weight of particles is computed using all world coordinates lane line P_{lane}, tangent of P_{lane} is P_t and current position. P_{pose}. Firstly, a gain value is calculated to prioritize points close to the reference position, utilizing parameters such as

$weight_{gain}$ and the differences in x and y coordinates between the lane line plane and the reference position:

$$gain = e^{weight_{gain} * \sqrt{\left(P_{lane_x} - P_{pose_x}\right)^2 + \left(P_{lane_y} - P_{pose_y}\right)^2}} \quad (11)$$

Subsequently, the cost map is queried at the reference position P_{pose} obtaining both the intensity C_v and angle of that position in cost map C_t. If the target is not mapped in the cost map, the weight of the particle remains unchanged; otherwise, the weight is adjusted according to the following equation:

$$Weight_{cur} = Weight_{pre} + gain * \left(|cos\left(P_t, C_t\right)| * C_v - 0.5\right) \quad (12)$$

This iterative process ensures that the weight of particles accurately reflects their alignment with the lane lines and the surrounding environment, facilitating robust lateral localization.

2.3. IMU Corrector

The process of refining the localization and navigation system involves several intricate steps, each contributing to the accuracy and reliability of the overall framework. Beginning with the initialization phase, the Inertial Measurement Unit (IMU) Dead-reckoning pose is established using an initialization pose. Subsequently, leveraging data from the IMU and vehicle odometry, the dead-reckoning pose P_{imu} is determined, while the particles filter's pose is denoted as P_{pf}, with the yaw of the particles' mean pose defined as θ_{pf}. To establish the equation of the line passing through P_{pf} and have θ_{pf}, the following expression is utilized::

$$y = m\left(x - x_{pf}\right) + y_{pf} \quad (13)$$

With m is the slope of the line is defined as:

$$m = tan(\theta_{pf}) \quad (14)$$

Next, the projection of P_{imu} onto the line defined by Equation 13 is conducted, leading to the calculation of the distance between the projected point $P^p_{imu} = [x^p_{imu}, y^p_{imu}]$ with P_{pf}. This process involves the following equations:

$$x^p_{imu} = \frac{m^2\left(x_{pf} - 1\right) + m\left(y_{imu} - y_{pf}\right) + x_{imu}}{m^2 + 1} \quad (15)$$

$$y_{imu}^{p} = m\left(x_{imu}^{p} - x_{pf}\right) + y_{pf} \tag{16}$$

$$LongDis_{imu}^{p} = \sqrt{(x_{imu}^{p} - x_{pf})^{2} + (y_{imu}^{p} - y_{pf})^{2}} \tag{17}$$

$$LatDis_{imu}^{p} = \sqrt{(x_{imu}^{p} - x_{imu})^{2} + (y_{imu}^{p} - y_{imu})^{2}} \tag{18}$$

The $LongDis_{imu}^{p}$ and the $LatDis_{imu}^{p}$ of a single particle can be described in Figure 8 for demonstration how IMU pose interacts with each particle.

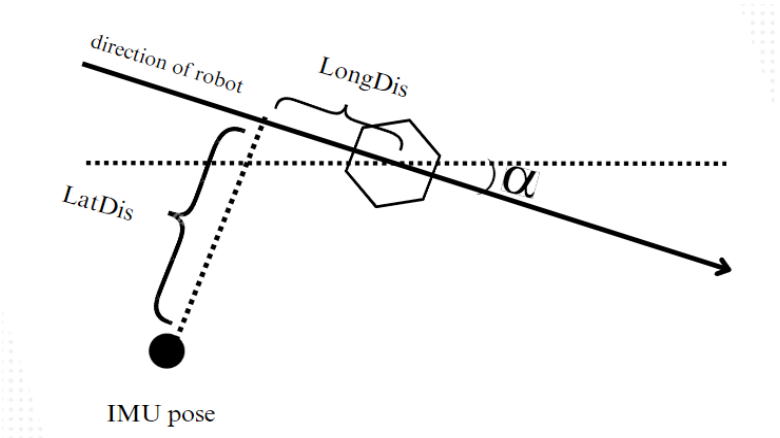

Figure 8: Lateral and Longitudinal distance calculation

After executing the aforementioned step, the system proceeds to verify whether the lateral displacement between the Inertial Measurement Unit (IMU) pose and the mean particle pose surpasses a predetermined threshold. If this condition is met, the IMU pose undergoes reinitialization, aligning it with the mean particle's pose. Consequently, the process of assigning weights to particles is omitted, in accordance with Equation 19. This operational flow is illustrated in Figure 9, depicting the reinitialization of the IMU pose when the lateral displacement between the mean pose of the particle and the IMU pose exceeds the defined threshold.

$$P_{imu} = \{mean_{P_{pf}} \; if \; LatDis_{imu}^{mean_{p}} > latDis_{threshold} \; P_{imu} \; if \; LatDis_{imu}^{mean\_p} < latDis_{threshold} \tag{19}$$

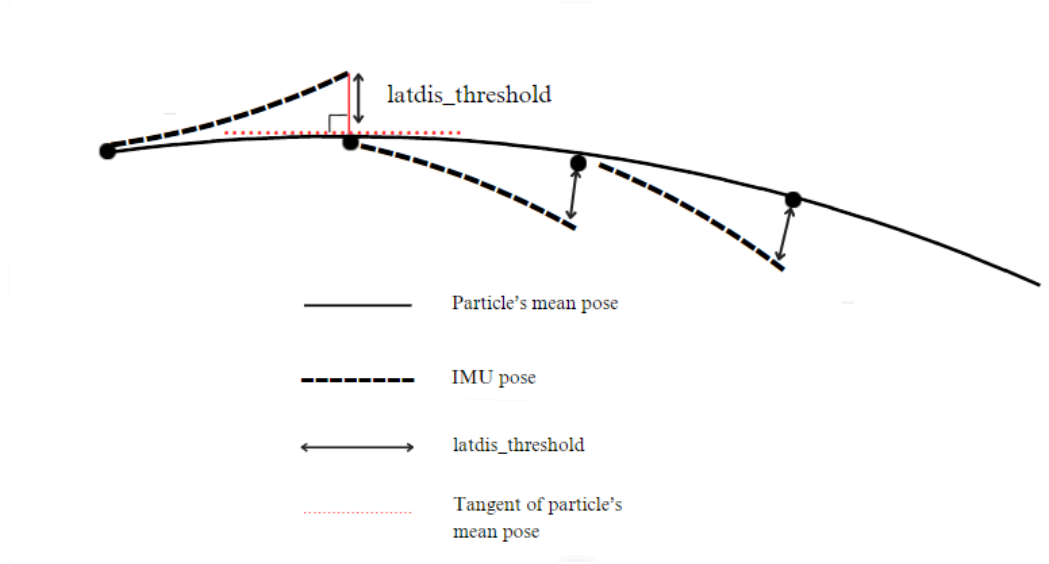

Figure 9: Reinitialize IMU pose with constraints

Then, the weight of that particle W_{pf} can be computed with $flat^{imu}_{radius}$, max^{imu}_{radius}, min^{imu}_{weight}, max^{imu}_{weight} is parameters similar to Equations 3, 4, and 5:

$$d = max(0, min(|LongDis^{p}_{imu}| - flat^{imu}_{radius}, max^{imu}_{radius}) \quad (20)$$

$$c = \frac{-\log log\,(min^{imu}_{weight} / max^{imu}_{weight})}{max^{imu}_{radius}{}^{2}} \quad (21)$$

$$W_{pf} = max^{imu}_{weight} \times e^{(-c \times d^{2})} \quad (22)$$

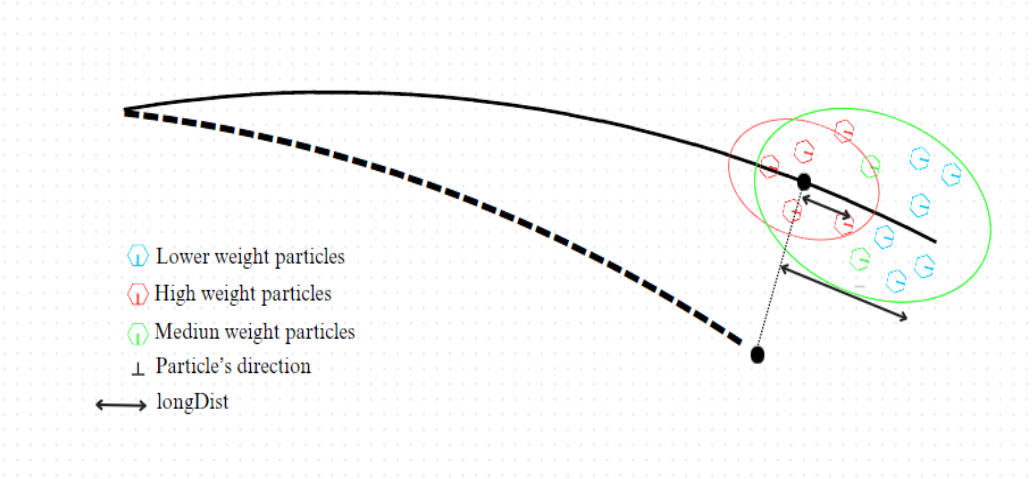

Figure 10: IMU weight update affect to final PF pose

With this fine-tune, the system can improve the longitudinal error based on lane level lateral error from previous work and gyro_odometry data from vehicle and IMU

by limiting the weight of particles to a certain longitudinal range calculated with vehicle's velocity and yaw.

In this sensor fusion system, GNSS, IMU, and camera sensors are integrated to enhance localization accuracy for the vehicle. While GNSS offers initial position estimates, its inherent inaccuracies, especially in challenging environments, necessitate correction from other sensors. The IMU provides high-frequency data, but its tendency to drift over time can result in cumulative localization errors. The camera sensor adds another layer of information, particularly useful in challenging scenarios, although it may have limitations in certain corner cases. By synchronizing their particle filters and weight updates, these sensors support each other, compensating for individual weaknesses and improving overall localization precision.

3. Experiment

3.1. Datasets and evaluation metrics

3.1.1. Datasets

In this section, the CULane dataset [17] serves as the primary resource for training and assessing the model. Traffic lane detection plays a pivotal role in autonomous driving systems, facilitating vehicle navigation, lane departure warning, and enhancing overall safety. Despite considerable advancements in computer vision, the accurate identification of lane markings across diverse environmental conditions remains a formidable challenge. The CULane dataset [17] provides a comprehensive platform for systematically investigating and tackling these challenges. This thesis endeavors to evaluate and juxtapose cutting-edge lane detection algorithms utilizing the CULane dataset [17], thereby illuminating their efficacy across various real-world scenarios.

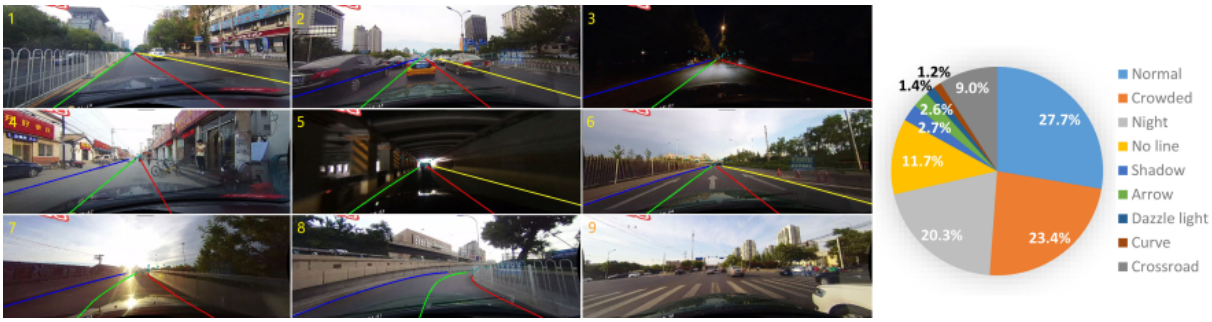

Figure 11: Example of 9 categories in CULane Dataset

CULane [17] stands out as a substantial and intricate dataset tailored for academic research into traffic lane detection. It originates from cameras installed on six distinct vehicles operated by various drivers in Beijing. Over 55 hours of video footage were amassed, yielding 133,235 frames for analysis. The dataset has been partitioned into 88,880 instances for training, 9,675 for validation, and 34,680 for testing. The test set is further categorized into a normal category and eight challenging categories, each exemplified by the nine scenarios mentioned earlier. These categories include Normal (representing lanes in regular conditions), Crowded (depicting lanes amidst heavy traffic), Night (illustrating lanes in low-light conditions), No line (showcasing lanes without road markings), Shadow, Arrow, Dazzle light, Curve (indicating lanes on curved roads), and Crossroad (depicting lanes at intersections).

Every frame within the dataset is meticulously annotated with cubic backbones delineating the traffic lanes. Even in situations where lane markings are obscured or obstructed by vehicles, the annotations are based on contextual cues, as exemplified in Figure 8. This level of detailed annotation provides a robust foundation for training and evaluating lane detection algorithms, facilitating research into enhancing their performance across diverse real-world scenarios.

3.1.2. Evaluation metrics

In lane detection tasks, the F1 score serves as a pivotal evaluation metric, offering a balanced assessment of a model's precision and recall capabilities. Its calculation involves the determination of true positives (TP), false positives (FP), and false negatives (FN) based on the model's predictions. The F1 score is derived from the harmonic mean of precision and recall, providing a single metric indicative of the model's effectiveness in detecting lane markings accurately.

Precision, denoted as the ratio of TP to the sum of TP and FP, measures the accuracy of the model's positive predictions. Conversely, recall, also known as sensitivity, quantifies the model's ability to detect all positive instances, expressed as the ratio of TP to the sum of TP and FN.

$$Precision = \frac{TP+FP}{TP} \tag{23}$$

$$Recall = \frac{TP+FN}{TP} \tag{24}$$

Where:

TP: stands for true positive, this occurs when the model correctly identifies a lane to any of the 9 lane categories as belonging to that category.
FP: stands for false positive, this occurs when the model incorrectly identifies a lane that does not belong to any of the 9 lane categories as belonging to one of those categories.
FN: stands for false negative, this occurs when the model fails to identify a lane belonging to any of the 9 lane categories as belonging to that category.

The F1 score, defined as twice the product of precision and recall divided by their sum, harmoniously balances precision and recall, ranging between 0 and 1. A higher F1 score denotes superior performance, signifying a model's proficiency in accurately detecting lane markings while minimizing false positives and false negatives.

$$F1\ Score = 2 \times \frac{(Precision+Recall)}{(Precision \times Recall)} \tag{25}$$

The F1 score serves as a crucial benchmark, offering insights into the efficacy of lane detection algorithms under consideration. It gauges the overall performance of models across varying datasets and experimental conditions, thereby facilitating informed comparisons and advancements in the field of autonomous driving and computer vision.

In the realm of localization tasks, the Mean Squared Error (MSE) serves as a fundamental metric for quantifying the accuracy of localization predictions. It measures the average squared difference between the predicted and ground truth coordinates across all samples. Formally, the MSE is computed as follows:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(pose^{i}_{gt} - pose^{i})^{2} \qquad (26)$$

With N is the total number of samples. $pose^{i}_{gt}$ represents the ground truth localization coordinates for the i^{th} sample. $pose^{i}$ represents the predicted localization coordinates for the i^{th}sample.

The MSE quantifies the average discrepancy between predicted and actual localization coordinates, with higher values indicating greater localization errors. In research and development contexts, minimizing the MSE is a primary objective, as it signifies improved accuracy in localization tasks. This metric is particularly relevant in various domains, including robotics, navigation systems, and augmented reality, where precise localization is critical for successful operation.

3.2. Implementation Detail

In the CULane dataset [17], a significant portion of redundant frames exists where the ego-vehicle remains stationary and lane annotations remain unchanged. To mitigate overfitting to these redundant frames, a strategy is employed where frames with an average pixel value difference below a threshold are removed. Through empirical validation, an optimal threshold of 15 is determined. Following this process, 55,698 frames (62.7% of the dataset) are retained for training. With this approach, the F1 score of CLRNet-DLA34 sees notable improvement from 80.47 to 80.86 with consistent 15-epoch training runs.

This research utilizes Autoware and AWSIM simulation platforms with the Nishishinjuku map on an RTX 3060 machine running Ubuntu 22.04. The deep learning model undergoes optimization using TensorRT, and all functions are optimized in C++ language with the O3 optimizer flag.

For the deep learning model, a confidence threshold of 0.15 is set to balance precision and recall. Upon detection, lanes with low confidence have their particle weight reduced by a factor of 5 as described in Equation 12. Additionally, Equation 11 incorporates a weight gains of 0.001, while Equation 19 sets $latDis_{threshold}$ to 1.0. Parameters such as $flat^{imu}_{radius}$, max^{imu}_{radius}, min^{imu}_{weight}, max^{imu}_{weight} are respectively set to

0.5, 3.0, 0.1 and 1.0. Similarly, $flat^{gnss}_{radius}$, max^{gnss}_{radius}, min^{gnss}_{weight}, max^{gnss}_{weight} are respectively set to 0.5, 10.0, 0.5 and 5.0. The Mahalanobis distance threshold is 30.0. In Figure 9, an example of the method employed in this research is depicted.

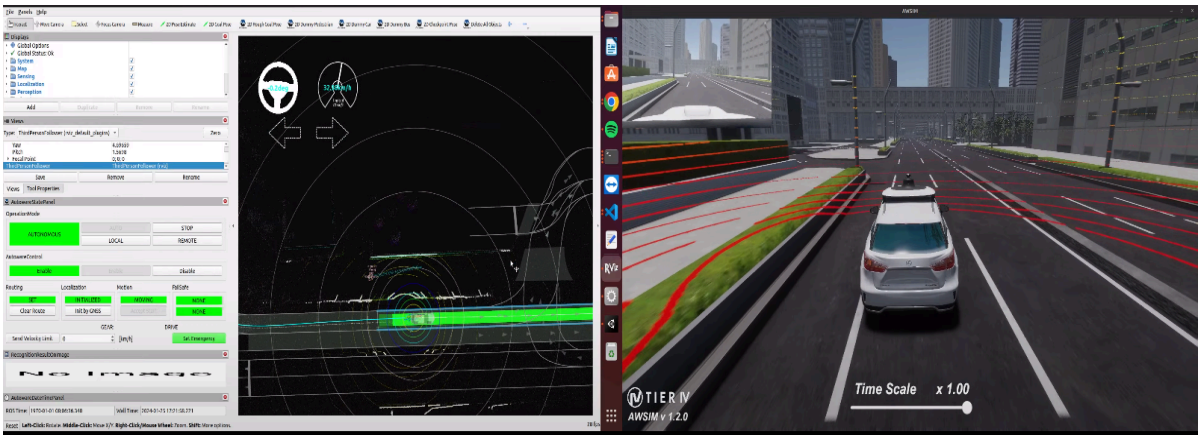

Figure 12: Integrated result of our method to Autoware and AWSIM simulation

3.3. Result Analysis

3.3.1. Whole System Result Analysis

Iterative Closest Point (ICP) and Normal Distributions Transform (NDT), LiDAR-based localization techniques, have also shown the ability to achieve centimeter-level accuracy. Still, because of their shortcomings, these approaches are not ideal for high-speed situations, even with their remarkable accuracy. Their comparatively low frequency is a significant drawback, as it could prevent them from providing updates fast enough to facilitate the kind of quick decision-making needed in high-speed settings. Furthermore, for many autonomous vehicle applications, their high cost renders their widespread deployment economically unfeasible.

Although these techniques are incredibly accurate, their low frequency and high cost make them unsuitable for high-speed scenarios. In order to enable safe and dependable autonomous driving at high speeds, it is still imperative to achieve comparable accuracy with less expensive sensor suites, albeit with somewhat less precision. It is imperative to strike a balance between real-time performance, affordability, and accuracy when creating localization solutions that satisfy the stringent needs of fast-moving autonomous vehicles.

For accurate lane-level localization at speeds of up to 50 km/h, a frame rate of approximately 15 fps is essential to maintain an estimation per meter. In scenarios involving highway speeds of up to 100 km/h, the system must achieve a minimum frame rate of about 30 fps to uphold this 1-meter estimation. Through the utilization of a proposed method involving a TensorRT lane detection model, the system can currently reach a frame rate of approximately ~35 fps under the worst-case conditions. And the opencv graph segmentation combined with [16] algorithm to find the most road-similarity is unreliable and eventually causes ego car collisions with the pavement and the mean euclidean error is unmeasurable. With our method, the ego can fine tune lateral error with lane information and longitudinal error with the help of imu

and vehicle odometer. As a result, the mean euclidean error of the proposed method is approximately 0.5m.

The ego car can retain lane-level accuracy in many driving scenarios by achieving a mean Euclidean error of 0.5 meters in localization. But this level of accuracy might not be enough for autonomous cars, particularly those operating at high speeds, to guarantee accuracy and safety. In order to make snap decisions and maneuver through complicated environments with very little room for error, high-speed vehicles need to be able to localize themselves incredibly precisely. Thus, although 0.5 meters might be adequate for simple lane-keeping duties, a higher accuracy, possibly less than 0.1 meters becomes necessary to guarantee the dependability and safety of autonomous cars, particularly at high speeds where even small deviations might result in potentially hazardous circumstances.

Table 2. Comparison of FPS and Mean euclidean error of difference visual-based localization methods in AWSIM simulation

Method	**Collision**	**FPS**	**Mean euclidean error**
Yabloc [16]	Yes	~14	Unmeasurable
Proposed method without IMU	No	~37	~1.5m
Proposed Method	**No**	**~35**	**~0.5m**

3.3.2. Corner Cases

The current AWSIM planning system can only generate a path if the goal pose is within the mapped area. This limitation prevents the ego car from navigating to unknown regions not represented in the map. Another issue arises when there are discrepancies between the map and real-world conditions. In such cases, the system may fail dramatically due to the disparities between the lanelet2 map and actual road conditions. Addressing these challenges typically requires manual efforts to update the map according to real-world observations. However, as perception technology advances and becomes capable of producing highly accurate road markings, there is potential for leveraging manual driving to build maps using perception data, with minimal additional effort required for fine-tuning. Ultimately, an improved map leads to better localization capabilities, highlighting the importance of addressing these challenges for robust autonomous driving systems.

When road markings are inadequately detected, they can be broadly categorized into two types: those with low recall and those with low precision. Low recall instances entail a reduced amount of information available for the matching process, leading to diminished confidence in the accuracy of localization. Conversely, low precision scenarios involve the inadvertent utilization of erroneous road markings for matching with the lanelet2 map. This erroneous matching can deceive the system into believing that localization is reliable, as even non-road markings may coincidentally align with positions on the lanelet2 map when transformed using the camera matrix. This highlights the potential pitfalls in relying solely on visual data for localization in dynamic environments.

In the scenario where GNSS signal is not reachable, the ego car employs a multi-faceted approach to maintain localization amidst challenges such as GNSS signal loss. Initially, it relies on its previous estimated pose and integrates IMU data for dead reckoning, providing a baseline for position tracking. This strategy allows the vehicle to maintain a continuous sense of its position, compensating for temporary disruptions in external localization cues.

Subsequently, the camera system comes into play, leveraging visual inputs to refine and correct the final pose estimation. By analyzing the surroundings and detecting road markings, lane boundaries, and other visual cues, the camera system enhances the accuracy of localization, especially in areas where GNSS signals might be unreliable or unavailable.

However, as the duration of GNSS signal loss prolongs, the uncertainty associated with the localization pose gradually increases. This escalation in uncertainty poses a significant challenge to the reliability of the ego car's position estimation. To address this, the vehicle employs a proactive measure: it initiates a predetermined protocol to halt its movement once a specified timestamp is reached following the loss of GNSS signal.

By halting under such circumstances, the ego car prioritizes safety and risk mitigation, recognizing the potential dangers associated with navigating without reliable localization information. This decision serves to prevent potentially hazardous situations that could arise from inaccuracies in position estimation, ensuring the safety of both occupants and surrounding traffic participants.

In essence, the ego car's adaptive approach to localization management underscores the importance of integrating multiple sensor modalities and implementing robust safety protocols to navigate effectively in dynamic environments, even in the face of challenging conditions such as GNSS signal loss.

4. Conclusion & Future Work

The proposed approach addresses the limitations inherent in traditional lane detection algorithms by leveraging the capabilities of a deep learning model. This model offers enhanced reliability in detecting lane lines, thereby providing a robust foundation for subsequent localization refinement. By integrating the deep learning-based lane detection output with the Lanelet2 predefined map, the lateral error is finely tuned, ensuring accurate alignment with the roadway geometry.

Furthermore, to refine the longitudinal error, additional sensor data such as Inertial Measurement Unit (IMU) data and vehicle odometry are utilized. These complementary sources of information enable precise correction of longitudinal positioning discrepancies, thereby enhancing overall localization accuracy.

The efficacy of the proposed method is underscored by its superior performance compared to previous works, as evidenced by both improved Frames Per Second (FPS) and reduced position error, as detailed in Table 2. Despite the notable advancements achieved, it's acknowledged that challenges persist, particularly in scenarios where lane lines are curved or not clearly visible, leading to occasional failures in the AWSIM simulation environment.

As the core lane detection model relies on CLRNet, trained on the CULane dataset, there remains room for further enhancement through the continuous refinement of this model. Future iterations of the approach stand to benefit from advancements in lane detection technology, promising even greater robustness and reliability across diverse environmental conditions.

In conclusion, the proposed method represents a significant stride forward in visual-based mono-camera localization challenges. Its comprehensive integration of deep learning-based lane detection, map matching, and sensor fusion techniques lays a solid foundation for future research endeavors in this domain, offering valuable insights and opportunities for further innovation and improvement.

References

1. Davison, A. J., Reid, I. D., Molton, N. D., & Stasse, O. MonoSLAM: Real-Time Single Camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6), 1052-1067. DOI: 10.1109/TPAMI.2007.1049. (2007)
2. Mur-Artal, R., Montiel, J. M. M., & Tardós, J. D. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, 31(5), 1147-1163. DOI: 10.1109/TRO.2015.2463671. (2015)
3. Wang, S., Clark, R., Wen, H., & Trigoni, N. DeepVO: Towards End-to-End Visual Odometry with Deep Recurrent Convolutional Neural Networks. arXiv preprint arXiv:1709.08429. DOI: 10.1109/ICRA.2017.7989236 (2017).
4. Mur-Artal, R., & Tardós, J. D. ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. arXiv preprint arXiv:1610.06475. DOI: 10.1109/TRO.2017.2705103 (2016)
5. Campos, C., Elvira, R., Rodríguez, J. J. G., Montiel, J. M. M., & Tardós, J. D. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM. arXiv preprint arXiv:2007.11898. DOI: 10.1109/TRO.2021.3075644 (2020)
6. Neven, D., De Brabandere, B., Georgoulis, S., Proesmans, M., & Van Gool, L. Towards End-to-End Lane Detection: an Instance Segmentation Approach. arXiv preprint arXiv:1802.05591 (2018)
7. Lo, S.-Y., Hang, H.-M., Chan, S.-W., & Lin, J.-J. Multi-Class Lane Semantic Segmentation using Efficient Convolutional Networks. arXiv preprint arXiv:1907.09438. (2019)
8. Chang, D., Chirakkal, V., Goswami, S., Hasan, M., Jung, T., Kang, J., Kee, S.-C., Lee, D., & Singh, A. P. Multi-lane Detection Using Instance Segmentation and Attentive Voting. arXiv preprint arXiv:2001.00236. (2020)
9. L. Tabelini, et al. (2021). "PolyLaneNet: Lane Estimation via Deep Polynomial Regression," in 2020 25th International Conference on Pattern Recognition (ICPR), Milan, Italy, pp. 6150-6156. doi: 10.1109/ICPR48806.2021.9412265
10. Meyer, A., Skudlik, P., Pauls, J.-H., & Stiller, C. YOLinO: Generic Single Shot Polyline Detection in Real Time. In Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops (pp. 2916-2925). (2021).
11. Tabelini, L., Berriel, R., Paixão, T. M., Badue, C., De Souza, A. F., & Oliveira-Santos, T. Keep your Eyes on the Lane: Real-time Attention-guided Lane Detection. In Conference on Computer Vision and Pattern Recognition (CVPR). (2021)
12. Zheng, T., Huang, Y., Liu, Y., Tang, W., Yang, Z., Cai, D., & He, X. CLRNet: Cross Layer Refinement Network for Lane Detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 898-907). (2022)
13. Liu, L., Chen, X., Zhu, S., & Tan, P. (2021). CondLaneNet: a Top-to-down Lane Detection Framework Based on Conditional Convolution. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2021, pp. 3773-3782. arXiv preprint arXiv:2105.05003.

14. David, J.-A. (2015). Visual Map-based Localization applied to Autonomous Vehicles. Available at: https://api.semanticscholar.org/CorpusID:111159539
15. Sadli, R., Afkir, M., Hadid, A., Rivenq, A., & Taleb-Ahmed, A. (2022). Map-Matching-Based Localization Using Camera and Low-Cost GPS For Lane-Level Accuracy. *Procedia Computer Science*, 198, 255-262. ISSN 1877-0509. DOI: 10.1016/j.procs.2021.12.237.
16. Tier4. (2023). YabLoc: A Repository for visual-based localization method. GitHub. https://github.com/tier4/YabLoc
17. Zhou, Y., Sun, J., & Li, Y. (2019). CULane: A Large-Scale Dataset for Semantic Segmentation of Urban Scenes. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (pp. 1273-1282).