Stage Fin d'Étude: Research and Development on a precise visual localization system based on particle filter using land marker features

BIE Xiaoyu

Tutor: ZHENG Jingsen, YI Shichun

Department: Autonomous Driving Unit(ADU), Baidu Inc.

Abstract

This report is a summary of my 5 months work of internship in the department of ADU(Autonomous Driving Unit) at Baidu Inc. During my internship, my work focuses on the research and development of a particle filter based visual localization system. In this system, we firstly use monocular camera and IMU(inertial measurement unit) to obtain a precise pose estimation for autonomous vehicle, then we introduce a stereo camera module and define a new feature match method to reach a more stable results. The whole system is developed by C++ on Linux, and provide a visualization demo using an open source lightweight library *Pangolin*.

1 Introduction

Autonomous driving technology has been attracting more and more attention, both in industrial and in academia, whether because of its tens of billions of markets, or because of its potential to change people's travel mode. Nowadays, many high-tech companies like Waymo, Uber, Baidu and OEMs like Daimler, Renault have built their own auto-driving R&D group, willing to bring a L3 or L4 level auto-driving solution in the next few years.

Among the autonomous driving system, vehicle localization is obviously one of the most import component, which provides indispensable information for the other auto-driving algorithm, such as traffic signs perception, behavior decision, vehicle motion planning. In the early years, we usually apply a high-cost GPS and dead reckoning fusion system to obtain an accurate localization information [1]. However, this system can not work well in the urban canyon environments or in some poor satellite visibility condition, even with real time kinematic(RTK) capabilities.

In practice, the autonomous driving vehicle is equipped with many types of sensors, such as cameras, radars, IMU(inertial measurement unit) etc. In the other word, we have many methods to obtain the localization information. Particularly, the technology of light detection and ranging, which is also known as Lidar, can receive an extremely accurate measurement in space distance. Using these devices, the multi-sensor fusion based localization algorithm provide precise and robust localization [2] [3].

Although the Lidar based methods achieve very good results, its immense consumption in sensor hardware blocks it away from low-cost and mass production. An alternative solution is to use some cheaper sensors, like camera, to replace the high-cost part. The camera can not only provide color and semantic information, but also contains its pose information in 3D-space. Many methods have been developed in ego-pose estimation [4] and its calibration [5] these technologies have been successfully applied in simultaneously localization and mapping(SLAM) and structure from motion(SFM) [6–8], or they can combine the other sensors like IMU to establish a visual inertial odometry(VIO) system [9].

In our group, we aim to develop a precise visual localization system. We have finished the first version, it is a particle filter based visual inertial localization system. During my whole internship, my main task is to take over this project developed by the predecessors, finish and debug this system with off-line data which are collected by our company in practice, and try to optimize it.

The reset of this report is organized as follows. Section 2 summarizes the research on related work in localization, or furthermore pose estimation. Section 3 presents our system overview. Section 4 demonstrate the evaluation for the implementation of our localization system to off-line datasets and optimization methods. Section 5 is the conclusion for my internship.

2 Related work

2.1 Pose estimation

In the former days, the integration of GPS and motion information is widely used in ego-pose estimation, such as the mobile phone and low-cost drone. Motion information can compensate for the short time outage and low update frequency of GPS. In the meantime, the drift caused by long time integration can be corrected by GPS. However, when the GPS has been invalid for a long time, which frequently occurs in urban canyon environment, this system can not overcome the integration error. Furthermore, the systematic noise property of the GPS that changes slowly with time is hard to correct only using the motion information [10].

To avoid this shortage, localization systems based on matching of a high definition map with perception data have been widely researched. With the different use of sensors, the localization

system can be divided in Lidar based localization [2] and vision based localization [10]. Lidar based localization system can provide a high precise depth image or 3D cloud point using TOF(time of flight) technology, which guarantees the localization precision and is inherently robust against illumination changes. However, the requirement of hardware is immense in this method. In the contrary, vision based localization system has a very low expenditure and can also achieve a high prevision. As in [2, 6, 10, 11], the map-perception-aided localization system can be classified into three group: 1) Data association based localization system, 2) Signal-level filter based localization, 3) Feature-level filter based localization.

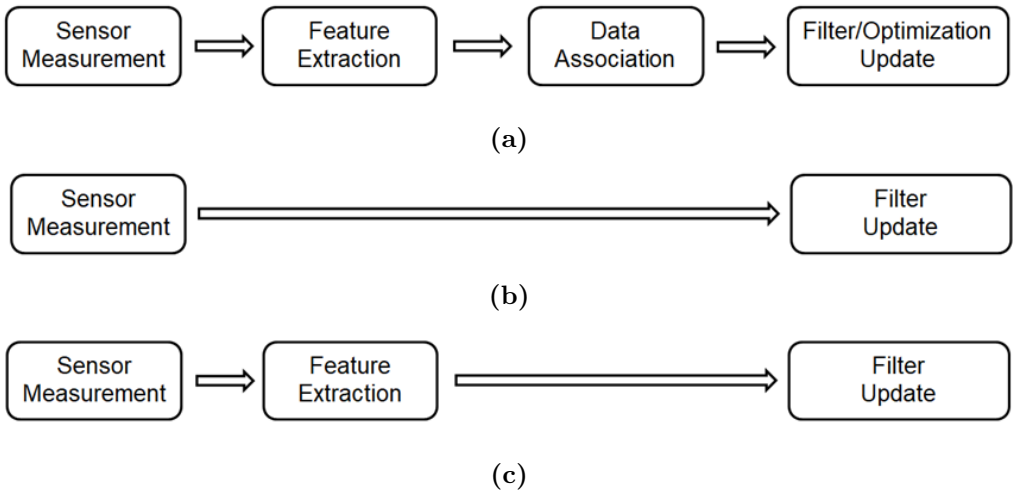


Figure 1: The three groups for map-perception-aided localization. (a) Data association based localization system. (b) Signal-level filter based localization. (c) Feature-level filter based localization.

A. Data association based localization system

Data association based localization system is a typical solution in simultaneous localization and mapping(SLAM). Figure 1 (a) demonstrate its pipeline. We extract features from sensor measurement(such as cameras, Lidars and radars). In camera based feature extraction, for example, we usually choose feature points that are easy to distinguish. This means, these points are with a large gradient. During the ego-motion of cameras, these points are affected by camera rotation, zoom and luminous changes, so some descriptors are applied to avoid it, like SIFT or BRIEF. After data association, we could use the filter update or optimization update to obtain a highly accurate pose estimation. Kalman filtering(extended Kalman filtering for nonlinear system) updates the state mean and covariance to calculate the true pose, while bundle adjustment(or pose graph if the map need not to update) use iterative optimization to find it.

In filter based state update, we only calculate once in each step, which means a O(1) time complexity. It has the advantage of the simple calculation framework and a low computational burden, many previous research have applied this method. However, Kalman filter assume the linearity of system and needs an accurate initialization for pose state and covariance. In practical, the ego-pose estimation system is non-linear and it is difficult to obtain a perfect initialization. Extended Kalman filter(EKF) can approximate the probability distributions of nonlinear systems, however, EKF suffers from the nonlinear errors because it only preserves the first order in the Taylor extension of nonlinear system.

In optimization based state update, we represent the ego-pose, 6 degree of freedom(DOF), in Lie algebra in order to calculate its Jacobian matrix. With the Jacobian matrix, it is easy to solve this least square problem using Gaussian-Newton method or Levenberg-Marquardt method. With adequate computational resources, optimization based pose estimation can achieve a higher accuracy than that of filter based method [12].

Although the data association methods can either have a simple calculation process or a high precision, there are limitations when they are applied in cluttered environments. In these environments, data associations is not easy because the sensor measurements are affected by clutter.

B. Signal-level filter based localization

Differ from the data association based method, the Monte-Carlo sampling can deal with nonlinear system, with unconventional noise model and with cluttered environment, because in Monte-Carlo sampling, we can create any model we want and choose the best result or a weighted average with presetting decision rules. There are two typical Monte-Carlo localization methods: signal-level and feature level localization [10].

As shown in Figure 1 (b), signal-level Monte-Carlo localization system update the state directly using the raw sensor data without feature extractions, as shown in Figure 1 (b). Due to the direct use of raw sensor data, there is no loss of sensor information from feature extraction and it can achieve high stability. In Lidar based localization system [2], the reflectivity data from Lidar are directly applied to the measurement updates of the localization system, the estimated pose is calculated by a histogram filter. However, as all the reflectivity data for roads need stored in the map and all these data are involved in estimation process, this localization system requires huge memory storage and high computational ability.

C. Feature-level filter based localization

In the Figure 1(c), unlike the signal-level filter based localization system, feature-level filter based algorithm extract features before the state update. On the one hand, feature-level algorithm use the sparse information in filter update, the computational burden therefore may be smaller than that for signal-level localization, on the other hand, due to the Monte-Carlo method, especially particle filter, we need not to establish an accurate data association before the measurement update, we simply calculate the weight for each particle. Its performance is determined by which features are used, how we choose the noise model and how we define the cost function of weight computation.

Generally, in autonomous driving, the features widely used are land markers, which are easy to detect and constantly exist. They are standardized, they would not change with climate and we can even obtain prior information. As for the noise model, since the IMU and wheel speed sensor can always provide a precise pose prediction in short period of time, we commonly consider the noise model for them are Gaussian in the step of particle generation. Therefore, the key point for feature-level filter based localization, specifically particle filter based localization, is how to define the cost function of weight computation. In the section 2.2, we describe several methods of cost function definition. Since most of lane markers have a line shape, we focus on the definition for line features. Although not all of them use the particle filter, their ideas can also be useful in the particle weight computation.

2.2 Cost function definition

A. Likelihood field intersection

In [10], shown in Figure 2, features are road surface markers(RSMs) such as lanes, traffic signs, stop lines and crosswalks to . All these features extracted from multiple cameras and from local map are projected into a virtual bird's-eye view image. Then a particle filter is applied to execute the filter update.

The details for particle filter will be demonstrated in section 3.3. For instance, we simply consider that we maintain some particles and each of them contains a vehicle pose and a weight. Among the particles, whose pose is closer to the truth should have a higher weight. The final estimated pose depends on these particles. In [10], these weights depend on the intersection of likelihood field, as shown in Figure 2. By applying the probabilistic noise model of the RSM features to all the pixels in the predicted measurements, the likelihood field generation is composed of two parts: noise from unexpected ego-vehicle motion (roll and pitch) and noise from fault detection.

The noise from unexpected motion is considered as Gaussian and its likelihood filed can be obtain using a Gaussian mixture, its variance distribution is shown in Figure 3 and the likelihood filed generation is described below:

$$p^{CAM}_{motion}(z^{CAM}_k|x_k,m) = \sum_{j=1}^{M} \frac{1}{M} p^{CAM}_{motion}((z^j)^{CAM}_K|x_k,m) \tag{1}$$

M represents the number of predicted feature pixels in the prediction measurement. Under this condition, we can see that the further features have a higher variance, which means a higher

tolerance to dis-match, while the closer features will be more sensitive to it.

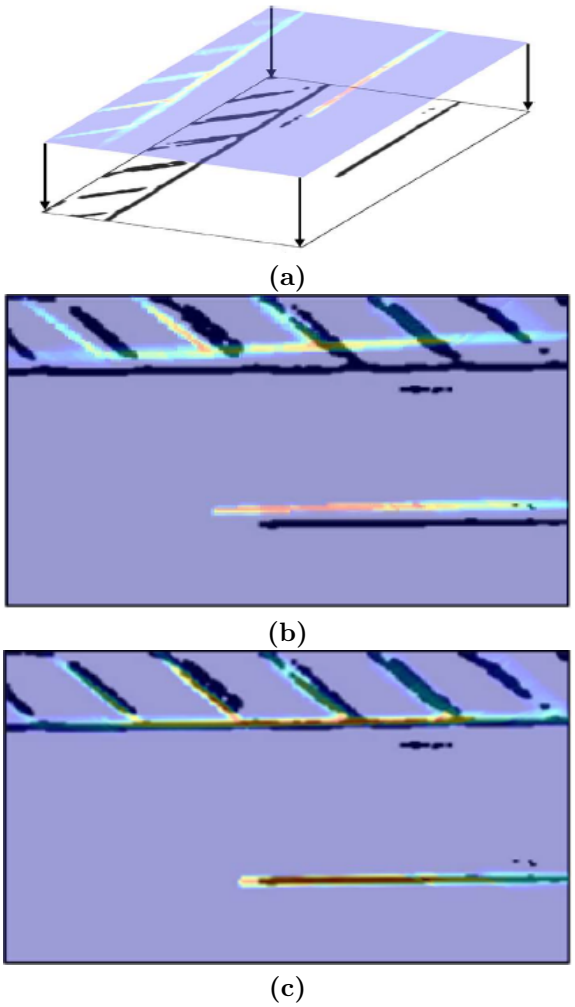


Figure 2: Measurement update process of the RSM features [10]. (a) Measurement update by matching the likelihood filed with RSM features. (b) Example of the poor matching. (c) Example of the correct matching.

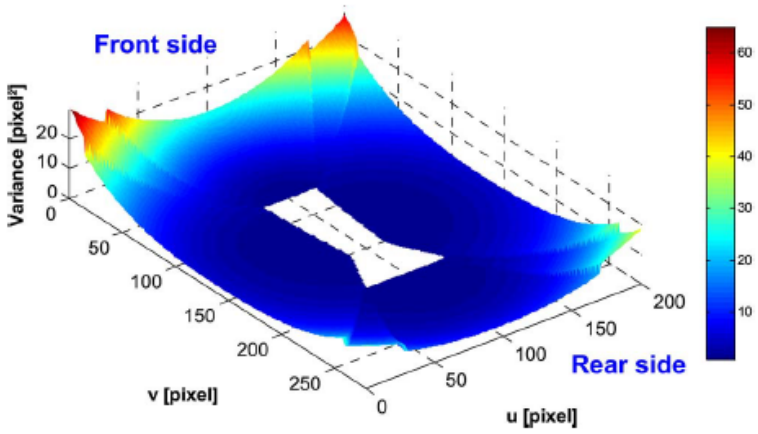


Figure 3: Example of variance of the Gaussian noise model for the AVM system due to the unexpected vehicle motion [10]

As for fault detection, this noise is regarded as random and can be represented using a uniform distribution:

$$p_{fault}^{CAM}(z_k^{CAM}|x_k, m) = \frac{1}{N_{pixels}} \tag{2}$$

N_{pixels} is the number of pixels on the image.
The final likelihood filed is generated by integration of these two parts:

$$p^{CAM}(z_k^{CAM}|x_k, m) = \alpha^{CAM} \cdot p_{motion}^{CAM}(z_k^{CAM}|x_k, m) + \beta^{CAM} \cdot p_{fault}^{CAM}(z_k^{CAM}|x_k, m) \tag{3}$$

while the α and β is determined by ratio of true-positive detection rate to false-positive detection rate of RSM features:

$$\frac{True - positiverate}{False - positiverate} = \frac{\alpha}{\beta} \tag{4}$$

B. Residuals from dashed lane marking

In [13], they use a Kalman filter, not the particle filter. Features in the HD-map are directly projected on the camera images and have been sampled into points. Cost function is defined as the residual between a map point and a measuring point, as shown in Figure 4 and Figure 5:

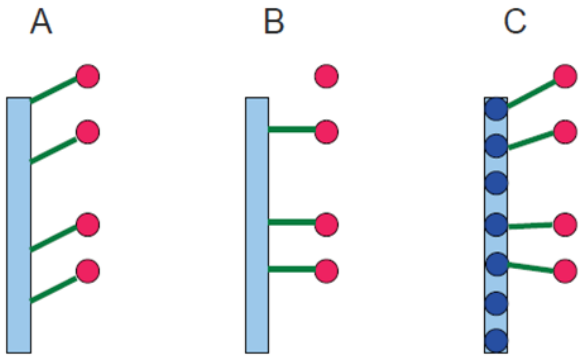


Figure 4: Three cases of measurement connections at the birds eye view of one dashed lane marking [13]; red points are marking measurements, the connections/residuals are shown as green line. (A) shows is the best case, unfortunately not possible; (B) shows the minimum point line distance match with no longitudinal constraints; (C) shows the connections with sampled map-points from map lines

Figure 5: Marking Detection (red crosses) and Map Matching (green lines) on real word data. The map is drawn in blue lines [13].

For each of map points, they search a closest measurement points to compute the residuals:

$$\vec{r} = \vec{P_m} - \vec{P_e} \tag{5}$$

$\vec{r}$ is the residual, $\vec{P_m}$ and $\vec{P_e}$ are measurement point and map point.

C. Line projection error

In geometric computer vision algorithms, point features are widely used to compute the camera pose and its ego-motion, however, correct points correspondence are sometimes difficult to realize because of low textured scenes. Some research therefore combine the line features and point features to increase the system robustness [11, 14]. For this reason, they define the cost function for line features, which is called: line projection error.

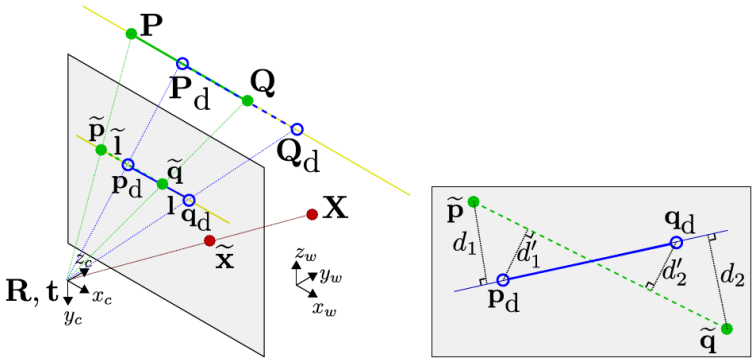


Figure 6: Line projection and error computation [11]

As shown in Figure 6-left. P, Q are the endpoints of a 3D line, and their 2D projection to the image plane are $\tilde{p}$, $\tilde{q}$. P_d, Q_d are endpoints of detected line, while p_d, q_d are their projection. Ideally, line $\overline{pq}$ and line $\overline{p_d q_d}$ should be coincident if the camera pose is precisely estimated. however, estimated pose is always a little different from truth, so there exist a projection error between $\overline{PQ}$ and $\overline{P_d Q_d}$, as shown in 6-right and demonstrated as:

$$E_{line}(P, Q) = d_1^2 + d_2^2 \tag{6}$$

Note that in practice, due to real conditions such as line occlusions or mis-detection, the image detected endpoints p_d, q_d will not match the projection of endpoints P, Q, so it will be more reasonable to define the line projection error as:

$$E_{line}(P, Q) = d'^2_1 + d'^2_2 \tag{7}$$

D. Direct edge alignment

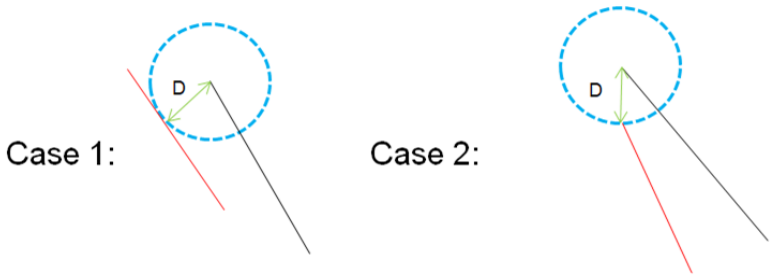


Figure 7: Direct edge alignment

In [15], the researchers propose a new method to calculate the line projection error, which is called direct edge alignment. In fact, the line projection error shown in Figure 6 can well minimize the projection error when the direction of lines is arbitrary, however, it can not handle the condition when their direction are nearly the same. Because it only minimize the distance between endpoints and lines, and has week constraint in along the lines' direction. So, in Figure 7, a more restrict error computation has been demonstrated. In this method, we take the consideration of the projection of endpoints. When the projection of an endpoint is inside the line, we use the distance between endpoint and line as cost error, when it is outside, we use the distance between two endpoints instead. In the other word, the cost function is the sum of the re-projections and nearest edge points in current image:

$$f(R,T) = \sum_i \min_j D^2(\Pi[R^T(P_i - T)], u_j) \tag{8}$$

where D denotes the Euclidean distance between those points.

In our localization system, we basically refer to these two articles in [10, 13] shown above. Identically, We choose the feature-level Monte-Carlo localization strategy by using particle filter and land markers. However, considering the practical implementation and the research progress among the whole team, we make some adaptation and optimization. For example, the features in HD-map are simplified to groups of endpoints location and their attributes, and we introduce a soft constraint feature association. As for the next two methods, we use them in the extended optimization.

3 System overview

In fact, the whole localization system consists of the front end module and the back end module. In the front end module, we use some deep learning methods to obtain a semantic image which contains the lane markings as its foreground and the others as tis background, and we can also obtain a relative move of vehicle pose using an inertial measurement unit(IMU) and a wheel speed sensors. Satellite navigation(GNSS) is used to provide an initial pose estimation and a valid boundary during localization [10]. Thanks my colleagues for perfectly doing this work for us, our main job content focus on the back end module. It is to use all these sensor information to acquire a precise pose estimation of vehicle, it is also called multi-sensor fusion localization.

3.1 System pipeline

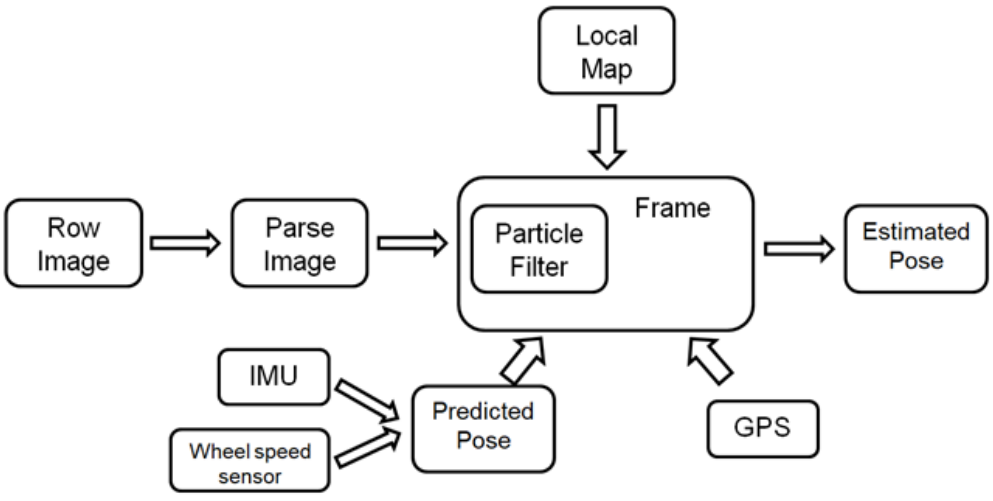


Figure 8: Localization system pipeline.

In the Figure 8, we present our system pipeline. It highly resembles that of the feature level Monte Carlo localization [10,13]. Seen that double integral of accelerometer in IMU(inertial measurement unit) will introduce a large uncertainty of translation in pose prediction, hence we use a wheel speed sensor to provide the translation and use the gyroscope in IMU to provide the rotation prediction. The predicted pose from these two sensors is considered as time update in particle filter. Using this prediction, we extract a relevant part of HD map as local map. Then, with the observation of image-map match, we can finally obtain a precise pose estimation. Position from GPS is only used to provide an initialization and give a valid boundary for particles. The details of particle filter and image-map match will be explained in section 3.3 and 3.4.

3.2 Software framework

Our localization algorithm is realized in Linux, based on C++. The framework is shown below:

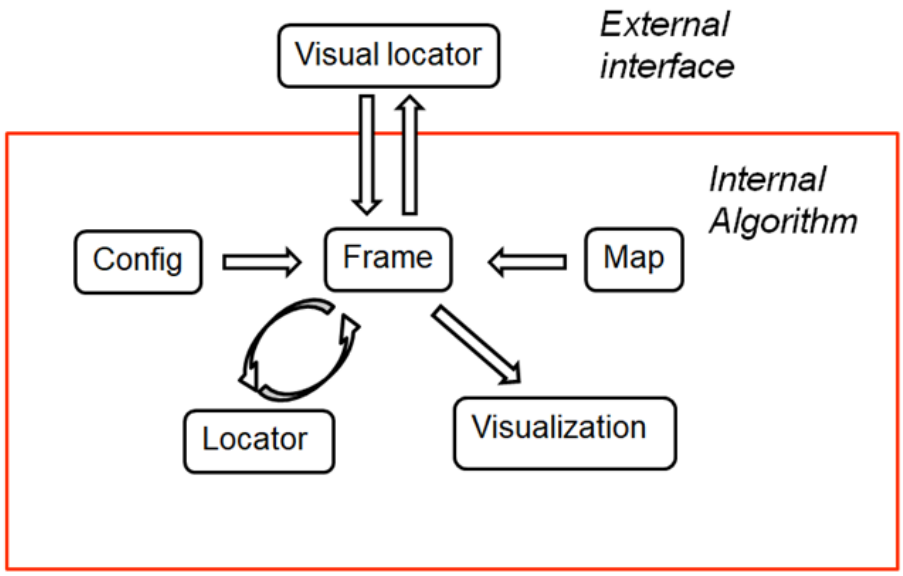


Figure 9: Software framework

In reality, we have mainly six modules in our system.:

- "visual locator" works as an external interface, in this module, we read external data and transfer them into our internal algorithm, and it can also obtain the final results for pose estimation
- "frame" works as a platform in the internal algorithm, which connects all the other modules. We save all the intermediate variables in this module, and deliver them where they are needed.
- "map" is in charge of generating local map from the pre-processed HD map. For each estimation step, when we get the prediction pose, we can choose relevant part of map, after proper cutting and mosaic, we can extract the local map in front of or behind the vehicle
- "config" consists of some hyper-parameters such as valid threshold for line match, particle numbers etc
- visualization is the visualization part. which uses the *Pangolin*, an open source lightweight library based on *OpenGL*. It will show the vehicle trajectory, the localization precision, and a reprojection of HD-map on images.
- "locator" is the core part in the whole system, which contains the particle filter algorithm and the lane marker match algorithm.

In the Figure 10 below, we demonstrate our visualization window. In the middle of window, we show the trajectory of estimated vehicle pose and the ground truth obtained by NovAtel. We also visualize the land markers obtained both from local map(the white lines and red lines) and from the projection of parse images(the blue lines). In the left of window there is a command bar to control the visualization. In the top right, we plot the lateral localization error, and the longitudinal error below it. In the bottom right, it is a blended image which is a combination of raw images, its parsed land markers and the reprojection of local map. Using this visualization tool, we can roughly evaluate the system performance and easily find the bugs if there exist.

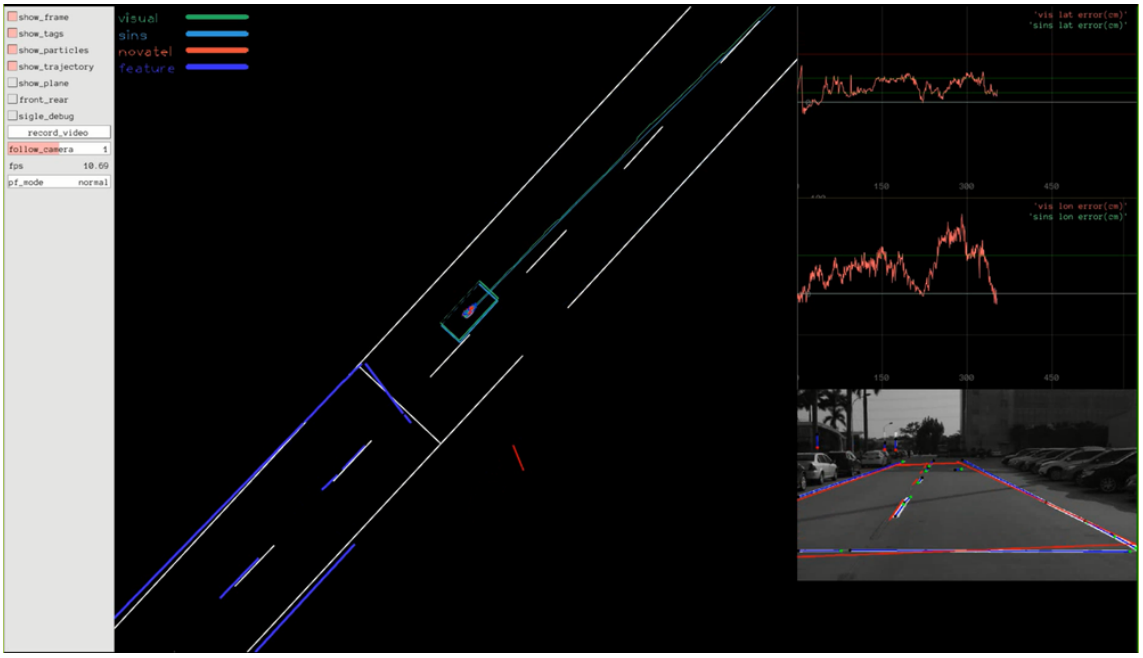


Figure 10: Visualization

3.3 Particle filter

3.3.1 Fundamental theory

In fact, the particle filter is a nonparametric implementation of Bayes filter [16]. In Bayes filter algorithm, we calculate the belief distribution from measurement and control data:

$$\begin{aligned} p(x_t|z_{1:t}, u_{1:t}) &\propto p(z_t|x_t, z_{1:t-1}, u_{1:t}) \cdot p(x_t|z_{1:t-1}, u_{1:t}) \\ &= p(z_t|x_t) \cdot \int p(x_t|x_{1:t-1}, z_{1:t-1}, u_{1:t}) \cdot p(x_{t-1}|z+1:t-1, u_{1:t-1}) dx_{t-1} \\ &= p(z_t|x_t) \cdot \int p(x_t|x_{t-1}, u_t) \cdot p(x_{t-1}|z_{1:t-1}, u_{1:t-1}) dx_{t-1} \end{aligned} \tag{9}$$

As denoted in the equation above, the main purpose for a Bayes filter is to calculate a posterior probability. In this equation, x is the state to estimate at time t, z is the measurement and u is the control. With Bayes' theorem, the posterior probability can be divided into two parts. $p(z_t|x_t)$ is the likelihood probability with measurement at time t. $p(x_{t-1}|z_{1:t-1}, u_{1:t-1})$ is the posterior probability at time $t-1$, so the integration in the second term can be regarded as the prediction of state at time t with the posterior probability at time $t-1$ and the control at time t.

Table 1 depicts the basic Bayes filter in pseudo-algorithmic form. Bayes filter is recursive, the belief $bel(x_t)$ at time t is calculated from the belief $bel(x_{t-1})$ at time $t-1$. For each step, the input is the belief $bel(x_{t-1})$ and the the control u_t, the measurement z_t at time t. Its output is the belief $bel(x_t)$.

In Line 3, the Bayes filter predicts the belief $\overline{bel}(x_t)$ at time t based on prior state $bel(x_{t-1})$ and the control u_t. This part is called the time update. Then, in Line 4, the prediction $\overline{bel}(x_t)$

```
1  Algorithm Bayes Filter(bel(x_{t-1}), u_t, z_t):
2  for all x_t do
3      overline{bel}(x_t) = ∫ p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx
4      bel(x_t) = η p(z_t|x_t) overline(x_t)
5  end for
6  return bel(x_t)
```

Table 1: Pseudo-code for Bayes filter

from time update part is regarded as a prior information. Multiplying by the likelihood probability from observation $p(z_t|x_t)$, we can finally obtain its posterior probability, which is denoted as belief $bel(x_t)$ at time t. This step is called measurement update.

When it refers to a linear Gaussian system, the Bayes filter can be easily implemented as a Kalman filter, or extended Kalman filter(EKF) for nonlinear system. However, these filters rely on a fixed functional form of the posterior, such as Gaussian. A popular alternative to these techniques are nonparametric filters, such as the particle filter. Instead of requiring a fixed function, they approximate posteriors by a finite number of values, each roughly corresponding to a region in state space [16].

The key idea of particle filter is to represent the posterior $bel(x_t)$ by a set of random state samples drawn from the posterior. In particle filter, the samples of a posterior distribution are called *particles* and are denoted as:

$$\chi_t := x_t^{[1]}, x_t^{[2]}, ..., x_t^{[N]} \tag{10}$$

Each particle $x_t^{[n]} (1 \leq n \leq N)$ represents a concrete instantiation of the state at time t, and N denotes the number of particles.

```
1  Algorithm Particle Filter(χ_{t-1}, u_t, z_t):
2  χ̄_t = χ_t = ∅
3  for i = 1 to N do
4      sample x_t^[i] ~ p(x_t|u_t, x_{t-1}^[i])
5      w_t^[i] ~ p(z_t|x_t^[i])
6      χ̄_t = χ̄_t + < x_t^[i], w_t^[i] >
7  end for
8  for i = 1 to N do
9          draw x_t^[i] with probability ∝ w_t^[i]
10     add x_t^[i] to χ_t
11 end for
12 return χ_t
```

Table 2: Pseudo-code for Particle filter

In the table 2, we demonstrate the pseudo-code for particle filter. The particle filter is composed of two part. In the first part, Line 3 - Line 7, it aims to generate a hypothetical state $x_t^{[m]}$ for time t based on particles $x_{t-1}^{[m]}$ and the control u_t and calculate an importance factor for each particle based on probability of the measurement z_t under particle.

The second part for particle filter is resampling. In this part, particle filter draws with replacement N particles from the prediction set $\bar{\chi}_t$. By introducing the importance factor, resampling force particles back to the distribution with posterior probability.

The intuition behind particle filter is to approximate the belief $bel(x_t)$ by the set of particles χ_t. Ideally, the distribution of x_t is the same as the Bayes filter posterior:

$$x_t^{[i]} \sim p(x_t|z_{1:t}, u_{1:t}) \tag{11}$$

In practice, this property can only be valid for $N \to \infty$. That means, for finite particle number, their distribution will be slightly different from the truth. However, this difference is negligible

when the number of particles is not too small(e.g. $N \geq 100$) [16].

3.3.2 Properties of the particle filter

Particle filter is an approximation to practical distribution of belief $bel(x_t)$, so there essentially exists approximation errors. There are four complimentary sources of approximation error. each of which will give rise to improved version of the particle filter [16].

A. Finite particles

The first approximation error corresponds to the finite number of particles. As we have pointed out, only infinity particles can completely produce the posterior distribution, finite particles will introduce a systematic bias in posterior estimate. Consider the extreme condition that $N = 1$, the loop in Line 3 through Line 7 in table 2 will only be executed once, the particle distribution $\bar{\chi}_t$ contains a single particle. As we will always apply a normalization among importance factor $w_t^{[i]}$, the measurement $p(z_t|x_t^{[m]})$ plays no role in the result of update. Thus, if $M = 1$, the particle filter generates particles from the probability:

$$p(x|u_{1:t}) \tag{12}$$

instead of the desired posterior probability $p(x|u_{1:t}, z_{1:t})$, it ignores all the measurement. Therefore, in practical implementation, we should use a large value of N in order to reduce the effect of dimensionality loss.

B. Randomness introduced in resampling

In particle filter, the resampling process will induces a loss of diversity in the particle population, which in fact manifests itself as approximation error. Even though the variance of the particles decrease, the variance of estimator of the true belief will increase. To understand this error, we again consider an extreme case, which is that a robot whose state does not change:

$$x_t = x_{t-1} \tag{13}$$

In vehicle ego movement estimation, the state represents the localization. When it has been stopped, we can furthermore assume the vehicle process no sensors, hence it fails to estimate the state, instead it will generate N identical particles. In practice, there exist two solution for tis variance reduction. One is to reduce the frequency of resampling, the other is to use a low variance samling.

For the first solution, when the state is known to be static $x_t = x_{t-1}$, we should suspend resampling process. Even if the state changes, it is a good idea to reduce the frequency of resampling. More specifically, it maintains the importance factor and update as follows:

$$w_t^{[i]} = \begin{cases} 1, & resampling\ take\ place \\ p(z_t|x_t^{[i]})w_{t-1}^{[i]}, & no\ resampling\ take\ place \end{cases} \tag{14}$$

However, the choice of whether to resample requires practical experience, a standard criteria is the variance of importance weight.

Table 3 indicates the pseudo-code for low variance sampling, The basic idea is to use a sequential selection of particles instead of independently selecting. This algorithm calls random generation function only once at the beginning, and selects samples with a probability proportional to importance weight with the random number. In reality, during the generation of particles, this algorithm copies the particles with large importance weight and drop out the particles with small weight. Obviously, the number of copies is proportional to its weight.

There are three advantages for this algorithm. First, it covers the space of samples in a more systematic fashion than the independent random sampler. Second, if all the samples have the same importance weight, the sample will not change after resampling process ($\bar{\chi}_t = \chi_t$), so that there are no lost information if no observation has been integrated into measurement update. Third, this algorithm is a sequential selection, which means a $O(N)$ complexity, while the classical resampling algorithm has a $O(NlogN)$ complexity (a $O(logN)$ search for each particle once a random number has been drawn, and we have N particles).

```
1  Algorithm Low variance sampling(χ_t, W_t):
2  χ̄_t = ∅
3  r = rand(0, M^{-1})
4  c = w_t^[i]
5  i = 1
6  for n = 1 to N do:
7          u = r = (n − 1)M^{-1}
8      while u > c
9          i ++
10         c += w_t^[i]
11     add x_t^[i] to χ̄_t
```

Table 3: Pseudo-code for Low variance sampling

C. Divergence of the proposal and target distribution

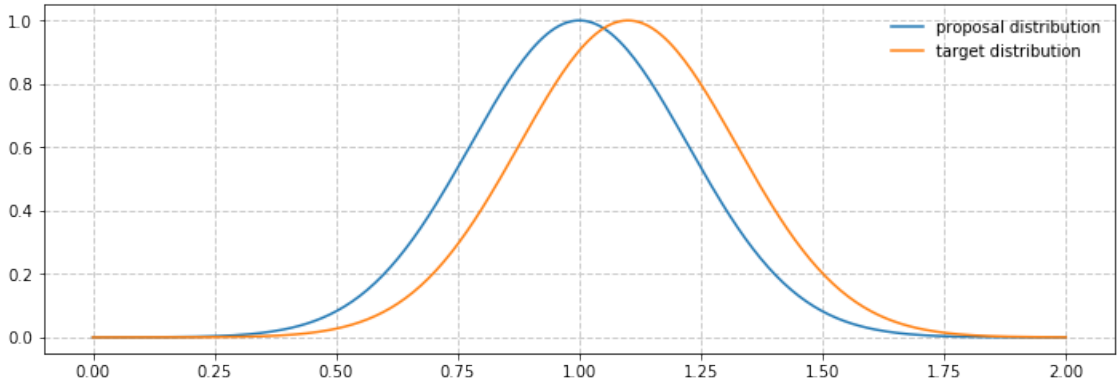
In particle filter, as shown in table 2, Line 4 generate a proposal distribution g of samples, and the posterior distribution is regarded as target distribution f, so we can simply calculate the importance weight:

$$w^{[i]} = \frac{f(x^{[i]})}{g(f(x^{[i]})} \tag{15}$$

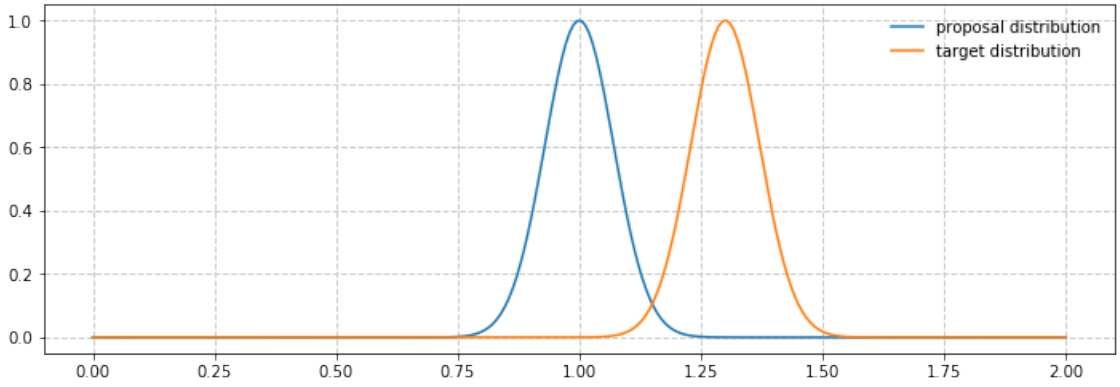
It is easy to see that the efficiency of the particle filter relies crucially on the 'match' between the proposal and the target distribution. One can image a condition, when the sensors of robot are highly inaccurate but its motion is very accurate, as demonstrated in Figure 11(a), the target distribution will be similar to the proposal distribution and the particle filter will be efficient. However, in the contrary, when the sensors are highly accurate and its motion is very inaccurate, in Figure 11(b), these distributions can deviate substantially and the resulting particle filter can become arbitrarily inefficient. An extreme example is the sensors are deterministic, for these sensors the likelihood probability for observation $p(z|x)$ becomes a pulse function, and $p(z_t|x_t^{[i]})$ will be zero for most of particles, with expectation of those that match the measurement z exactly. Such a situation can be fatal: the proposal distribution will practically never generate a sample x which exactly match the measurement z. Thus, all the importance weights will be zero and resampling precess becomes ill-condition. Consequently, we often assume a higher noise level of sensors in practice.

D. Particle deprivation problem

When we perform the particle filter in a high-dimensional space, there may be no particles in the vicinity to the correct state. This could because the limitation of particle numbers, but it might happen in any particle filter as the probability is always larger than zero. Thus this condition can not be ignored in a long-term estimation problem. In order to reduce this effect, we could in practice add a small number of randomly generated particles into the samples after each resampling process.



(a) Accurate motion with inaccurate sensor



(b) Accurate sensor with inaccurate motion

Figure 11: Proposal distribution and target distribution

3.3.3 Implementation in practice

As we know, the vehicle ego-motion state is a 6 DOF(degree of freedom) vector: the rotation(yaw, pitch, roll) and the translation(X, Y, Z). However, in practical implementation, due to the vehicle movement characteristics, the pitch, roll and Z will be quasi-unchanged. So, in general, we generate a vehicle kinematic model with on-board sensor inputs [10] :

$$x_k = \begin{bmatrix} X_k \\ Y_k \\ \psi_k \end{bmatrix} \tag{16}$$

where (X, Y) is position in global coordinates and ψ is vehicle heading, subscript k denotes the kth step. Our main task is to estimate the vehicle state at each time t

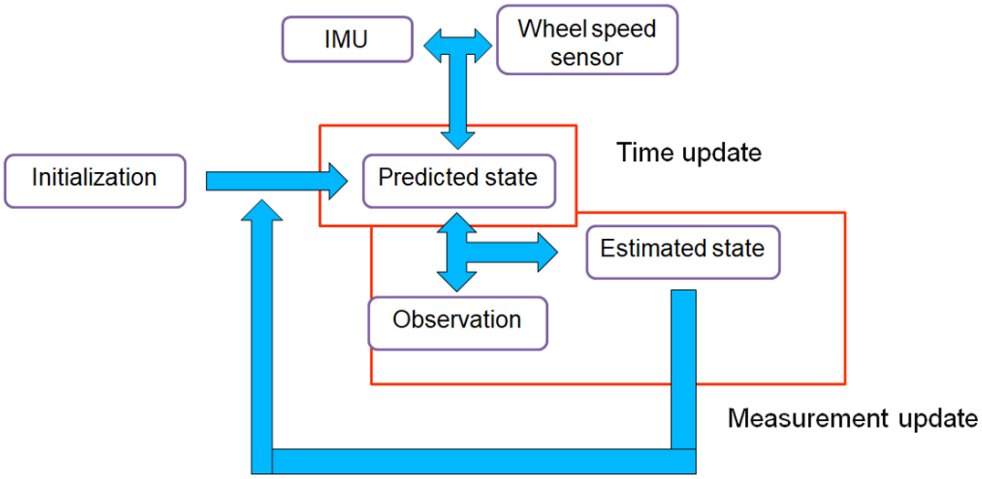


Figure 12: Particle filter frame

Figure 12 denotes the overall process of the particle filter based localization algorithm. We initialize the system with the GPS information, then we execute recursively the time update step and the measurement update step. For time update, we read information from IMU and wheel speed sensor and give a prediction for current state. Then, with the observation, we will finally obtain an estimated state in measurement update step. More precisely, the pipeline is shown below:

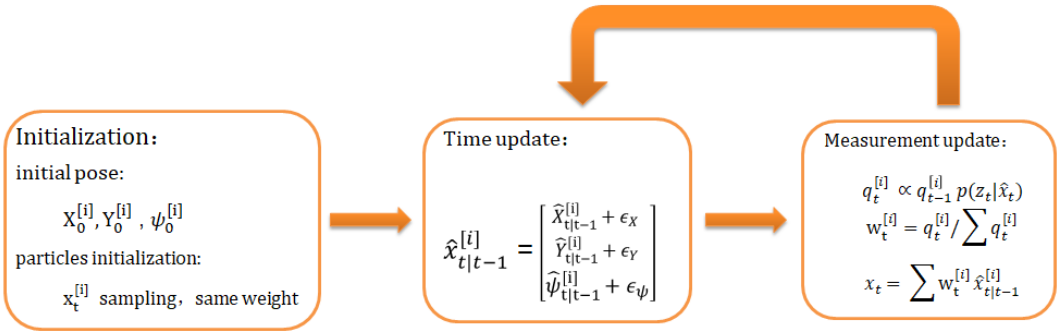


Figure 13: Particle filter pipeline

In our system, we create a class *Particle*, each particle has three main elements, the pose state X, Y, ψ, the particle probability q and its weight w.

In initialization step, we read the initial pose state from GPS, which is denoted as X_0, Y_0, ψ_0. Based on this initial state, we sample randomly for each particle and set the same weight:

$$\begin{cases} X_0^{[i]} = X_0 + m_x \\ Y_0^{[i]} = Y_0 + m_y \\ \psi_0^{[i]} = \psi_0 + m_\psi \\ w_0^{[0]} = 1/n \end{cases} \tag{17}$$

where m_x, m_y, m_ψ are random value with pre-set range and n is the number of particles. In order to reduce the approximation error cased by finite particles(see Section 3.3.2 A), we normally set $n = 200$ (sometimes we use $n = 100$ to decrease computational consumption).

Then, we enter into the time update step. We read the angular velocity from IMU and the relative motion from wheel speed sensor, hence we can calculate a relative move $T_{t|t-1}$ and get a

prediction of state:

$$\hat{x}_{t|t-1}^{[i]} = T_{t|t-1} x_{t-1}^{[i]} \tag{18}$$

Based on the predicted state, we update all the particles' pose state by adding a random noise on the prediction:

$$\hat{x}_{t|t-1}^{[i]} = \begin{cases} \hat{X}_{t|t-1}^{[i]} + \epsilon_X \\ \hat{Y}_{t|t-1}^{[i]} + \epsilon_Y \\ \hat{\epsilon}_{t|t-1}^{[i]} + \epsilon_\psi \end{cases} \tag{19}$$

$\epsilon_X, \epsilon_Y, \epsilon_\psi$ are random noise on position and heading. As discussed in Section 3.3.2 C, we always consider a higher level of noise avoid the divergence of the proposal and target distribution.

Finally, with observation, we could carry out the measurement update. The main task in this step is to calculate the likelihood probability of measurement $p(z_t|\hat{x}_t)$ with the lane marker match, the methodology will be explained in the next section. Here we suppose to have already obtained its probability. The we can update the probability for each particle:

$$q_t^{[i]} = q_{t-1}^{[i]} p(z_t|\hat{x}_t^{[i]}) \tag{20}$$

After normalization, we obtain the particles' weights:

$$w_t^{[i]} = q_t^{[i]} / \sum q_t^{[i]} \hat{x}_{t|t-1}^{[i]} \tag{21}$$

then, we can obtain the final estimated state by calculating a weighted average:

$$x_t = \sum w_t^{[i]} \hat{x}_{t|t-1}^{[i]} \tag{22}$$

normally, we delete the particles with low weight in advance.

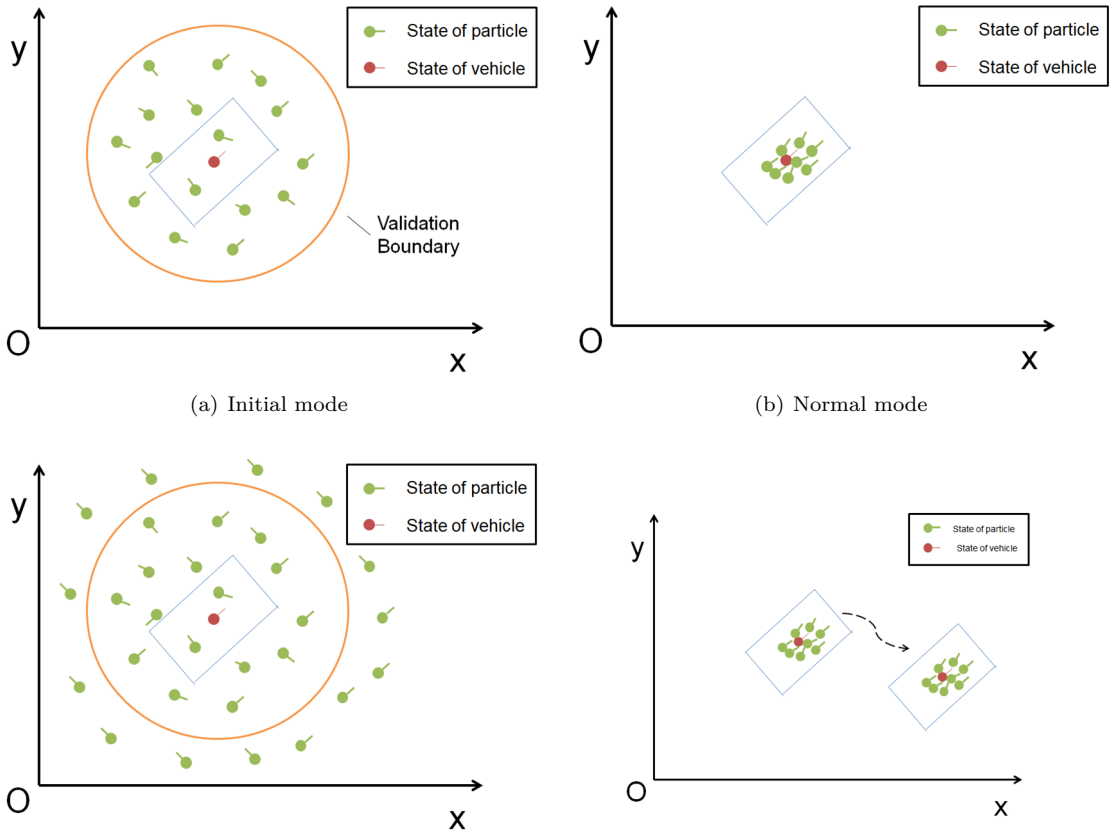
Although the pipeline in Figure 13 can handle most of condition in practical implementation, however, we should take consideration of lone-term estimation and the extreme cases. Therefore, to establish a robust system, we introduce the resampling and set four working modes for particle filter.

A. Resampling

Resampling regenerates the new set of randomly generated particles on the basis of their relative likelihoods. Resampling prevents the concentration of the probability mass on only a few particles. As discussed in Section 3.3.2 B, the resampling will be executed with certain condition:

$$\hat{N}_{eff} = \frac{1}{\sum_{i=1}^{N} (w_t^{[i]})^2} < 0.6N \tag{23}$$

N_{eff} represents the effective number of samples that indicate the degree of depletion. It has the maximum value ($\hat{N}_{eff} = N$) when all the particles have the same weight. In contrast, it has the minimum value ($\hat{N}_{eff} = 1$) when the weights have been denoted to a single particle. Therefore we choose an upper threshold of N_{eff} as $0.6N$. To reduce the randomness introduced in resampling, we choose a low-variance sampling. As the estimation state has only 3 DOF, the particle deprivation problem have not been considered in our algorithm.



(a) Initial mode

(b) Normal mode

(c) Unstable mode

(d) Tracking mode

Figure 14: Work modes for particle filter

B. Initial work mode

As shown in Figure 14(a), the initial work mode will be executed in the initialization or re-initialization. In this work mode, we read the GPS information and give an initial pose state. In the figure, the red circle with a bar represents the true vehicle pose state, while the green one represents the pose state of particle (the position of circle indicates the vehicle position X, Y and the direction of bar indicates the heading ψ). With a prior information of GPS, a validation boundary is set to limit the range of random sampling:

$$p^{[i]}_{sampling}(\hat{X}^i_0, \hat{Y}^i_0 | X_{GPS}, Y_{GPS}) = \begin{cases} 1/\pi\sigma^2_{GPS}, \ for \ (\hat{X}^i_0 - X_{GPS})^2 + (\hat{Y}^i_0 - Y_{GPS})^2 < \sigma^2_{GPS} \\ 0, \ otherwise \end{cases} \tag{24}$$

meanwhile, in initial work mode, each particle will be given the same weight.

C. Normal work mode

Normal work mode, in Figure 14(b), is the most common mode in our algorithm, it will execute the time update and measurement update which have been discussed above. In this mode, we can see that all the particles surround the true pose state.

D. Unstable work mode

During the pose estimation, some times the system will suffer from a low confidential situation. It can be caused by non-convergence of particles or lack of enough observation. In this condition, the valid boundary from GPS position will be set to delete the particles out of range, seen in Figure 14(c). If the particles turn to converge, the work mode will change to normal work mode,

otherwise, the work mode will change to tracking mode if unstable mode have been continued for a long time.

E. Tracking work mode

In this work mode, as shown in Figure 14(d), the observation is considered as invalid. All the particles will be updated with prediction and contain their weights:

$$\begin{cases} x_t^{[i]} = T_{t|t-1} x_{t-1}^{[i]} \\ w_t^{[i]} = w_{t-1}^{[i]} \end{cases} \tag{25}$$

If the duration of tracking work mode has been out of our pre-set threshold, the system will through out a bad case and turn to re-initialization.

3.4 Lane marker match

As discussed above, the likelihood probability of measurement $p(z_t|\hat{x}_t^{[i]})$ is based on lane marker match. In this section, we will explain in precise its methodology. In general, the lane marker match is composed of 3 parts, image pre-process, local map extraction and reprojection based line match.

A. Image pre-process

In Figure 8, we have demonstrated the system overview, the input data for each frame is a raw image, like the figure shown below:

Figure 15: Raw image

This is a grayscale image with 8 bits format (the grayscale value is from 0 to 255). The localization system is based on feature-level filter, which means a pre-process for these images is necessary. In our system, we implement a semantic segmentation method based on deep learning (this work has been finished by my colleagues). The parse image is shown in Figure 16:

Figure 16: Parse image

In the parse image, it has been separated into foreground (the land markers) and the background (the black areas in the image). Meanwhile, the foreground pixels are classified into different attributes (white solid line, stop line and pole-like line). Then, a probabilistic hough transformation is introduced to extract lines and to create the line features. Each line feature is compose of the position of endpoints and its attribute.

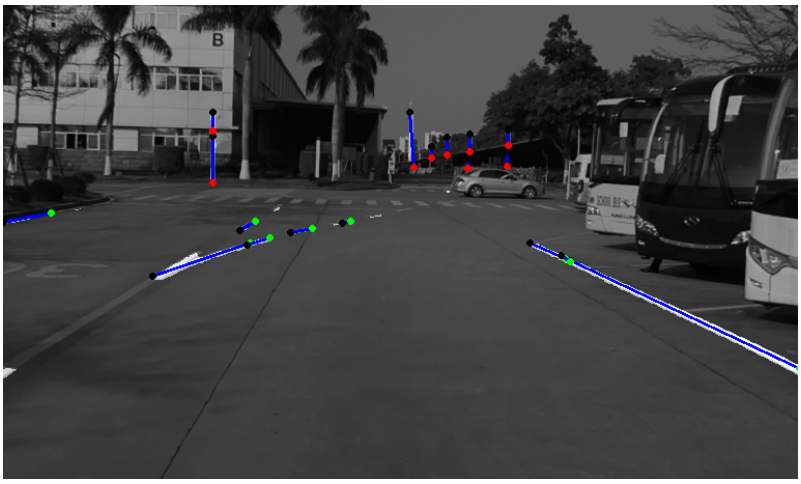


Figure 17: Merged image

Figure 17 denotes a merged image of raw image and its line features. The green points and red points are location of starting points with different attributes, while the black points are location of end points. The blue lines represent a visualization of line features. We should realize that these lines are not included in line features, only location of endpoints and attributes are included.

B. Local map extraction

In the localization system, the maps are pre-built and are divided in small pieces with global coordinates. Figure 18 is a visualization of a piece of local map. Practically, the map are stored in the form of line features (location of endpoints and attributes).

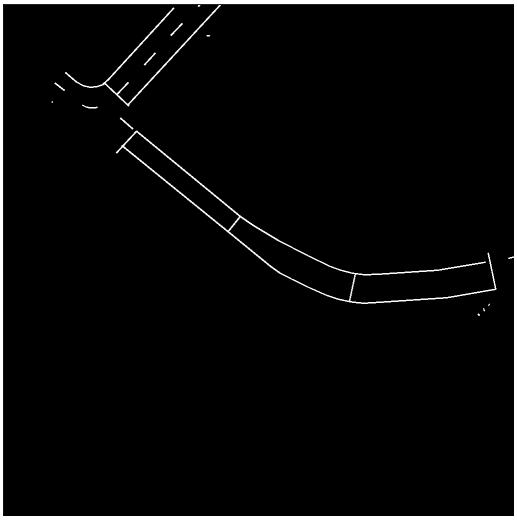


Figure 18: Local map

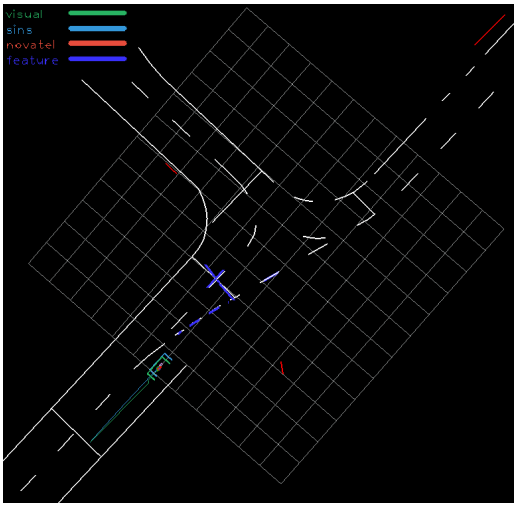


Figure 19: ROI of local map

For each frame, the predicted pose state could be obtained in advance, therefore the local map is extracted based on the prediction and a pre-set ROI(region of interest). Figure 19 denotes the ROI for a front camera. In general, the ROI is compose of parts of several local map, so the clipping and stitching are introduced in map extraction.

C. Reprojection based line match

As we know, each particle represent a pose state. Based on its pose state, we can re-project the line features in local map to the coordinate of camera and calculate the likelihood probability of measurement with line match.

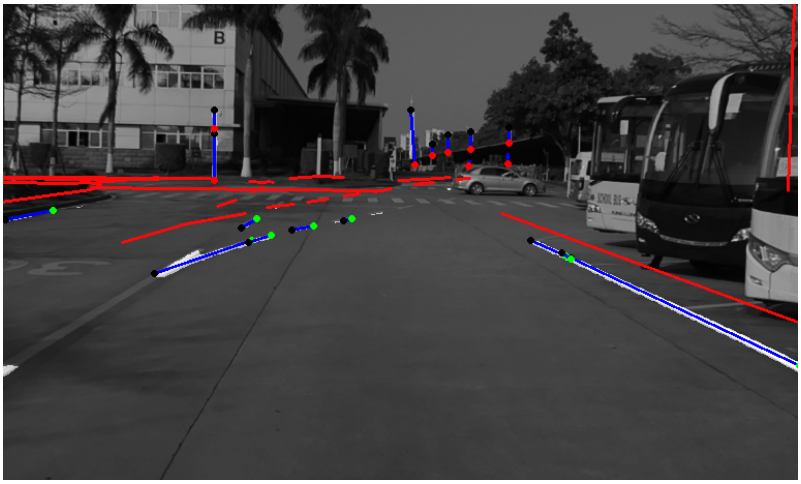


Figure 20: Reprojection with low likelihood probability

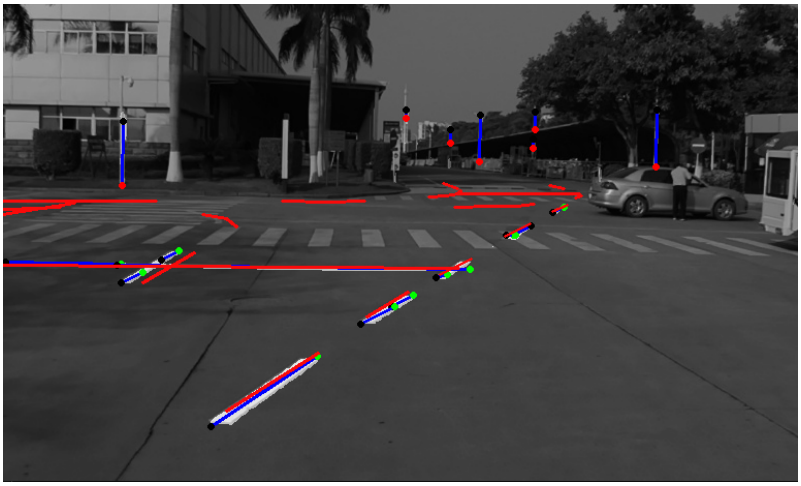


Figure 21: Reprojection with high likelihood probability

Figure 20, 21 denote the two case of reprojection. In human perception, we could easily make the conclusion for which has the higher likelihood probability. However, in computational logic, an exact formulation should be defined. A simple method is to use the line projection error in Section 2.2 C. However, in land marker based localization, since most of land markers are white solid lines which have the longitudinal direction, if we only consider the line distance, this criteria will have a strong constraint on latitude localization error but a week constraint on longitudinal localization error and heading error. So, more constraints should be added:

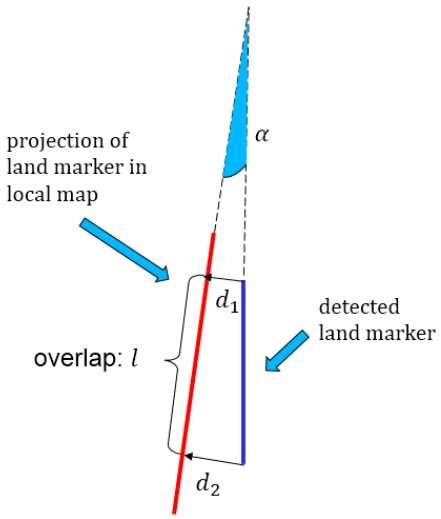


Figure 22: Reprojection based line match method

As shown in Figure 22, in order to increase the constraint in longitudinal and heading, we propose a more strict match error function:

$$e = f(d_1, d_2, \alpha, l) \tag{26}$$

where d_1, d_2 are line distance, α denotes the angle between two lines and l represents the overlap. Instead of feature association, we choose a brute search by traversing all the lines and accept all the match under certain condition (e.x. α is less than a threshold). This means a detected line can match several lines in local map, and vice versa. This is because the lines are always cracked due to original map generation, local map clipping and stitching, curve in hough transformation, etc. The final likelihood probability depends on the sum of match error with accepted line match:

$$p(z_t|\hat{x}_t^{[i]}) \propto 1/\sum e_i \tag{27}$$

4 Evaluation and Optimization

4.1 Evaluation for practical data

Our team is a part of Intelligent Driving Group(IDG, Baidu). Our group aim to develop the real L4 level autonomous vehicle capable of sensing and navigating without human input. Meanwhile, we plan to achieve the mass production for civil use. So instead of evaluating in public data set, we build our own motorcade for data collection and algorithm evaluation using the modified Lincoln MKZ, seen in the figure below:

Figure 23: Modified vehicle

This modified vehicle is equipped with cameras in different orientation, an inertial measurement device(IMU), wheel speed sensors, GPS, radars and LiDARs. These sensors are widely used in self-driving technology development company.

Figure 24: NovAtel system

Besides, in order to evaluate our localization prevision, we also equip a high-cost GNSS-INS system which is produced by NovAtel. This system can provide a high precise estimation of vehicle pose, rotation and translation. However, it is too expensive to have a practical implementation in autonomous driving. The Figure 24 in the left is a sketch for it. We choose the road section where the NovAtel system has a good output to test our algorithm. Consider that in real driving condition, the most important for vehicle localization is its lateral error, longitudinal error and its heading error, so in the following off-line test, we take these three criteria to evaluate our system performance.

In the previous section, we have explained the back end of our localization system, especially the theory of particle filter, and discussed about its improved version. In this section, we firstly use a front camera to provide the raw images and compare its localization precision by reference to NovAtel. For the convenience of debugging and optimization, we collect all the sensors' data in advance and take the off-line evaluation.

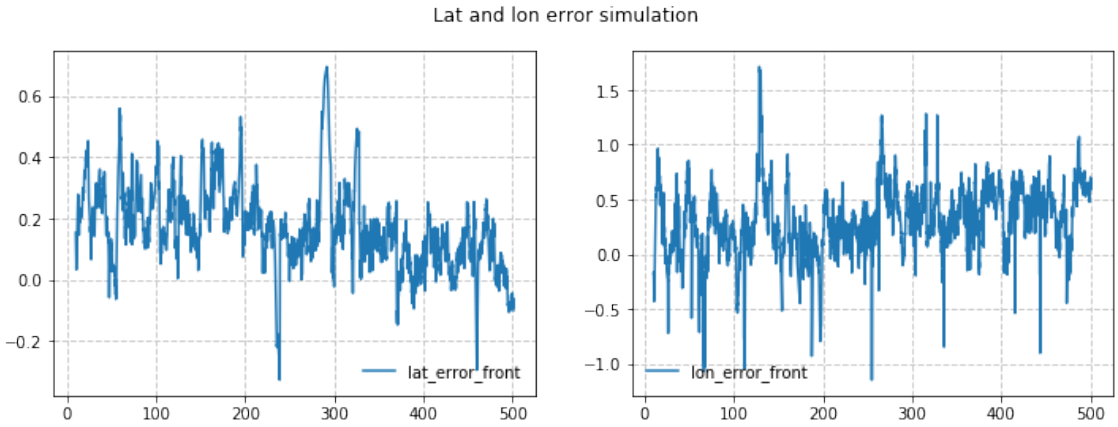


Figure 25: Lateral error and longitudinal error, use front camera

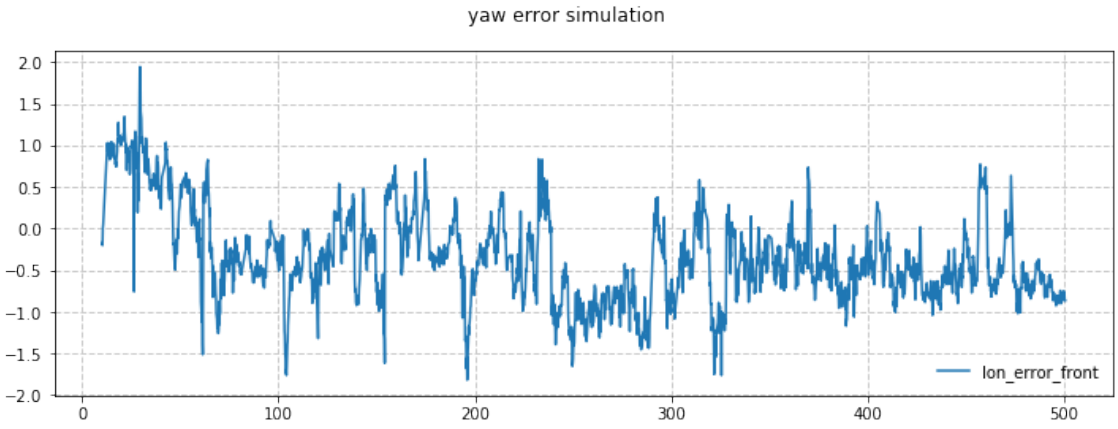


Figure 26: Heading error(yaw), use front camera

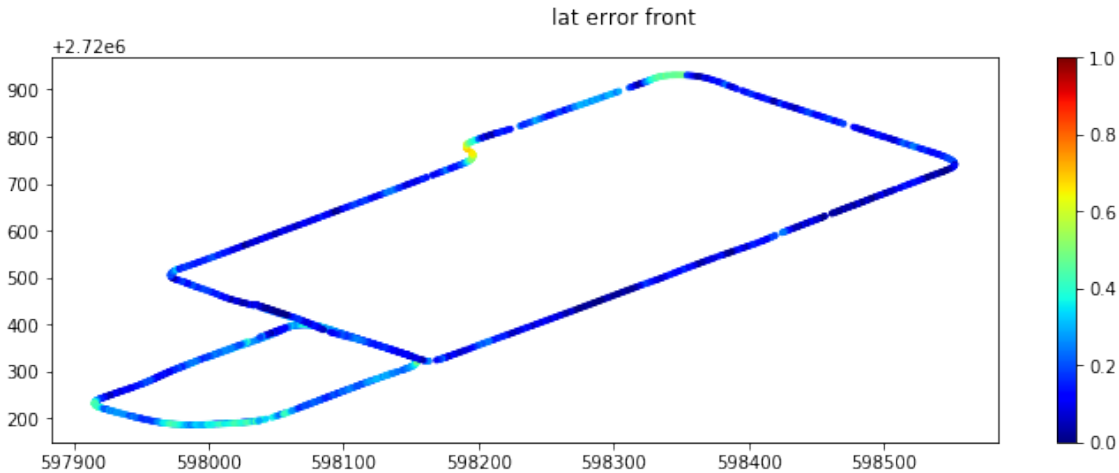


Figure 27: Lateral error in global coordinates, use front camera

Figure 25, 26 and 27 indicate the localization error for front camera based visual localization. More precisely, the statistical results is shown in Table 4, $e_{0.8}$ indicates that 80% of frames whose localization error is less than $e_{0.8}$ (the same for $e_{0.9}$ and $e_{0.99}$), this parameter can well evaluate the robustness for localization system . We can see that the mean lateral error is approximately $17cm$ with a variance of $11cm$, while the mean longitudinal error is approximately $36cm$ with a variance of $25cm$. The longitudinal error is much higher than the lateral error because most of landmarks are longitudinal, so that the constant in lateral is more stricter than that in longitudinal.

Seen in Figure 25, we could find that the lateral error and longitudinal error are positive in most cases, this might be caused by a calibration error. Since the reference coordinate systems of sensors and vehicle are different, we normally calibrate all the sensors and transform the motion data in vehicle coordinate system. In our algorithm, we suppose that this calibration is the truth. Unfortunately, this assumption can not be guaranteed due to the calibration precision and looseness of sensors. Therefore, we believe that there exists a constant deviation in error count.

Consider that the lateral localization precision is the most important for a autonomous vehicle, we draw its lateral error in global coordinates in Figure 27. We found that in most cases, the lateral error is under $20cm$, however, this error significantly increase in several corners, this problem and its solution will be discussed in the next section.

	mean	std	$e_{0.8}$	$e_{0.9}$	$e_{0.99}$
lateral error(m)	0.1760	0.1190	0.2574	0.3399	0.5580
longitudinal error(m)	0.3669	0.2514	0.5817	0.6737	1.1272
yaw error(deg)	0.5587	0.3434	0.8433	1.0259	1.4543

Table 4: Localization error, front camera

4.2 Dual cameras enhancement

In the former system, the lateral localization precision decrease in the corners. This is because the raw images are obtained by a front camera. Although it is a common solution in most localization problem for robotics. However, for autonomous vehicle, as discussed, the pose estimation is based on the lane marker match. Figure 28 demonstrate an image from front camera when the vehicle is around the corner. In this case, the land markers extracted from raw image are not enough to enable the measurement update. As discussed in Section 3.3.3, the work mode will turn to **Unstable work mode** and soon change to **Tracking work mode**. Without observation, the localization result exclusively depends on the predicted pose provided by IMU and wheel speed sensors. This prediction will be highly suffered from the drift and initial heading when the **Tracking work mode** starts.

Figure 28: Image from front camera

In the contrary, there are enough land markers in the image from a rear camera at the same position, as shown in 29:

Figure 29: Image from rear camera

For each frame, we execute the land marker match in the camera's coordinate system, which means we could easily add the rear camera in our system with simply saving the extrinsic parameters of rear camera and a rear oriented ROI. This method can be extended to a multi-camera based localization system by holding all the extrinsic parameters and different ROI. We compare the results between the front camera based localization system and the dual cameras based localization system:

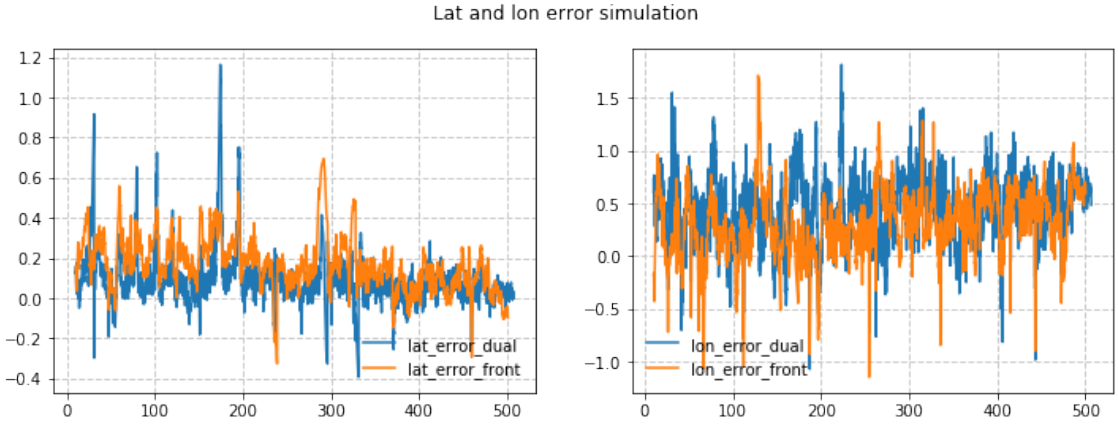


Figure 30: Lateral error and longitudinal error, front camera(orange) and dual camera(blue)

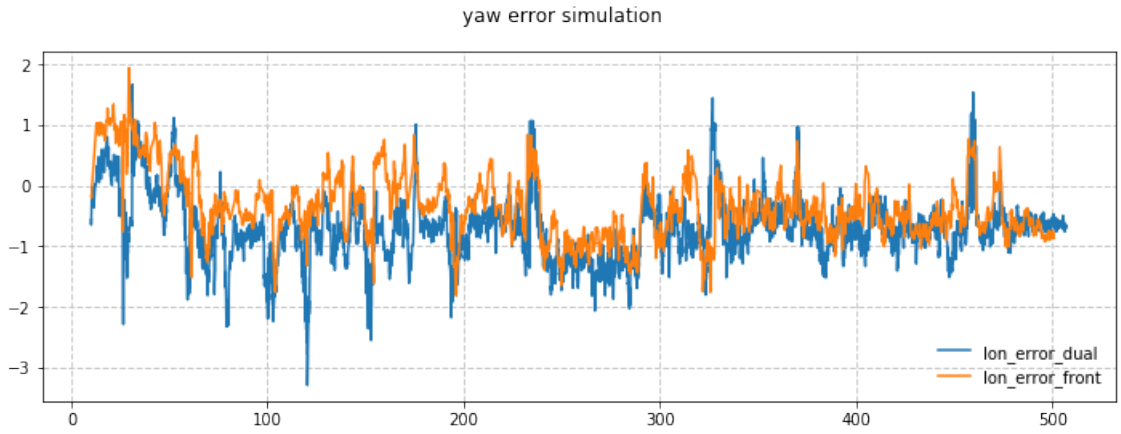


Figure 31: Heading error(yaw), front camera(orange) and dual camera(blue)

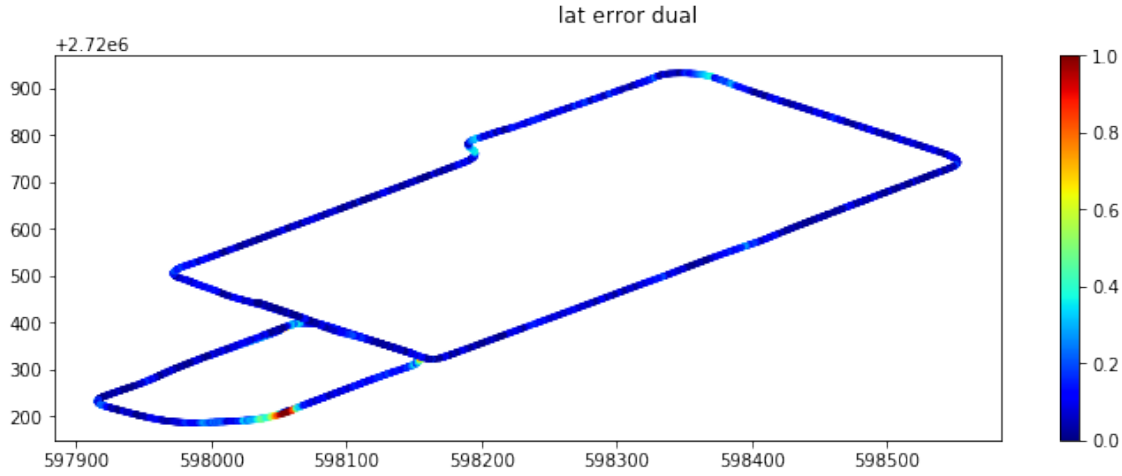


Figure 32: Lateral error in global coordinates, use front camera

We can see that the lateral localization precision has been increased by using dual camera system, especially in the corners, the statistical result in Table 5 can also prove it. The mean lateral error has been decreased to $10cm$ and the robustness has been increased (it has a lower $e_{0.8}$ and $e_{0.9}$). The $e_{0.99}$ has been increased because in the lower right corner of the small circle, shown in Figure 32, the localization error has significantly increased. In this section of road, we find that there exits the frame loss for front camera, the localization system is lead by rear camera. Table 6 denotes the statistical result for a single rear camera based localization system. We find that this system is much worse than the system based on front camera. It is caused by the distortion in rear camera. Meanwhile, this distortion result in the worse localization precision in longitudinal and heading.

	mean	std	$e_{0.8}$	$e_{0.9}$	$e_{0.99}$
lateral error(m)	0.1029	0.1134	0.1414	0.2007	0.6392
longitudinal error(m)	0.4672	0.2701	0.6972	0.8077	1.1724
yaw error(deg)	0.7819	0.4088	1.0980	1.3392	1.9600

Table 5: Localization error, dual camera

	mean	std	$e_{0.8}$	$e_{0.9}$	$e_{0.99}$
lateral error(m)	0.2396	0.5922	0.1958	0.2909	3.3287
longitudinal error(m)	0.5695	0.4334	0.7835	0.9766	2.5113
yaw error(deg)	0.6934	0.7150	0.9753	1.3017	5.2474

Table 6: Localization error, rear camera

Difficulties for dual cameras enhancement

Although the localization system based on dual cameras has a higher precision in lateral, the distortion in rear camera limits the potential improvement. This rear camera equipped on our vehicle has a wide-angle lens, which means a high level of distortion, Figure 33 demonstrates a raw image from rear camera. We could easily find the deformation of buildings and trees in the red boxes. Figure 34 is the reprojection of land markers from local map with pose in groundtruth. The distortion in rear camera will lead to a dismatch between detected lines and map lines, even with pose in groundtruth.

Figure 33: Distortion in rear camera

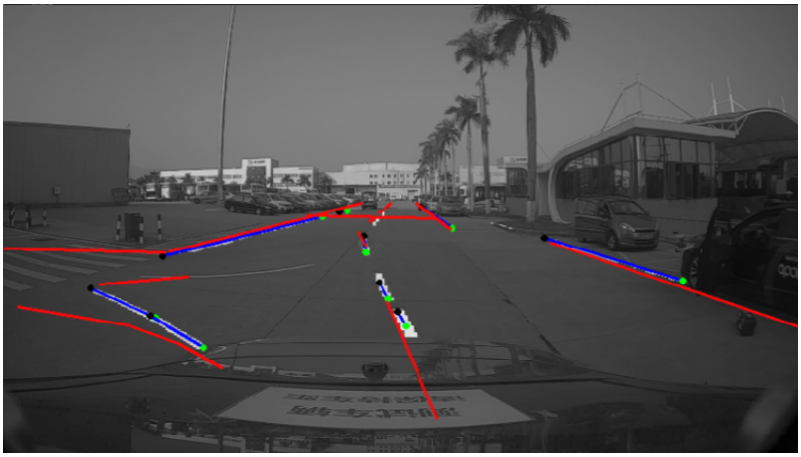


Figure 34: Effect of distortion

The most common distortion in camera is the radial distortion, it can be denoted as below:

$$\begin{cases} D_x = x_d(k_1 r^2 + k_2 r^4 + k_3 r^6 + \cdots) \\ D_y = y_d(k_1 r^2 + k_2 r^4 + k_3 r^6 + \cdots) \end{cases} \tag{28}$$

where D_x, D_y are displacement caused by distortion and x_d, y_d are distance between distorted image points and distortion center. For the classical camera, we just calibrate the first two terms

k_1 and k_2, and our calibration team is the same. However, since the rear camera has the wide-angle lens, it needs to calibrate the first three terms k_1, k_2 and k_3. Unfortunately, due to project progress arrangement, the requirement for re-calibration can not be finished for instance. But we believe that the dual cameras enhancement (or multi-camera enhancement) would still give rise to system performance with re-calibration.

4.3 Lane marker match in stereo space

In our testing vehicle, we also equip a stereo vision system, which could provide a raw image in the left camera and a relative disparity image. In this condition, we could realize the localization algorithm with lane marker match in stereo space. As discussed in Section 3.4, the longitudinal constraint depends on the stop lines and the overlap between matched lines. However, the stop lines are not always appeared and the constraint from overlap is not strong enough. If we match the lane markers in stereo space, it would provide a higher precision. This work has been finished and proved its potentiality, but there still exists the problem to be solved. The evaluation and inadequacies will be discussed in the following (Section 4.3.3).

4.3.1 Stereo vision

Stereo vision is a technology which use images from cameras with different perspective to obtain the 3D position of points [4].

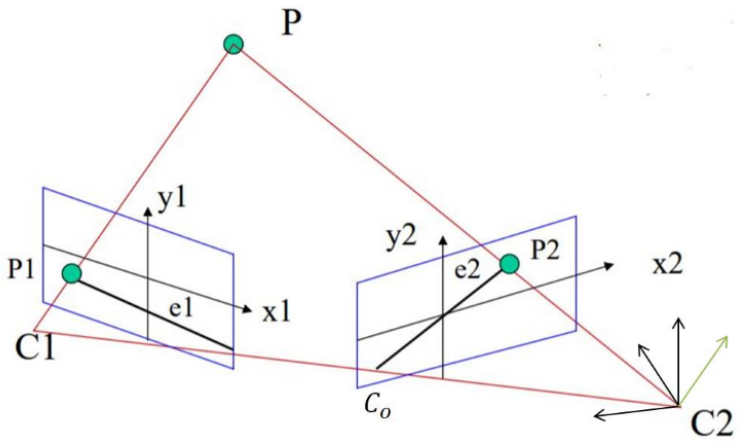


Figure 35: Stereo vision

Figure 35 is the schematic for two-view stereo vision system. C_1 and C_2 are optical centers of cameras. Their position and orientation are supposed to be known. For a 3D point P, its projection on the images are P_1 and P_2, once we match the two point in these two images, we could restore the 3D position of P using triangulation. More precisely, in the Figure 36, we transform the two camera coordinate system in parallel. The triangulation could be simplified to a similar triangle problem. If we define the disparity:

$$d = x_l - x_r \tag{29}$$

the depth z is:

$$z = \frac{f \cdot b}{d} \tag{30}$$

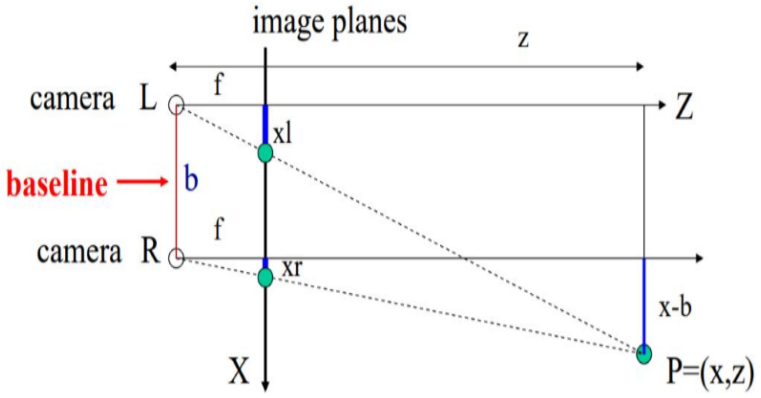


Figure 36: From disparity to depth

4.3.2 System overview

Figure 37 demonstrate the lane markers match in stereo space. C is the coordinate system of vehicle. Since we have the raw image and the disparity image, we could choose the pixels which are marked as land markers and extract their disparity, known the extrinsic parameters of stereo systems, we could project all these pixels in camera coordinate system and transform them in wold coordinate system with vehicle pose. After that, we could calculate the measurement likelihood probability $p(z_t|\hat{x}_t)$ by matching detected lines with map lines in 3D space.

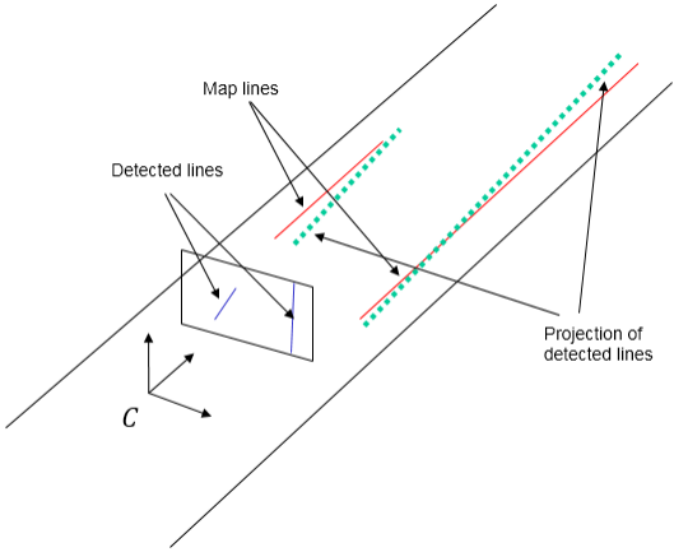
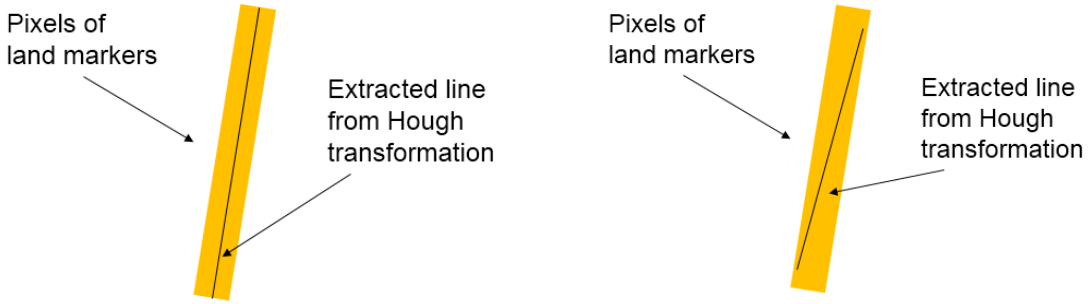


Figure 37: Stereo match

Unlike the 2D match, we project pixels of land markers rather than lines extracted from Hough transformation. This is because the Hough transformation could not fit well with the real land markers, especially when the land markers are projections from 3D to 2D. Figure 38 shows the Hough transformation cases in practice. In our system, the map lines are extracted from a reflection map generated by Lidar 3D points while the detected lines are extracted from RGB images. This divergence would effect the precision of localization with match in stereo. Therefore, we project directly the pixels.



(a) Hough transformation in best case

(b) Hough transformation in normal case

Figure 38: Hough transformation in line extraction

As for cost function, in order to increase the longitudinal constraint, we choose the distance of nearest edge points, as discussed in Section 2.2 D:

$$p(z_t|\hat{x}_t) = f(D_1, D_2, ..., D_n) \tag{31}$$

To compute $p(z_t|\hat{x}_t)$, the complexity is $O(n^2)$. It is importance to reduce the number of pixels to accelerate the algorithm. In this condition, we choose to compress the image (from 712×1192 to 462×774) and only use the edge of land markers:

Figure 39: Edge extraction, Canny edge detector

Figure 39 demonstrates the edge extraction, with a Canny edge detector. Using edge pixels can not only reduce the pixels in math, but also can provide a more precise depth information.

4.3.3 Evaluation and inadequacies

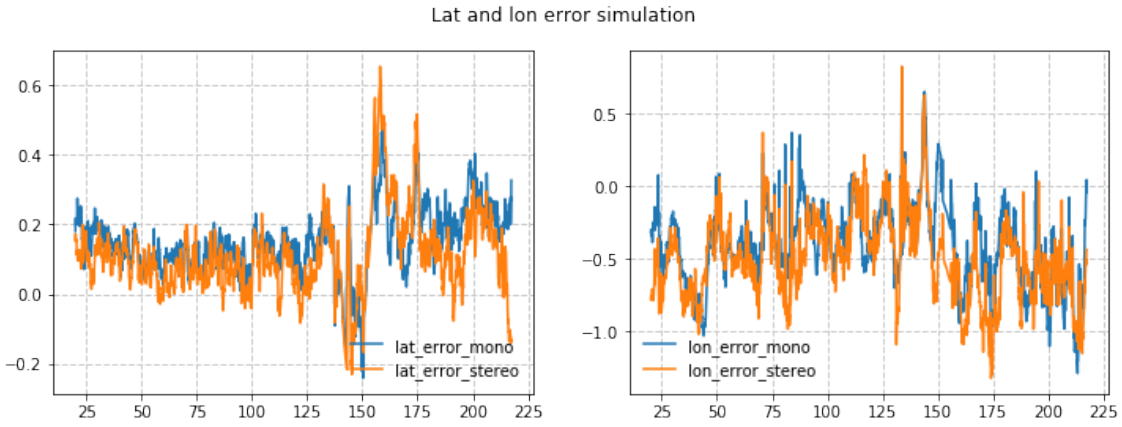


Figure 40: Lateral error and longitudinal error, mono camera(orange) and stereo camera(blue)

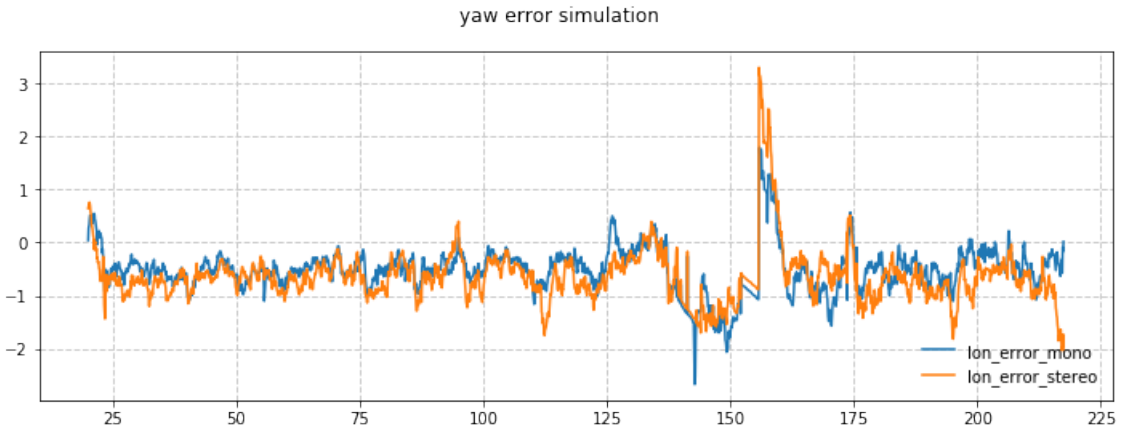


Figure 41: Heading error(yaw), mono camera(orange) and stereo camera(blue)

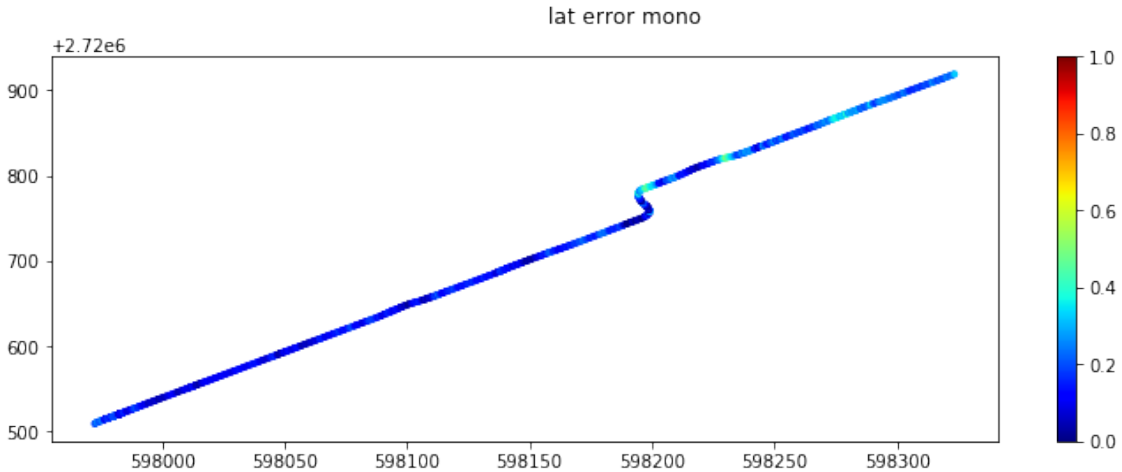


Figure 42: Lateral error in global coordinates, use mono camera

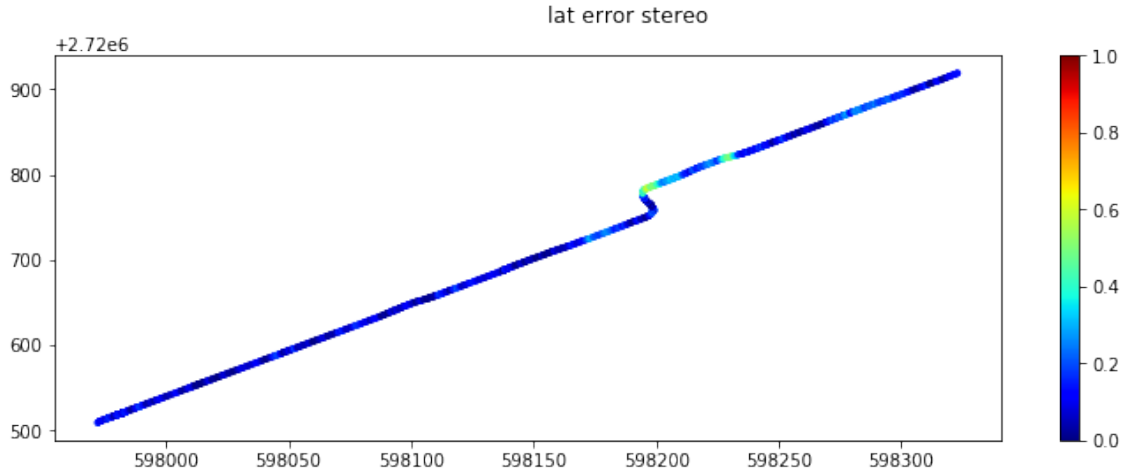


Figure 43: Lateral error in global coordinates, use stereo camera

Figure 40, 41, 42 and 43 show the comparison of mono camera based localization system and stereo camera based localization system, we find that the lateral localization precision has been increase. The statistical result can well prove it:

	mean	std	$e_{0.8}$	$e_{0.9}$	$e_{0.99}$
lateral error(m)	0.1584	0.0826	0.2187	0.2666	0.4184
longitudinal error(m)	0.4376	0.2451	0.6431	0.7943	1.0776
yaw error(deg)	0.5695	0.3409	0.7996	0.9993	1.6665

Table 7: Localization error, mono camera

	mean	std	$e_{0.8}$	$e_{0.9}$	$e_{0.99}$
lateral error(m)	0.1255	0.1015	0.1810	0.2402	0.5075
longitudinal error(m)	0.5323	0.2625	0.7641	0.8864	1.1428
yaw error(deg)	0.7277	0.3785	0.9730	1.1691	1.8917

Table 8: Localization error, stereo camera

However, the problem of decrease in longitudinal and heading precision still exists. In stereo camera based system, we no longer suffer from the camera distortion, but we meet the new challenge, the mismatch and depth precision.

Mismatch

The mismatch problem comes in depth restoration for stop lines. As shown in Figure 44, the epipolar line is generally introduced in match point search. For a pixel in a stop line, its epipolar line overlaps with the stop line, which means a high probability to match with a wrong pixel. In Figure 45, we draw the point cloud of land markers in 3D space (use library PCL), the red circles are stop line and its projection in 3D space, we could find that the mismatch of stop line will lead to a totally wrong depth. In practical implementation, we delete all the stop lines in stereo camera based localization system.

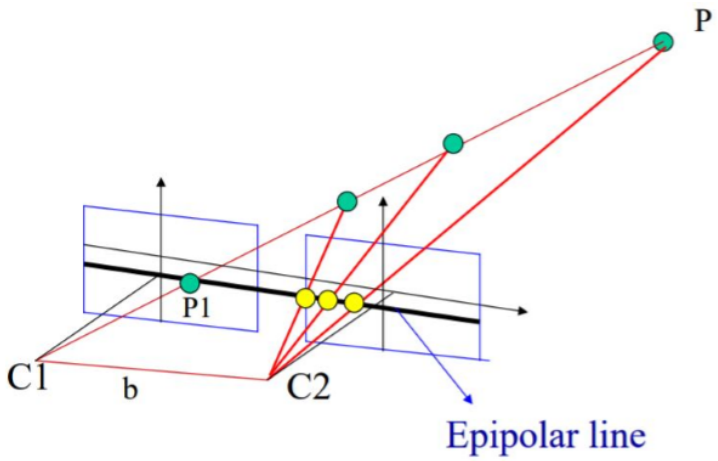


Figure 44: Mismatch for lateral lines

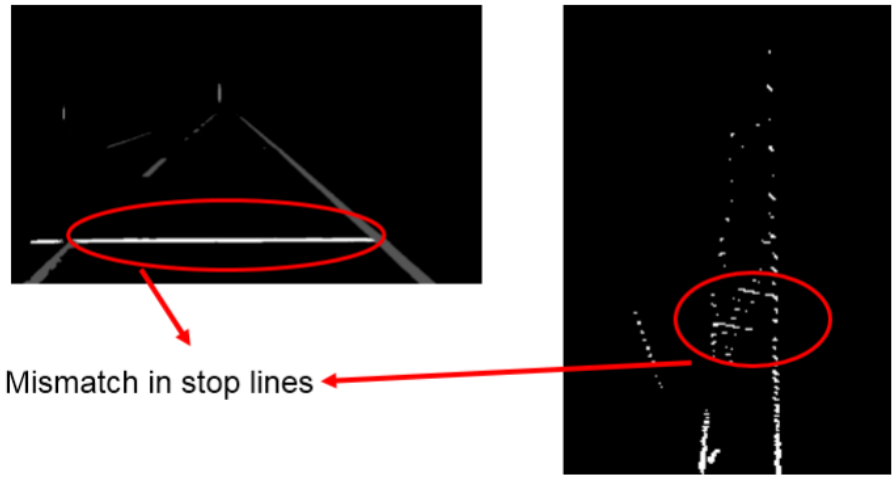


Figure 45: Influence of mismatch for stop lines

Depth precision

Besides the mismatch, the depth precision will also affect the localization accuracy, which is caused by calibration deviation and potential mismatch for the other land markers.

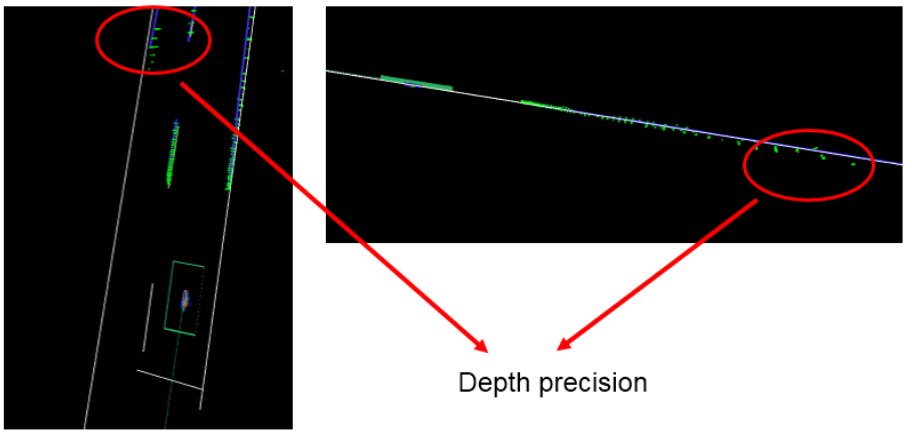


Figure 46: Depth precision

5 Conclusion and future work

During my 5 months internship, my work was arranged as follows. The first one and a half months, I took over the predecessor's work and rebuild several parts of localization system in order to support a multi-camera frame. After that, in the next one month, I investigated the recent research on line-based match method and proposed a feasible match method for lane marker match in 3D space. For the final two and a half months, I developed the stereo-camera localization system based on the former one, analyzed its performance and its drawback. For the lateral localization precision, the most important in autonomous vehicle, we finally achieve a 10 *cm* precision based on dual camera system and a 12 *cm* precision based on stereo camera system, however, both of them have the potential improvement.

For future work of vision based localization for autonomous vehicle, on the one hand, we could continue to optimize this particle filter based frame, try to solve the problems proposed in this report, or try to fuse multiple frames (in our system, we only use the current frame of image and the particles' weights in the previous frame). On the other hand, we could use other models, if the computation resource is enough, with bundle adjustment and pose-graph relaxation, the optimization based localization would have a higher precision than filter based method [12], in [17] a direct method for SLAM based on normalized information distance is proposed for being higher robust to significant scene changes, furthermore, recent researches give the potential combination of stereo vision and deep learning [18].

During this internship, I get much more familiar with the visual SLAM system, especially the pose estimation based on filter or based on optimization. Meanwhile, I obtain my first experience of programming in C++ and operation on Linux. I would like to thank my tutors and my colleagues for their help and guidance in my internship.

References

[1] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[2] Guowei Wan, Xiaolong Yang, Renlan Cai, Hao Li, Hao Wang, and Shiyu Song. Robust and precise vehicle localization based on multi-sensor fusion in diverse city scenes. *arXiv preprint arXiv:1711.05805*, 2017.

[3] Ji Zhang and Sanjiv Singh. Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, 41(2):401–416, 2017.

[4] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[5] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22, 2000.

[6] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[7] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[8] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[9] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *arXiv preprint arXiv:1708.03852*, 2017.

[10] Kichun Jo, Yongwoo Jo, Jae Kyu Suhr, Ho Gi Jung, and Myoungho Sunwoo. Precise localization of an autonomous car based on probabilistic noise models of road surface marker features using multiple cameras. *IEEE Trans. Intelligent Transportation Systems*, 16(6):3377–3392, 2015.

[11] Albert Pumarola, Alexander Vakhitov, Antonio Agudo, Alberto Sanfeliu, and Francese Moreno-Noguer. Pl-slam: Real-time monocular visual slam with points and lines. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 4503–4508. IEEE, 2017.

[12] Luc Jaulin, Michel Kieffer, Olivier Didrit, and Eric Walter. *Applied interval analysis: with examples in parameter and state estimation, robust control and robotics*, volume 1. Springer Science & Business Media, 2001.

[13] Markus Schreiber, Carsten Knöppel, and Uwe Franke. Laneloc: Lane marking based localization using highly accurate maps. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 449–454. IEEE, 2013.

[14] Yijia He, Ji Zhao, Yue Guo, Wenhao He, and Kui Yuan. Pl-vio: Tightly-coupled monocular visual–inertial odometry using point and line features. *Sensors*, 18(4):1159, 2018.

[15] Manohar Kuse and Shaojie Shen. Robust camera motion estimation using direct edge alignment and sub-gradient method. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 573–579. IEEE, 2016.

[16] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[17] Geoffrey Pascoe, Will Maddern, Michael Tanner, Pedro Piniés, and Paul Newman. Nid-slam: Robust monocular slam using normalised information distance. In *Conference on Computer Vision and Pattern Recognition*, 2017.

[18] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. Mvsnet: Depth inference for unstructured multi-view stereo. *arXiv preprint arXiv:1804.02505*, 2018.