CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Vision-based Vehicle Ego Velocity Estimation

Master's thesis in Computer science and engineering

RUIKUN DENG, RUI PENG

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Vision-based Vehicle Ego Velocity Estimation

RUIKUN DENG, RUI PENG

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Vision-based Vehicle Ego Velocity Estimation

RUIKUN DENG, RUI PENG

Supervisor: Ulf Assarsson, Department of Computer Science and Engineering
Advisor: Dhasarathy Parthasarathy, Volvo Group
Examiner: Erik Sintorn, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Vision-based Vehicle Ego Velocity Estimation
RUIKUN DENG, RUI PENG
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Vehicle ego velocity estimation is an interesting topic of research, as a reliable and accurate estimation method is crucial in vehicle motion control. At the same time, smart vehicles are often equipped with multiple sensors, such as cameras, radar, and so on. By incorporating the vision-based velocity method, we can increase the system redundancy in case of other sensor failures. In this thesis, we proposed a method that can predict accurate forward velocity as well as leftward velocity. In our experiments, the best performance is 0.526(km/h) MAE and 0.751(km/h) RMSE for forward velocity estimation, and 0.171(km/h) MAE and 0.250(km/h) RMSE for leftward velocity estimation. Moreover, to make our method more reliable in practical use, we also performed uncertainty estimation on the model with the best performance, which makes our method more applicable.

Keywords: Deep learning, Computer Vision, Velocity estimation, Uncertainty estimation

Acknowledgements

Many thanks to Parthasarathy Dhasarathy and Karlsson Daniel at Volvo Truck for providing the opportunity for the thesis project. Also many thanks to Ulf Assarsson and Erik Sintorn as our supervisor and examiner for their invaluable guidance and support throughout our thesis.

Ruikun Deng, Rui Peng, Gothenburg, June 2023

Contents

List of Figures

List of Tables

1

Introduction

Motion control involves all techniques that influence the dynamics of a vehicle, which is a key component for smart and safe vehicles. Velocity estimation plays an important role, which provides critical information to the vehicle system. For example, by estimating the forward velocity, the vehicle's engine can be optimized to operate at the most fuel-efficient speed. Additionally, lateral velocity estimation can help optimize the vehicle's suspension and handling for improved stability and comfort.

1.1 Current Practice

A common way to estimate the runtime velocity is to use an Inertial Measurement Unit (IMU). The IMU is an electronic device that measures linear and angular acceleration using a combination of sensors such as accelerometers and gyroscopes. Using the acceleration and the time interval, we can estimate the velocity indirectly. However, IMUs can sometimes be inaccurate due to several reasons. One reason is that IMUs measure the rate of change of acceleration and rotation, and these measurements are integrated over time to calculate position and orientation. However, even small errors in these measurements can accumulate over time, resulting in drift. Drift is especially problematic when there is no external reference to correct it. Besides, IMUs require accurate calibration to function. Calibration errors can arise due to issues like manufacturing defects, aging of components, or temperature changes. Even a small calibration error can significantly impact the accuracy of the IMU.

There are also other ways, such as wheel speed sensors and Global Positioning Systems (GPS). However, the wheel speed sensor cannot estimate the lateral velocity and GPS suffers from signal loss and signal reflection in complex environments. Surely, we can use more accurate sensors and GPS devices to ensure a higher accuracy of velocity estimation, but the high cost cannot be ignored in practice. Thus, there is always a trade-off between the performance of physical components and the cost of them. Therefore, a new method that has great performance at an acceptable cost is becoming important in the industry.

1.2 Computer Vision

One practical choice is to use cameras, i.e. computer vision, which is a field of study focused on enabling machines to interpret and understand digital images or videos to stimulate humans' views. Considering the way how human beings sense how fast they are moving, vision is a critical functionality. When an object appears to become larger as it approaches us, our brain can use this information to update the distance between this object and us, and then estimate its velocity. By estimating the velocities of other objects and understanding the environment we stay in, we conclude our ego velocity. Following this idea, we are hopefully using computer vision techniques to simulate such procedures and help the vehicles to estimate their ego velocities.

Object distance plays a critical role in velocity estimation. Fortunately, there are many related works in the computer vision field. The plain idea is to use triangulation [1]. We can use monocular cameras to retrieve images from two different positions and locate the objects by triangulation. The drawback of this idea is triangulation is used to compute the distance between the target and the camera. However, to estimate ego velocity, the methods should be able to understand the surrounding environment, which the triangulation cannot satisfy. A more practical technique is Simultaneous Localization and Mapping (SLAM) [2], which is a series of techniques aiming to build a map of an unknown environment in real-time, while also keeping track of its own position and orientation within that environment. SLAM techniques always integrate data from cameras, LIDAR, and IMUs to model the environment. Although they do not necessarily require camera data, cameras are one of the most common and powerful sensors used in SLAM. We are highly inspired by these ideas to build our method to solve our problem.

1.3 Neural Networks

Neural Network (NN) [3], the base of AI technique, has been more and more popular since 2012. Since it was introduced to computer vision, it has significantly stimulated the development of this field. Inspired by the neuron in the human brain, a neural network is a type of algorithm that is designed to recognize patterns in data.

To be able to learn like humans, NNs are usually designed with interconnected nodes that process and transmit information using mathematical operations. Usually, a NN is often used as a tool for optimization because it has the ability to learn and improve the performance of a given task through the process of training. During training, the NN adjusts its internal parameters (weights and biases) in response to a set of input-output pairs, in order to minimize the difference between the predicted outputs and the ground truth outputs.

NN is particularly useful in computer vision, where the raw input data is a high-dimensional image with many pixels, and it is often difficult to manually engineer features that can accurately represent the information in the image. A NN can be trained to recognize patterns and features in images, such as edges, lines, and

shapes, and explain these features at an abstract level so that machines can use the information to address different tasks, e.g. classification, object detection, etc.

2

Related Work

In this chapter we will discuss previous methods that have been proposed to estimate vehicle velocity and uncertainty. To estimate ego-motion based on monocular camera vision is an interesting topic even before deep learning became popular. Qifa Ke and Takeo Kanade leveraged the direct least-squares method for recovering motion to calculate the incremental image motion between the reference frame and another frame in the same video [4], [5]. Koichiro et al. estimated ego-motion by correspondence points detection and 3D structure reconstruction [6].

Then as deep learning showed great ability to solve computer vision problems, methods leveraging deep learning were proposed and achieved good results. Moritz et al. used neural networks to first extract features from original monocular RGB input frames, then regressed the relative velocity of each vehicle in the input sequence using the extracted features [7]. S Umamaheswaran et al. combined deep learning and traditional computer vision methods to estimate the speed of cars near to the camera. With the input of a pair of consecutive frames, they used the You Only Look Once (YOLO) algorithm to detect nearby vehicles, then used the Scale Invariant Feature Transform (SIFT) to extract matching feature points to calculate vehicle position and speed [8]. Zhenbo Song et al. combined multiple visual features extracted from input frames, including depth, scene geometry as well as optical flow, to estimate inter-vehicle relative velocity [9]. Regarding velocity estimation from a sequence of video frames as a time-dependent problem, Hitesh and Binoy proposed their method based on long short term memory (LSTM), proving that the Recurrent Neural Network (RNN) is also capable of performing velocity estimation [10]. Athul and Thariq also noticed the importance of spatial relationships across consecutive video frames, and they used 3D Convolutional Neural Networks to better capture the features and produced good results [11].

3

Theory

In the following sections, we will introduce the definition of our problem and the theory related to our task.

3.1 Problem Formulation

Given a sequence of consecutive image frames captured by a camera and their associated timestamps, the intrinsic camera parameters, and the ground truth of IMU data in terms of forward velocities and leftward velocities, the objective is to build a machine learning model which is able to estimate the camera's velocity purely from the image data. More specifically, let

- I be the input image sequence, and I_t be the image frame at time t.
- V be the ground truth velocities and V_t be the velocity at time t.
- $\hat{V}$ be the predicted velocities and $\hat{V}_t$ be the velocity at time t.

The Problem is to find a method F that

$$F(I_{t-1}, I_t) = \hat{V}_t \tag{3.1}$$

which minimizes the error

$$E = |\hat{V}_t - V_t| \tag{3.2}$$

3.2 Neural Networks

Neural networks (NNs) refer to a kind of computational model inspired by the structure and function of biological neural networks. It consists of interconnected layers of artificial neurons or nodes that process and transmit information. The fundamental goal of a neural network is to learn patterns or relationships in data through a process called training. An artificial neuron, or node, receives input from other neurons or external data. Each input is multiplied by a weight, representing the strength of the connection between the neurons. The weighted inputs are then summed and passed through an activation function, which determines the output of the neuron. The procedure can be described as:

$$y = f(\sum(wx + b)) \tag{3.3}$$

where x, w, and b are the input, weight, and bias of a node in a neural network. The linear operations among them enable the neuron to do linear predictions. f, the activation function, introduces non-linearity into the neuron, allowing it to learn complex patterns and relationships in the data. By adjusting the value of w and b, such a neuron can be viewed as a simple mathematic model to fit different functions. Built with millions and even billions of neurons, neural networks obtain an amazing ability to make reliable predictions for a specific scenario.

3.3 ResNet

ResNet [12], also called Residual Neural Network, is designed to address the issue of vanishing and exploding gradients, which tend to occur in deep neural networks as the number of layers increases. A deeper neural network is usually considered to have a stronger power to understand abstract information and concept from the input. Since our target is to make our model able to understand the distance and velocities from images, ResNet will be helpful to enhance learning ability. There are some key components of RestNet which will be described as follow.

3.3.1 Residual Block

The building block of a ResNet is the residual block. Each residual block consists of a series of convolutional layers, followed by batch normalization layers and activation functions (typically ReLU). The input to the block is added to the output of the block through the residual connection, forming a skip connection. Figure 3.1 gives the visualization of a residual block.

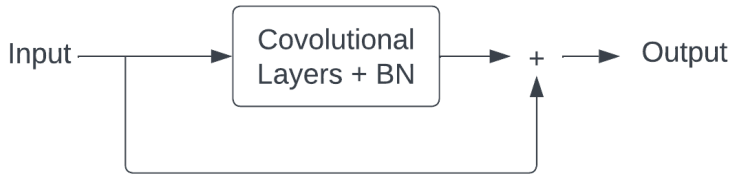


Figure 3.1: An example to describe a typical residual block: the most novel part of this idea is the output is composed of the input and the output of some convolutional and Batch Norm (BN) layers. The notation "+" refers to an identity mapping function.

The identity mapping function in Figure 3.1 is applied when the input of a block or layer is directly added to its output without any modification, which can be expressed as:

$$\text{Output} = \text{Input} + f(\text{Input}) \tag{3.4}$$

It is important to note that, in some cases, a simple identity mapping may not be directly applicable if the input and output dimensions of the residual block are different. In such cases, a linear projection can be used to match the dimensions before applying the residual connection. This can be achieved using a 1x1 convolutional layer without an activation function or a fully connected layer.

In practice, each residual block consists of a series of convolutional layers, typically with 3x3 filters, followed by batch normalization and ReLU activation. Here is a common breakdown of the structure of a residual block:

1. Convolutional Layer: A 3x3 convolutional layer is applied to the input. The purpose of this layer is to learn features from the input data.
2. Batch Normalization: Batch normalization is applied after the convolutional layer. It normalizes the outputs of the previous layer, ensuring that the inputs to the activation function are centered and scaled properly. This helps in stabilizing and accelerating the training process.
3. Activation (ReLU): ReLU (Rectified Linear Unit) activation is applied element-wise to the output of the batch normalization layer. ReLU introduces non-linearity, allowing the network to learn complex relationships between features.
4. Convolutional Layer: Another 3x3 convolutional layer is applied to the output of the activation. This layer further extracts features from the input data.
5. Batch Normalization: Batch normalization is applied again after the second convolutional layer.
6. Identity Mapping: Identity mapping refers to the skip connection that directly connects the input of the block to its output. This is done by adding the input of the block (identity) to the output of the second convolutional layer. The purpose of identity mapping is to ensure the smooth flow of information through the network, enabling the gradient to propagate easily during training.
7. Element-wise Sum: The output of the identity mapping is added element-wise to the output of the second convolutional layer. This sum operation combines the original input with the learned residual, allowing the network to learn the residual mapping.

Recall the definition of the gradient vanishing problem, which is gradients are probably close to zero while backpropagating. It will lead to part of the neural network losing the ability to learn while training. However, by creating skip connections, the gradients can flow more directly through the network, preventing them from becoming too small as they backpropagate. In effect, the gradients have a more direct path to the early layers of the network, making it easier for the network to learn and adjust the weights, i.e. against gradient vanish.

3.3.2 Bottleneck Block

To reduce computational complexity, a variant of the residual block called the bottleneck block is often used in deeper ResNets. A bottleneck block consists of three

convolutional layers organized as follows:

1. 1x1 convolutional layer that reduces the number of input channels (dimensionality reduction).
2. 3x3 convolutional layer that performs the main spatial convolution with the reduced number of channels.
3. 1x1 convolutional layer that restores the original number of channels (dimensionality expansion).

This structure allows the network to save computational resources by performing the main convolution operation on a reduced number of channels. Also, using 1x1 convolutions for dimensionality reduction and expansion helps control the number of parameters in the network. Besides, each bottleneck block also has a residual connection (skip connection) that adds the input of the block to its output.

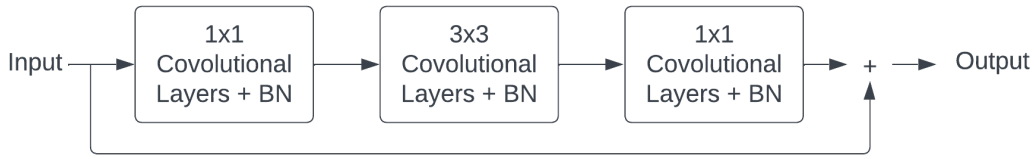


Figure 3.2: An example to describe a typical bottleneck block. A variant of the residual block is able to decrease the complexity of the neural network.

3.3.3 ResNet-18

ResNet-18 is one of the popular implementations of the ResNet architecture. It is known for its compact architecture compared to deeper variants like ResNet-50 or ResNet-101. Despite having fewer layers, it achieves strong performance and strikes a balance between complexity and computational efficiency.

As Figure 3.3 describes, Resnet-18 consists of four stages, each containing two residual blocks. Each residual block has two convolutional layers. The architecture starts with initial convolutional and pooling layers to extract low-level features, followed by multiple stages of residual blocks. Finally, global average pooling, a fully connected layer, and a softmax activation are applied for the prediction.

We believe Resnet-18 has a strong ability to understand abstract information from images, Therefore, we use it as a projector to predict the velocities from the abstract high-dimension features given by the other pre-trained models.

3.4 Model Pretraining

Model pretraining is a crucial step in the development of machine learning models. The main idea behind pretraining is to train a model on a large dataset before fine-tuning it on a smaller, task-specific dataset. Pretraining helps to initialize

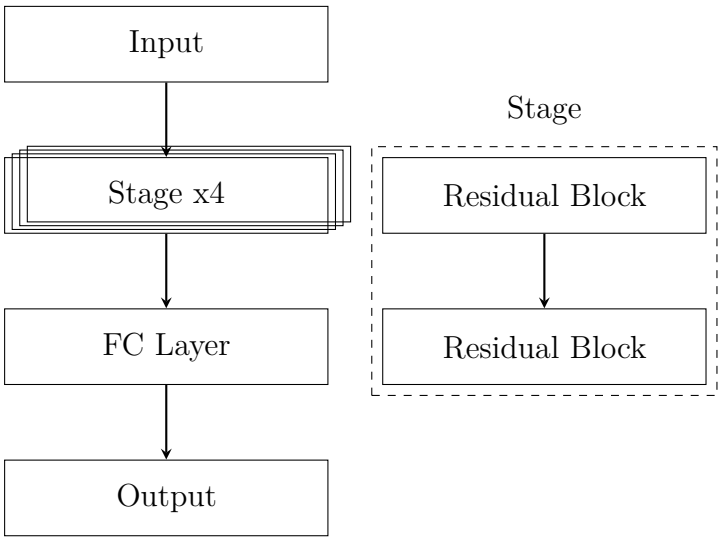


Figure 3.3: ResNet-18 Architecture: the input image is passed through a 7x7 convolutional layer with a stride of 2 to extract initial low-level features. Then the features go through four stages. Each stage contains residual blocks. Then a global average pooling layer is applied to reduce the feature map dimensions. Finally, a fully connected layer is applied to output the prediction

the model with meaningful weights and representations that can be refined further during the fine-tuning stage. This process often leads to better performance and faster convergence when compared to training a model from scratch on the target task.

There are two different types of model pretraining we are interested in: transfer learning [13], [14] and unsupervised pretraining [15]. In transfer learning, the model is pretrained on a large, labeled dataset from a related task or domain. The pretrained model is then fine-tuned on the smaller, target dataset. It is particularly effective when the source and target tasks share similar underlying structures or patterns. A prominent example of transfer learning in natural language processing is pretraining of models like BERT or GPT on massive text corpora to learn general language understanding, which can then be fine-tuned for specific tasks like sentiment analysis or question-answering. In unsupervised pretaining, the model is first trained on an unsupervised task, such as autoencoding or clustering, using a large dataset. The objective is to learn useful feature representations without relying on labeled data. Once the unsupervised pretraining is complete, the model is fine-tuned using a smaller, labeled dataset for the specific task of interest, such as classification or regression.

Transfer learning can be highly beneficial for our task. By utilizing a pretrained model that has already learned robust and meaningful feature representations from large-scale image datasets, we can improve the performance, data efficiency, and convergence speed of our vehicle velocity estimation model. In the following sections, we will introduce some related methods which are potentially useful for us to construct our own methods.

3.5 Optical Flow

Optical flow [16] is used to present the pixel-level motion between frames. To be more specific, assuming we have a time series frame list I, we use $I(x, y, t)$ to represent the pixel at the location (x, y) and time t. Then we have the equation:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t), \text{where } \Delta x, \Delta y, \Delta t \to 0 \tag{3.5}$$

According to Taylor series, the equation above can be written as:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t + o \tag{3.6}$$

Notation o represents the higher order terms, which can be ignored. Combining equation 3.5 and 3.6, we can get:

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0 \tag{3.7}$$

Divide Δt on both side of equation 3.7, then the equation turns to be:

$$\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t} = 0 \tag{3.8}$$

We use I_x, I_y, I_t, V_x, V_y to represent $\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}, \frac{\partial I}{\partial t}, \frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t}$, respectively. The equation 3.8 can be denoted as:

$$I_x V_x + I_y V_y + I_t = 0 \tag{3.9}$$

In this context, the vector (V_x, V_y) is the optical flow of the pixel $I(x, y, t)$. There are plenty of methods to estimate it.

3.6 Camera Pose

Camera pose, i.e. 6DOF pose, refers to the position and orientation of the camera in 3D space. The term "6DOF" stands for "six degrees of freedom," which refers to the six parameters that are needed to fully specify the camera's pose: three for the position (up, left, and forward coordinates) and three for orientation (yaw, pitch, and roll angles), shown as Figure 3.4.

Assume we have a time series image list I and I_t to represent the pixel at time t. Note the camera pose of image I_t as P_t, and Δ as the moving distances between frame I_{t-1} and I_t, we can compute Δ as

$$\Delta = P_t - P_{t-1} \tag{3.10}$$

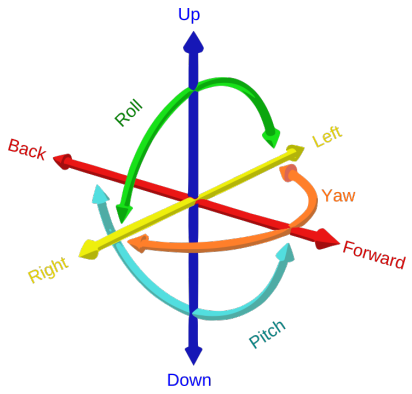


Figure 3.4: Camera 6DOF Pose: 3 directions (up, left, and forward coordinates) + 3 orientations(yaw, pitch, and roll angles)

Besides, we use Δ_x, Δ_y to represent the moving distances on forward and leftward directions. We hope to find a method F which satisfies:

We hope to find a method F which satisfies:

$$(\Delta_x, \Delta_y) = F(I_t, I_{t-1}) \tag{3.11}$$

so that we can estimate the velocities by

$$V_x = \frac{\Delta_x}{\Delta_t}, V_y = \frac{\Delta_y}{\Delta_t} \tag{3.12}$$

3.7 Camera Depth

In computer vision, camera depth, also called depth perception and depth mapping, often refers to the distance between the objects and the camera. It can be used to create a sense of three-dimensionality in images, which makes it a crucial technique to virtual reality, robotics, and photography. Pixel-level camera depth refers to obtaining depth information for each pixel in an image, where each pixel corresponds to the distance from the camera to a specific point in the scene (Figure 3.5).

Since taking a photo is initially a process that projects a 3D scenario to a 2D plane, the photo loses 1 dimension of information critical to restoring objects from 2D to 3D. Thus, the mainstream way is to use sensors and extra devices to capture extra data to compensate for such information missing. However, neural network-based methods can also provide impressive accurate results for estimating per-pixel depth. For example, Monodepth2 [17] is one of the state-of-art machine-learning methods to generate camera depth from the image taken by monocular cameras. It takes a single image input I and produces a depth map D via a convolutional neural network. Moreover, this model is pretrained with KITTI dataset. Therefore, it is sensitive to the traffic scenario and can be smoothly transferred and used in our work.

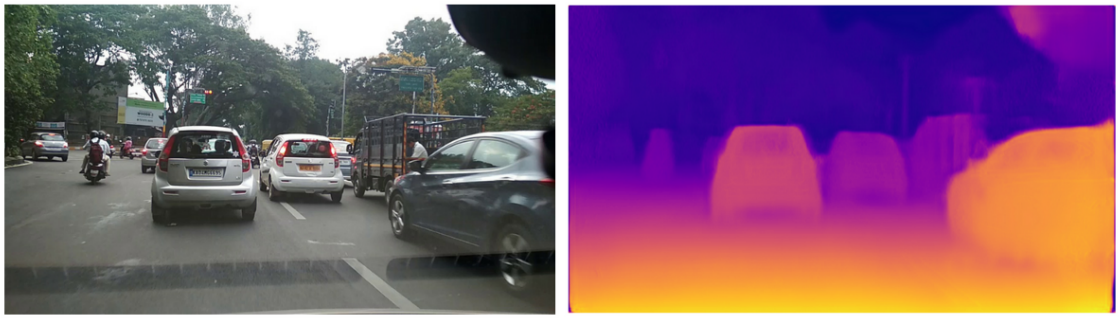


Figure 3.5: Depth map: the left is the original picture and the right is the correlating depth map. The farther the pixel is in 3D space, the darker the color is in-depth map

3.8 Uncertainty Estimation

Model Uncertainty refers to the model's inability to make precise predictions or decisions due to limitations in the data, training process, or the complexity of the problem being solved [18]. Essentially, it is a measure of the confidence the model has in its predictions. It is an essential reference for us to evaluate whether we should trust the result of our model.

Usually, common NNs only focus on decreasing the residual error between the predicted values and ground truth. To model this process, let the input data be $x \in X$, the ground truth be $y \in Y$, and the NN model be $f : \hat{y} \rightarrow f(x)$ where $\hat{y}$ refers the prediction. Then, the mission of training such a model is to use backpropagation to decrease the residual error $||y - \hat{y}||$. It has proved to be useful in different fields. However, this method has an important assumption, which is the data used to train the model, i.e. X and Y, should be accurate. However, in practice, there is an unavoidable gap between the observed data and the true data, which is called noise. Therefore, knowing how confident the model is in its prediction and the observed data, i.e. uncertainty estimation, is critical.

Generally, uncertainty can be divided into aleatoric uncertainty and epistemic uncertainty, caused by data noise and model instability, respectively [19]. We will introduce and model them in the following subsections.

3.8.1 Epistemic uncertainty

Epistemic uncertainty arises from the model's limited knowledge or lack of understanding about the underlying process that generates the data. It is also known as model uncertainty or reducible uncertainty.

Before modeling epistemic uncertainty, we need to model the prediction process of NNs. Let $D = \{d_1, d_2, ..., d_n\}$ be the observed data, i.e. training data. Let $\hat{d} \notin D$ be the unobserved data. Then the prediction process can be described as $p(\hat{d}|D)$, i.e. the possibility distribution of outputs given the observed dataset. When we consider the influence of model uncertainty, we mark the model as $m \in M$. Instead of being

a deterministic model in common NNs, here the model is non-deterministic. Then the prediction process can be described as

$$p(\hat{d}|D) = \int_M p(\hat{d}, m|D)dm = \int_M p(\hat{d}|m, D)p(m|D)dm \tag{3.13}$$

Because we know once the model m is confirmed, the output unobserved data $\hat{d}$ is confirmed, i.e. $p(\hat{d}|m, D) = p(\hat{d}|m)$. Then we can rewrite the equation 3.13 as

$$p(\hat{d}|D) = \int_M p(\hat{d}|m)p(m|D)dm \tag{3.14}$$

Inside equation 3.14, $p(m|X)$ represents the epistemic uncertainty clearly. It shows the probability, i.e. uncertainty, of our model to take m from the model space M. Unfortunately, computing $p(m|D)$ is intractable. To explain it, we need to split data D to inputs X and outputs Y. Then the epistemic uncertainty can be rewritten as $p(m|X, Y)$, and

$$p(m|X, Y) = p(m)\frac{p(Y|X, m)}{p(Y|X)} \tag{3.15}$$

Apparently, modeling $p(Y|X)$ is intractable because it is the final target of the NNs training processes. But if we have infinite training data and ignore the aleatoric uncertainty, i.e. the training data contains no noise, we can find the proper $m \in M$ to 100% fit the training data. In that case, the epistemic uncertainty will disappear. However, in practice, it is impossible to collect infinite data without noise, making it difficult to decrease epistemic uncertainty.

Many methods are proposed to solve this problem. The most common approach is to use a simple probability distribution to replace $p(m|X, Y)$, which is also called variational inference. A very direct method proposed in 2015 [20], replaces the fixed value of NNs weight by a prior distribution and learns the posterior distribution in the training process. However, this method is too computationally complex, especially for some large NN architectures. Therefore, an approximate method was proposed to add Monte-Carlo dropout between NN layers to simulate Bernoulli distributions [21]. This method makes normal NNs into non-deterministic NNs without adding new parameters to avoid increasing computing complexity. It will be an important method when we measure and decrease the epistemic uncertainty of our models.

After we model the weight distribution and assume there is no aleatoric uncertainty in the observed data, we can use the variance of the predicted outputs, marked as $\sigma_{\hat{y}}$ to approximate the epistemic uncertainty. However, the aleatoric uncertainty in our case cannot be simply ignored. In the next section, we will discuss how to model it.

3.8.2 Aleatoric uncertainty

Unlike the epistemic uncertainty modeling the uncertainty of the model itself, the aleatoric uncertainty is used to model the uncertainty of the input data x, i.e. the

noise of input data denoted as σ_x. It is a very difficult task because we need to learn or predict the noise for the unobserved data. Therefore, to estimate the noise for the unseen data, there are two different assumptions.

The first assumption is that the noise for every input is always a constant value. The regression mission following this assumption is called homoscedastic regression. If we view a common NN regression model, we will figure out that the noise is learned by the model as the model updates. And in the end, the influence of the noise on the model will be theoretically ignored.

However, the first assumption does not fit our case well. In our case, the noise of data definitely depends on the driving scenario. For example, if a vehicle is driving on a mountain road in bad condition, the video captured will be shaking, which can be viewed as huge noise inside the input data. And, on the other hand, if the vehicle is driving here on a city road in good condition move, the input data will contain less noise. To define this kind of regression mission, there is a second assumption, called heteroscedastic session. Suppose the noise is different based on the different data. We will follow this assumption to model the aleatoric uncertainty of our model.

To model the heteroscedastic uncertainty, let the x, y be the input and output data, f be the model, w be the weight of the model and σ be the variance of x. Then the optimizing target can be described as

$$L(w) = \frac{||y - \hat{y}||^2}{2\sigma_x^2} + \frac{1}{2}\log\sigma_x^2 \tag{3.16}$$

in which $\hat{y} = f^w(x)$ and f^w refers to the prediction function of the model with weights w. However, the input noise σ_x is hard to estimate. Thus, it is common to find some methods to approximately represent the noise. One method is to use the non-deterministic model to predict a set of outputs $\hat{y}$, and then use the variance $\sigma_{\hat{y}}$ to approximate the noise,

$$L(w) \approx \frac{||y - \hat{y}||^2}{2\sigma_{\hat{y}}^2} + \frac{1}{2}\log\sigma_{\hat{y}}^2 \tag{3.17}$$

However, this approximation is under the assumption that there is no epistemic uncertainty in the model. Therefore, to combine both epistemic uncertainty and aleatoric uncertainty, another method is to let the model predict both outputs and the variance of the input data, marked as $\sigma_{\hat{x}}$. In this case, the optimizing target can be approximated as

$$L(w) \approx \frac{||y - \hat{y}||^2}{2\sigma_{\hat{x}}^2} + \frac{1}{2}\log\sigma_{\hat{x}}^2 \tag{3.18}$$

Then the difficulty turns to how the model predicts the right $\sigma_{\hat{x}}$. Alex Kendall and Yarin Gal give a great solution[19]. Besides, they also provides a way to combine both epistemic uncertainty and aleatoric uncertainty. We will introduce them in the next section.

3.8.3 Predictive uncertainty

As we introduce two different kinds of uncertainty in the least two sections, it's common to ignore one of them and use the predictive uncertainty to represent another one. Predictive uncertainty is composed of both epistemic uncertainty and aleatoric uncertainty. Therefore, if we want to measure predictive uncertainty more accurately, it is necessary to consider both of them.

Recapping the last section, the model predicts the variance of input $\sigma_{\hat{x}}$ and uses it to approximate σ_x. Let the model with the ability to predict $\sigma_{\hat{x}}$ be f^{σ}, we can describe the prediction process as

$$[\hat{y}, \sigma_{\hat{x}}] = f^{\sigma}(x) \tag{3.19}$$

Based on this, if we consider the epistemic uncertainty, we will need a non-deterministic model. We choose Monte-Carlo dropout Bayesian Neural Network (BNN) [21] as we mentioned in Section 3.8.1. We mark such BNN as $f^{\sigma}_{\mathrm{BNN}}$ and use equation 3.18 as our optimizing target. Besides, to avoid that the zero-division error occurs in practice, it's better to use the variant

$$L(w) \approx \frac{||y - \hat{y}||^2}{2\exp(s)} + \frac{1}{2}s \tag{3.20}$$

in which $s = \log(\sigma^2_{\hat{x}})$. Using this equation, we can reasonably keep the balance between uncertainty prediction and output prediction. On one hand, the square residual error $||y - \hat{y}||^2$ will be depressed by $\exp(s)$. It means if the input contains too much noise, the model tends to ignore the residual error and update its weights less. On another hand, the rest part of the equation, s, makes sure the model cannot ignore all inputs by simply increasing the predicted uncertainty.

At last, to model the summarized uncertainty during inference, we use the below equation to approximate the predictive uncertainty σ,

$$\sigma = \lambda\bar{\sigma}_{\hat{x}} + (1 - \lambda)\sigma_{\hat{y}} \tag{3.21}$$

in which $\bar{\sigma}_{\hat{x}}$ is the mean of the predicted input variance, $\sigma_{\hat{y}}$ refers to the variance of the predicted output $\hat{y}$ related to a same input x, and $\lambda \in [0, 1]$ represents n optional parameter limiting the influence of epistemic uncertainty and aleatoric uncertainty.

For example, if the model is going to give a prediction as the input is x, it will forward x with different T times weight $m \in M$ to get $\{\hat{y_1}, \hat{y_2}, ..., \hat{y_T}\}$ and $\{\sigma_{\hat{x}_1}, \sigma_{\hat{x}_2}, ...\sigma_{\hat{x}_T}\}$, marked as $\hat{Y}$ and $\Sigma_{\hat{x}}$, respectively. Then $\sigma_{\hat{y}}$ is the variance of set Y, and $\bar{\sigma}_{\hat{x}}$ is the mean of set $\Sigma_{\hat{x}}$.

4

Methods

In this chapter, we will introduce the design of our NN model in Section 4.1, 4.2. We also state the two loss functions for our deterministic and non-deterministic models in Section 4.3. In Section 4.4, we give a brief introduction to the dataset we used. And because the raw data cannot directly be used for model training, we describe the method to preprocess the raw data in Section 4.5.

4.1 Working with Pretrained Models

In this section, we give a brief description of how we build our deep learning neural networks to achieve velocity estimation.

4.1.1 Velocity-related Feature-extracting Models

According to our research, we found many velocity estimation methods shared the same pattern. To be specific, given the raw input image sequences, they first extracted related image features according to their own designs and then performed velocity estimation based on the extracted features. We also thought this is a reasonable way to solve this task.

In our experiments, we found three pretrained neural networks that we considered useful for extracting velocity-related features. The first neural network is RAFT (Recurrent All Pairs Field Transforms for Optical Flow) [22], which is able to directly predict the optical flow. The second neural network is Monodepth2 [17], which is able to predict monocular depth. The third neural network is designed to predict the camera pose, which is part of Monodepth2 project.

4.1.2 Fine-tunning

After obtaining the pretrained models, the next step is to build our own models by modifying them. In our experiments, the basic idea of modification is to freeze the pretrained models, and add a projection layer. The projection layer takes the velocity-related features extracted from the pretrained model, and predicts the corresponding velocity.

In our experiments, we take the following advantages of fine-tuning:

- Reduced training cost: It is time-consuming and computationally expensive to train a deep learning neural network. The pretrained models already learned the process of feature extraction, so we are able achieve to good performance with reduced cost.
- Adaptation to custom tasks: Fine-tuning allows us to modify the pretrained models to solve our own custom tasks, which brings great flexibility and achieves good performance as well.

Detailed neural network architectures are elaborated on in the next section.

4.2 Neural Network Design

We use three highly valuable pretrained models: RAFT, posenet, monodepth to extract abstract but useful information for further prediction. We have introduced the target of these models in Sections 3.5, 3.6 and 3.7. These models have been meticulously trained and fine-tuned on extensive datasets, enabling them to capture intricate visual features and spatial relationships in the input data. After getting abstract information, we put them into our self-designed projector to project the information to velocities, described in Figure 4.1.

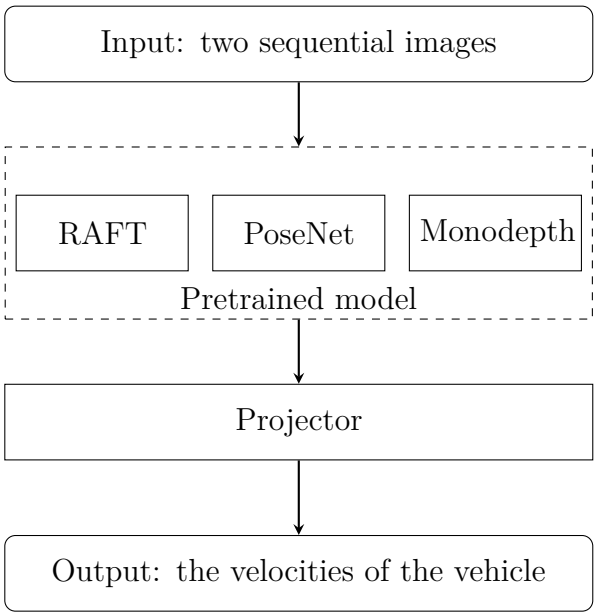


Figure 4.1: Model design: the input, two sequential images, first pass the selected pretrained model and be transferred to some abstract information such as optical flow, pose and depth. Then, the projector will project this information to velocities.

As we introduced in Section 3.4, we will directly use the structure of the pretrained models and reuse the published weights. We design a deterministic projector that purely predicts the velocities and a non-deterministic projector that estimates the uncertainty of the prediction. Both of them will be introduced in the following two sections.

4.2.1 Deterministic Projector

Our deterministic projector is based on Resnet-18, introduced in Section 3.3.3. In the vanilla Resnet-18, the last layer is a fully connected layer that turns the high-dimension features into the possibility of 1000 classes, i.e. it is originally designed for classification tasks. In our case, we modify the dimension of the fully connected layer to turn the high-dimension features into a two-element vector representing velocities. It will also change the final task from classification to regression.

ResNet-18 has been successfully used in various computer vision tasks, including image classification and object detection. We believe it has the ability to extract useful information from two dimension feature maps. Thus, we want to use it to understand and project the abstract information given by the other pretrained models to a velocity-related space, and use the fully connected layer to explain the information in this space to the value of velocities.

4.2.2 Non-deterministic Projector

A non-deterministic model will give different outputs even for the exact same input. This feature makes us able to analyze the probability distribution of the output and indirectly analyze the noise of the input. It is very important in the uncertainty research which we have mentioned in Section 3.8.3.

There are multiple methods to turn a typical NN model from deterministic to non-deterministic. What we use is adding Monte Carlo dropout layers between each layer of NN models, described as Figure 4.2. This modification will approximate Bernoulli-distributed weights, which is important for us to analyze the epistemic uncertainty, mentioned in Section 3.8.1.

Besides, the difference between the standard dropout layer and the Monte Carlo dropout layer needs to be specifically noticed. The standard dropout randomly zeroes some of the elements of the input with probability p using samples from a Bernoulli distribution during only the training process. In the inference process, i.e. applying the model on the unobserved data, the standard dropout layer output is the same as the input value. However, it does not match our expectations. Instead, Monte Carlo dropout does the same as the standard dropout while training but also keeps this behavior while inferencing.

4.3 Loss Design

Since our models are based on NNs, the loss function is important. While training, the model uses it to evaluate the error of the prediction and backpropagates the error to the whole network to update weights. A good loss function makes training faster and the prediction more fittable to targets.

We divide our models into deterministic models and non-deterministic models. Deterministic models refer to the typical NNs which give only the predictions. Besides, if the input is determinate, the output of a deterministic model is fixed. In this

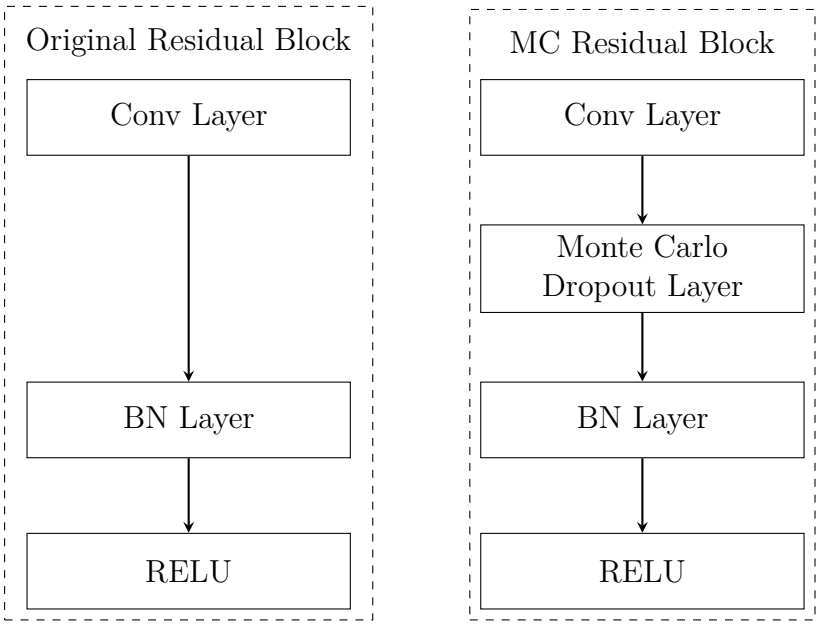


Figure 4.2: Make Resnet-18 non-deterministic: we insert Monte Carlo dropout layer between convolutional layer and batch norm layer and keep the other design the same as the original design

case, the training target is very simple, which is making the prediction close to the ground truth. Therefore, we use MSE loss, explained in Section 4.3.1, to evaluate the difference between the prediction and the ground truth.

Non-deterministic models, in contrast, are more complex. These models have uncertain weights and give different predictions to the same input. Besides, our models also predict the noise of input data to calculate the uncertainty. Thus, a well-designed loss, as we call it predictive uncertainty loss, is required while training. We will introduce it in detail in Section 4.3.2.

4.3.1 Mean Squared Error Loss

Mean Squared Error (MSE) loss is a commonly used loss function in machine learning, particularly in regression tasks. It is a measure of the average squared difference between the predicted and actual values. Assume we have the ground truth y and the corresponding prediction $\hat{y}$ given by our model. The MSE loss L_{MSE} can be expressed as

$$L_{\text{MSE}} = (\hat{y} - y)^2 \tag{4.1}$$

MSE loss penalizes the model more for making predictions that are further away from the actual values. It is widely used because it is continuous, differentiable, and relatively easy to interpret. However, it can be sensitive to outliers since it squares the differences, making it more influenced by large errors. Thus, we use it to help us train our deterministic models and hope it is able to predict the velocities for vehicles as accurately as possible.

4.3.2 Predictive Uncertainty Loss

We transfer our models from deterministic to non-deterministic because we want to import probability theory and make our models able to recognize the noise of inputs. We introduced the uncertainty theory in Section 3.8.3. Here, we use $L(m)$ from equation 3.20 as our loss function L_{PU}. Assume we have the ground truth y and the corresponding prediction $\hat{y}$, $\hat{\sigma}$ given by our model. $\hat{\sigma}$ is the predicted noise of input data, then

$$L_{\mathrm{PU}} = \frac{(y-\hat{y})^2}{2\exp(\log\hat{\sigma}^2)} + \frac{1}{2}\log\hat{\sigma}^2 \tag{4.2}$$

Compared with pure MSE Loss, the MSE $(y-\hat{y})^2$ in equation 4.2 will be depressed by $2\exp(\log\hat{\sigma}^2)$. It means if the model predicts that the input contains too much noise, it tends to ignore the MSE error and update its weights less. For the rest of the equation, $\frac{1}{2}\log\hat{\sigma}^2$, makes sure that the model cannot ignore all inputs by simply increasing the predicted uncertainty.

Since the noise of input data is always difficult to define and label, a significant benefit of using this loss is that we don't need a "noise" label in the dataset to train our model, i.e. our model can implicitly learn the ability to estimate the noise contained in the input data. It provides huge convenience for us to use the same training data as the deterministic model training without relabelling them.

4.4 Dataset

During the project we used two different datasets, one is the KITTI [23], and the other is a confidential dataset named Volvo Truck dataset.

KITTI is a popular dataset that is widely used in mobile robotics and autonomous driving for multiple computer vision tasks. It contains hours of front-facing traffic videos recorded with a variety of corresponding information such as gray-scale stereo sequences, color stereo sequences, 3D Velodyne point clouds and 3D GPS/IMU data.

Volvo Truck dataset contains thousands of hours of front-facing traffic videos featuring high-precision and accurate ground truth recorded by high-performance IMU, such as 3D position, 3D velocity, 3D acceleration and 3D orientation.

4.4.1 KITTI

We used rectified and undistorted KITTI raw data recordings as our dataset, and it is already converted from videos to frames. During our experiments, we split it into two parts. For the training part, the whole training dataset contains 46731 frames, and the validation dataset contains 472 frames. For the test part, the whole test dataset contains 729 frames. Figure 4.3 is a sample video frame.

Besides images, the corresponding log file contains the following data information:

- Unix Timestamps
- Geographical Coordinates: latitude, longitude and altitude.

Figure 4.3: A sample video frame from KITTI dataset

- Altitude: roll, pitch and yaw.
- Velocity: velocity towards north, velocity towards east, forward velocity, leftward velocity, upward velocity.
- Acceleration: forward acceleration, leftward acceleration, upward acceleration.
- Angular rate: angular rate around x, angular rate around y, angular rate around z, angular rate around forward axis, angular rate around leftward axis, angular rate around upward axis
- Accuracy: position accuracy, velocity accuracy.
- Sensor Settings: GPS modes.

In our project, we only used forward velocity and leftward velocity as our ground truth.

4.4.2 Volvo Truck Dataset

The original Volvo Truck dataset is a set of camera videos with corresponding data log. The whole dataset contains approximately 1500-hour videos captured with 30 frames per second. So we randomly chose nine videos as the training dataset, and another one video as the basic test dataset. After that, we had to first preprocess the data before doing experiments, such as conversion from video to images, which is elaborated on in the next part. Here we give a brief description of the Volvo Truck dataset after process.

The training dataset contains 871187 frames, and the validation dataset contains 8799 frames, which adds up to 879986 frames. The basic test dataset contains 55646 frames. Figure 4.4 is one raw video sample video frame. Figure 4.5 is the corresponding undistorted video frame.

Besides the basic test dataset, we also selected another 4 test sets recorded under different weather conditions. They were used to test the generalization ability of our best model.

Besides images, the corresponding log file contains the following data information:

Figure 4.4: One sample video frame from KITTI dataset

Figure 4.5: The undistorted sample video frame from KITTI dataset

- Unix Timestamps
- Geographical Coordinates : latitude, longitude and altitude.
- Attitude: roll, pitch and yaw.
- Velocity: velocity x, velocity y, velocity z.
- Acceleration: acceleration x, acceleration y, acceleration z.
- Angular rate: roll rate, pitch rate, yaw rate.
- Standard Deviation: standard deviation of roll, standard deviation of pitch, standard deviation of yaw.

We only used velocity x and velocity y in our experiments.

(a) Night

(b) Rainy day

(c) Rainy night

(d) Snowy

Figure 4.6: Test samples of different scenarios

4.5 Data Preprocessing

Before constructing our own custom Pytorch dataset, we had to first process our raw data. There are mainly two things to do: synchronization, consistency and image undistortion.

4.5.1 Synchronization

For both datasets, the frames and data log are stored separately, which means we have to find the corresponding velocity information from the data log for each video frame. The point is that the matching pair of frame and data don't have the same exact timestamp, so for each frame, we have to find the data item with the closest timestamp to it. Our naive solution is to search the whole data log for each frame, which has $O(n^2)$ time complexity. This was unacceptable considering our dataset size.

Based on the observation that both timestamps have the ascending order, when we try to find the corresponding data item, the starting point could be set to the next data item of the last matched data item. This new solution has only $O(n)$ time complexity, and turned out to be efficient enough even on the large Volvo Truck

dataset.

4.5.2 Consistency

As our design, the input is a pair of consecutive images. However, both KITTI and Volvo Truck dataset consists of multiple videos, which means the last frame of each video does not have its consecutive frame to make an input pair. So we have to skip each last frame and start from the first frame of the next video. This is done by comparing timestamp interval between two consecutive frames with a pre-set threshold. If this interval is larger than the threshold, it means this pair of frames span two videos, which should be ignored.

4.5.3 Image Undistortion

Unlike the already rectified KITTI images, the raw Volvo Truck videos were recorded with a fisheye camera. As we already mentioned, Figure 4.4 is a distorted frame sample. Since the fisheye camera parameters were not available, our first idea was directly using the distorted frames to train our model, which turned out to work poorly. Therefore we leveraged the double longitude fisheye image correction method to undistort Volvo Truck video frames [24]. Figure 4.5 is the corresponding undistorted frame.

In our implementation, we did not strictly follow the mathematical process of the double longitude method, but the undistortion result is good enough for our experiments.

To be specific, starting from a distorted video frame, we first used openCV [25] to obtain a valid cropped image by mainly using openCV boundingRect function. This step selects the center rectangle area that can retain original information to the greatest extent and ignores frame edges that are too distorted. We set the max between the height and width of this rectangle as the output image length. And we make the centers of the two images coincide logically for later calculation. The intermediate sphere model mapping is as follows:

$$x^2 + y^2 + z^2 = R^2 (z \geq 0) \tag{4.3}$$

$$\tan(\phi) = z/x (0 \leq \phi \leq \pi) \tag{4.4}$$

$$\tan(\theta) = y/x (0 \leq \theta \leq \pi) \tag{4.5}$$

Note that variable R equals half of the previous square length. Then, for each image pixel $p(i, j)$, where i,j are indices for the final undistorted image pixels, we calculate the corresponding original pixel indices from the original cropped rectangle image:

$$\phi = \pi - (\pi/R) * i \tag{4.6}$$

$$\theta = \pi - (\pi/R) * j \tag{4.7}$$

$$u_{rel} = R/\sqrt{\tan^2 \phi + 1 + \tan^2 \phi / \tan^2 \theta} \tag{4.8}$$

$$v_{rel} = R/\sqrt{\tan^2\theta + 1 + \tan^2\theta/\tan^2\phi} \tag{4.9}$$

Note that the current origin of coordinates is the center of the rectangle image, while the origin of coordinates of pixel coordinates is the left-up vertex of the rectangle image, so the corresponding pixel in the original image of our final image pixel $p(i, j)$ is:

$$u = \begin{cases} x_0 + u_{rel}, & \text{if } \phi < \pi/2 \\ x_0 - u_{rel}, & \text{if } \phi > \pi/2 \end{cases} \tag{4.10}$$

$$v = \begin{cases} y_0 + v_{rel}, & \text{if } \theta < \pi/2 \\ y_0 - v_{rel}, & \text{if } \theta < \pi/2 \end{cases} \tag{4.11}$$

Finally, for each output pixel (i, j), the corresponding input pixel is (u, v).

5

Results

In this chapter, we present the performance and uncertainty evaluation of our used models on two different datasets, KITTI and Volvo Truck dataset.

For the performance part, we first tested our four models. They are trained and tested on the two datasets respectively. After that, we selected the best two models and tested their performance in different scenarios on the Volvo Truck dataset. Besides the basic test set recorded in the sunny daytime, we selected four additional test sets from Volvo Truck dataset recorded in different conditions, including night, rainy day, rainy night, and snowy day.

For the uncertainty evaluation, we test our non-deterministic models on Volvo Truck dataset. We compare the result with the deterministic models. Besides, we assume that the data taken from sunny daytime contain less noise than the data taken from rainy nights. Thus, we compare the uncertainty given by non-deterministic models and show the uncertainty is higher on rainy nights, which proves our method detects the noise successfully.

5.1 Evaluation Metrics

Given that velocity estimation is a regression task, we calculated mean absolute error (MAE) and root mean squared error (RMSE) to evaluate the performance of our methods.

5.1.1 Mean Absolute Error

MAE is defined as:

$$\text{MAE} = \frac{\sum_{i=1}^{n} |v_i - \hat{v}_i|}{n} \tag{5.1}$$

MAE is a commonly used metric in machine learning to measure the average errors between the ground truth and the predicted values from models. It provides a simple and straightforward quantitative description of the error magnitude, without considering the direction or sign. Actually, MAE has several advantages evaluating the performance of our predictive model:

- Simplicity and Interpretability: The value calculated from MAE has the same

units as the estimated variables, which makes it easy to understand the meaning of the value.

- Reduced Magnitude Sensitivity: Unlike other error metrics like Mean Absolute Percentage Error (MAPE), MAE is not sensitive to the magnitude or scale of predictions and ground truth. It treats all errors equally, which makes it a suitable metric to evaluate models that have a wide predictive range.
- Multiple Model Comparison: MAE is a clear measurement for comparing the performance of models having different architectures. It enables straightforward comparison and decision-making. That is to say, the best model is the one that has the least MAE.

5.1.2 Root Mean Square Error

RMSE is defined as:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(v_i - \hat{v}_i)^2}{n}} \tag{5.2}$$

Besides MAE, we also used RMSE for performance evaluation because this makes it more comprehensive. It has the following advantages when evaluating the model performance:

- Large Error Sensitivity: Due to the squaring operation in the calculation, RMSE put more emphasis on larger errors.
- Interpretability: Unlike Mean Square Error (MSE), RMSE has the same unit as the variables being predicted, which makes it easy to interpret its meaning.

The reason why we combined both MAE and RMSE for performance evaluation is that RMSE is a good complement to MAE. As we mentioned before, MAE treats all errors equally, which means MAE is not sensitive to large errors. While RMSE is more sensitive to large errors, because it takes into account the variance of errors. To be specific, RMSE amplifies the influence of large errors due to square operation in the calculation. So by comparing the RMSE of models, we are able to compare the stability of models. This is because less RMSE compared to MAE means a model does not produce large errors during the test.

5.2 Model Performance

In this section we present both visualization and performance metrics of test results. To make plots more clear, we down-sampled when plotting results on Volvo Truck dataset due to its large size, that is, we selected every 100 points when plotting.

5.2.1 Basic Results on KITTI

Though KITTI dataset already contains corresponding 3D Velodyne point clouds for each frame, which could be used to obtain depth information, we only use raw

gray-scale images and velocity data, given that we only have real-time video stream as input in practical use.

Figure 5.1 is the visualization of the test result of Raft-Projector. It shows a relatively low ability on predicting forward velocity with a pretty low accuracy, and is completely incapable of estimating leftward velocity. This is proved by our performance metrics. In Table 5.1, we can find that Raft-Projector has large MAE and RMSE on the estimation of both velocities. The large MAE means Raft-Projector produces large errors during estimation. The RMSE is similar to MAE, which means Raft-Projector does not produce significant outliers.

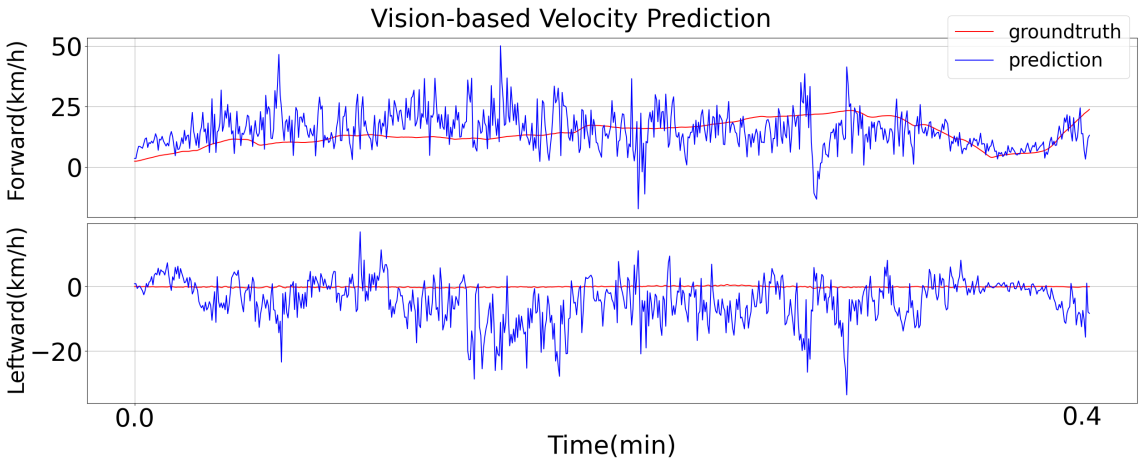


Figure 5.1: Raft-Projector Test Result On KITTI

Figure 5.2 is the visualization of the test result of Pose-Projector. From the plot we can see it works better than Raft-Projector. It has a higher accuracy on predicting forward velocity, and also shows the ability to estimate the leftward velocity yet with low accuracy. In Table 5.1, we can see that Pose-Projector has large MAE and RMSE on forward velocity and a small MAE and RMSE on leftward velocity. This proved our observation. The RMSE is similar to MAE, which means Pose-Projector does not produce significant outliers.

Figure 5.3 is the visualization of the test result of Depth-Projector. From the plot we can see Depth-Projector has the best performance. It works well on predicting forward velocity, but is still inaccurate when estimating leftward velocity. In Table 5.1, we can see that Depth-Projector has smaller MAE and RMSE. This is consistent with our observation. The RMSE is similar to MAE , which means Pose-Projector does not produce significant outliers and it is relatively robust.

5.2.2 Basic Results on Volvo Truck dataset

Here is the performance of our model on basic Volvo Truck test dataset. As 4.5 shows, the basic Volvo Truck test dataset was recorded on a sunny day.

Figure 5.4 is the visualization of the test result of Raft-Projector. From the plot we can see Raft-Projector is not unable to estimate velocity on Volvo Truck dataset.

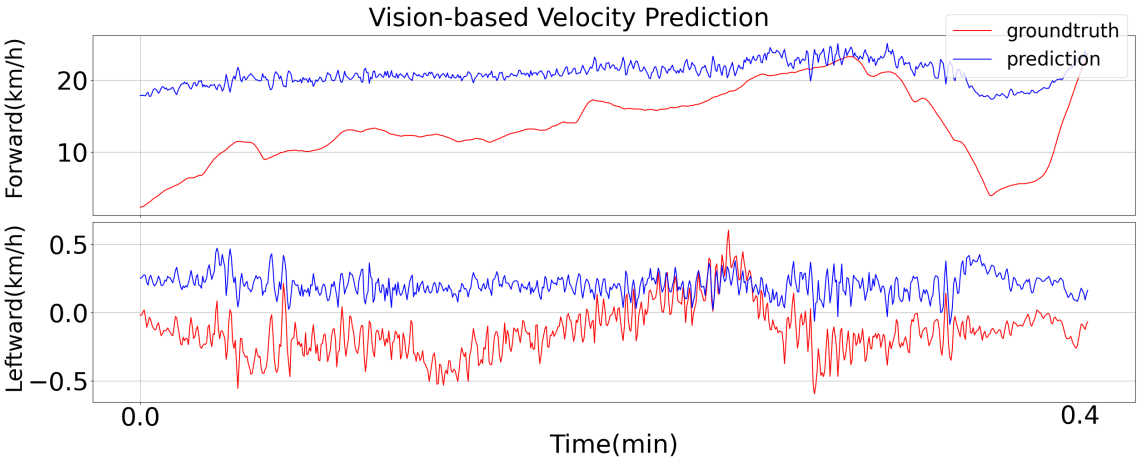


Figure 5.2: Pose-Projector Test Result On KITTI

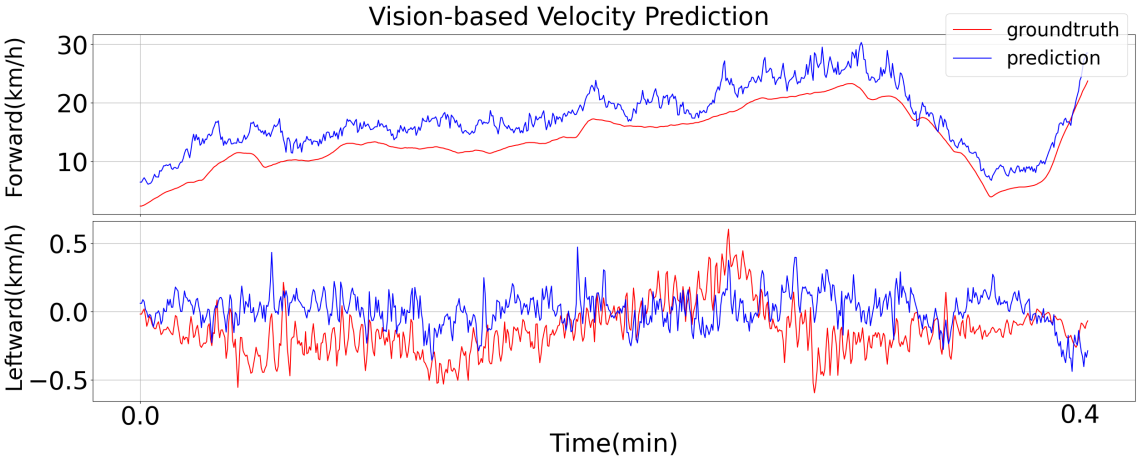


Figure 5.3: Depth-Projector Test Result On KITTI

For example, the largest forward velocity prediction is around 100(km/h), while the max value of forward velocity is about 40(km/h); the largest leftward velocity prediction is around 100(km/h), while the max value of leftward velocity is about 5(km/h). In Table 5.2, Raft-Projector has large MAE and RMSE, and RMSE is much higher than MAE. This means Raft-Projector failed to estimate velocity on basic Volvo Truck test dataset.

Figure 5.5 is the visualization of the test result of Pose-Projector. We can see that Pose-Projector is completely unable to learn the Volvo Truck dataset. In Table 5.2, we can see Pose-Projector has smaller MAE and RMSE compared to Raft-Projector, but it does not mean Pose-Projector is better than Raft-Projector. This result is reasonable because the feature extraction part, Pose, was originally trained on the KITTI dataset. And this result shows its inability for fine-tuning. So for a camera of a certain configuration, an individual pose prediction network should be first trained.

Figure 5.6 is the visualization of the test result of Depth-Projector. It has the best performance on the Volvo Truck dataset in estimating both velocities. The line of

Table 5.1: Model Performance on KITTI

(a) MAE

Model	Forward velocity(km/h)	Leftward velocity(km/h)
Raft-Projector	6.538	5.929
Pose-Projector	7.243	0.347
Depth-Projector	3.60	0.214

(b) RMSE

Model	Forward velocity(km/h)	Leftward velocity(km/h)
Raft-Projector	8.873	8.014
Pose-Projector	8.137	0.375
Depth-Projector	3.906	0.251

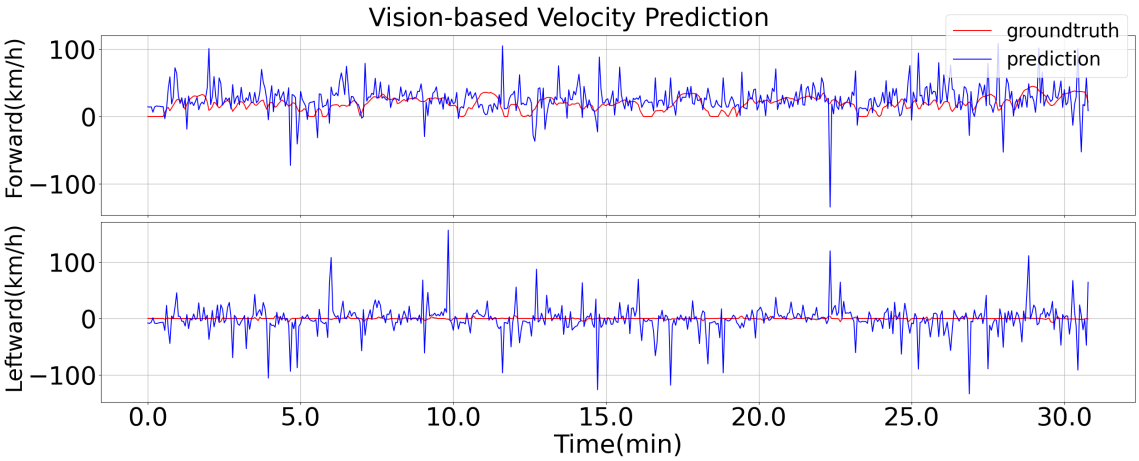


Figure 5.4: Raft-Projector Test Result On Basic Volvo Truck Test Dataset

prediction fits well with the line of ground truth. In Table 5.2, we can see Depth-Projector has small MAE and RMSE, which means it is effective and robust for velocity estimation. Another important thing is that Depth-projector even works better on Volvo Truck dataset than on KITTI.

5.2.3 Further Results On Volvo Truck dataset

According to our previous tests, we thought Depth-Projector was the solution to velocity estimation. To give a comprehensive analysis of Depth-Projector, we tested the performance of our methods according to different scenarios. As shown in Figure 4.6, we selected another 4 test datasets from Volvo Truck dataset.

Figure A.1 is the visualization of the test results of Depth-Projector on multiple Volvo Truck test sets, and Table 5.3 is the performance metrics of Depth-Projector on different test dataset.

From 4 plots in Figure A.1, we can see Depth-projector shows great generalization ability. It works well in different weather environments, although we can see a loss of accuracy compared to the performance on basic test dataset. By comparing the data within 5.3, we noticed that generally, Depth-Projector works better during

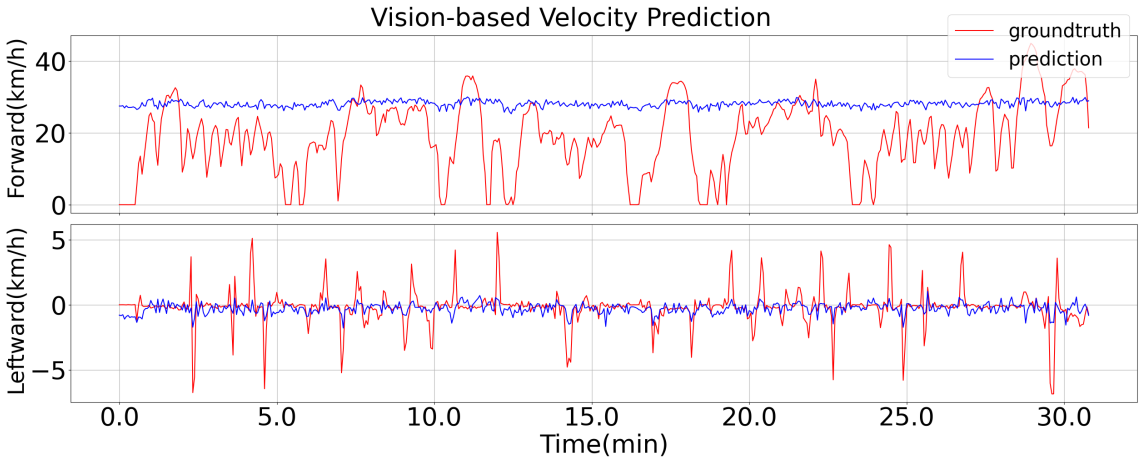


Figure 5.5: Pose-Projector Test Result On Basic Volvo Truck Test Dataset

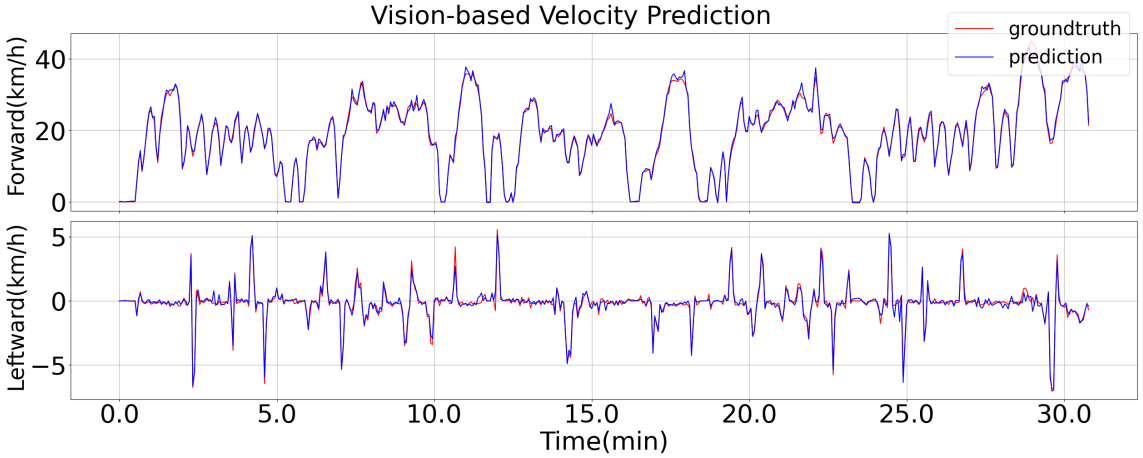


Figure 5.6: Depth-Projector Test Result On Basic Volvo Truck Test Dataset

rainy day and snowy day than during night and rainy night. One reasonable way to improve model performance could be to increase the brightness of the camera input image. One possible way is to increase the headlights' brightness. Another solution is to modify camera settings to increase camera exposure such as increasing ISO of the camera. At the same time, we also noticed that there is some blur on the camera lens. The lens is blocked by these stains and effective information in the input is reduced. So it is possible if we keep the lens clean, we might achieve the same performance as on the sunny day test.

5.3 Uncertainty Evaluation

In this section, we have two main parts to present. One is the accuracy of the prediction and the other one is the ability of uncertainty estimation, shown in Section 5.3.1 and 5.3.2, respectively.

Table 5.2: Model Performance on Volvo Truck dataset

(a) MAE

Model	Forward velocity(km/h)	Leftward velocity(km/h)
Raft-Projector	16.664	16.406
Pose-Projector	10.604	0.722
Depth-Projector	0.526	0.171

(b) RMSE

Model	Forward velocity(km/h)	Leftward velocity(km/h)
Raft-Projector	26.361	30.080
Pose-Projector	13.076	1.214
Depth-Projector	0.751	0.250

Table 5.3: Depth Performance on Multiple Volvo Truck dataset

(a) MAE

Scenarios	Forward velocity(km/h)	Leftward velocity(km/h)
Night	2.043	0.189
Rainy day	1.233	0.217
Rainy night	1.751	0.328
Snowy	1.759	0.228

(b) RMSE

Scenarios	Forward velocity(km/h)	Leftward velocity(km/h)
Night	2.994	0.266
Rainy day	1.847	0.307
Rainy night	2.482	0.509
Snowy	2.010	0.322

5.3.1 Prediction Accuracy

We test our non-deterministic model on Volvo Truck test dataset. As we did in Section 5.2.3, we test our model in five different scenarios, which are daytime, night, rainy day, rainy night, and snowy. The result is shown in Table 5.4. As we can see, the MAE is significantly higher compared to Table 5.3, i.e. our non-deterministic model sacrifices performance to gain the ability to predict uncertainty. This result is not beyond our expectations because we use the same projector to predict both velocities and uncertainty, and they affect each other while training. Unfortunately, this combination is unavoidable because the uncertainty training is unsupervised and we need the velocity prediction to assist the training process. Thus, without a better solution, we view it as a tradeoff for the model to gain the ability to predict uncertainty.

5.3.2 Uncertainty Estimation

The biggest difficulty to evaluate the uncertainty is that we lack the criterion. It's also a problem when we try to train the uncertainty model. We used a well-designed loss function to implicitly learn the uncertainty while training. When it turns to

Table 5.4: Performance of Non-deterministic model (MAE)

Scenarios	Forward velocity(km/h)	Leftward velocity(km/h)
Day	0.899	0.315
Night	1.887	0.097
Rainy day	2.554	0.395
Rainy night	2.314	0.393
Snowy	4.713	0.468

the result evaluation, however, it's unavoidable to find a criterion. Our solution is to use the residual error. The residual error usually represents the gap between the observed data and the predicted data. For example, assume the observed data is y and the predicted data is $\hat{y}$, then

$$\text{residual} = ||y - \hat{y}|| \tag{5.3}$$

In practice, we normalize the velocities to eliminate the influence of the magnitude difference. Then we draw both the residual curve and the uncertainty curve as Figure 5.7. Comparing the predicted uncertainty value with the residual, we figure out the curves have similar peaks. It means if the model makes a huge mistake, it will probably figure out and predict a high uncertainty, i.e. the model has the ability to predict its own mistake. Surely, the peaks are not one hundred percent matched, we believe it is caused by the limitation of unsupervised training. It is still exciting to make the model able to evaluate its own prediction.

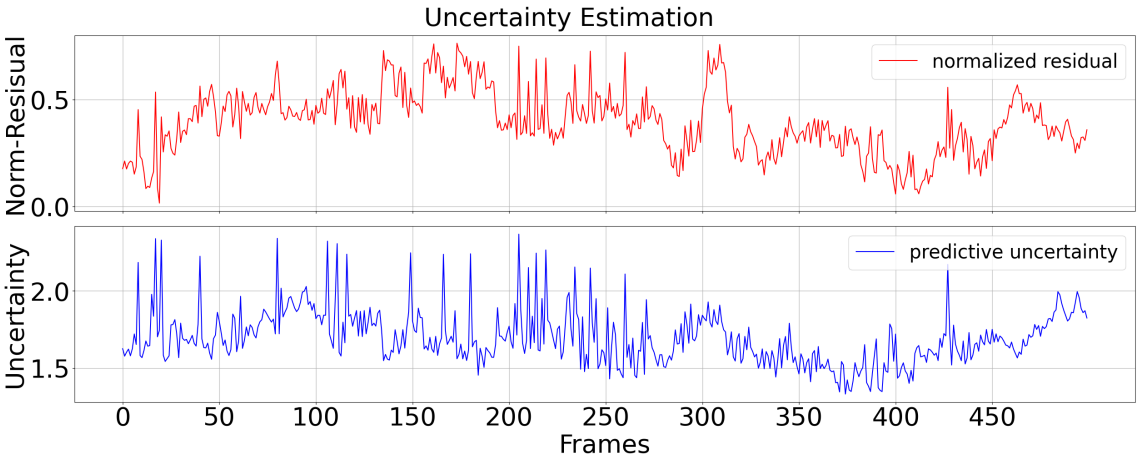


Figure 5.7: Uncertainty estimation: we take the normalized residual error as the reference. The predictive uncertainty have simliar peaks as thre normalized resisual error. It proves that the the model can predict its own "mistake".

6

Discussion

6.1 Importance Of Training Data Accuracy

As shown in the results chapter, our best model is Depth-Projector. If we compare its performance between KITTI and Volvo Truck dataset, we can find that the same model turns out to have quite different performances. This is an important lesson for training a deep learning model. When training a deep learning model, an accurate training dataset provides precise and reliable ground truth for corresponding input. If the training dataset contains inaccurate annotations, the model will learn incorrect associations, which prevents the model from finding the correct underlying patterns, resulting in bad performance. This is unacceptable for practical use because in real-world scenarios, high robustness and better performance are required, and the model should also have good generalization ability and be resilient to noise. Therefore, every company that wants to leverage deep learning to solve their practical tasks should pay more attention during the data collection process, because the quality and accuracy of the dataset will largely determine the final performance.

6.2 Drawback of Uncertainty Prediction

We use the same test dataset to test our non-deterministic model in Section 5.3. The results show that our model can predict reasonable uncertainty. However, the significant influence on performance cannot be ignored. We think this is because we use the same projector to predict both uncertainty and velocities. The uncertainty is implicitly predicted and has no explicit boundary so the prediction accuracy is hugely influenced. We also observed our predictive loss occasionally jumped during training, which prove our guess indirectly.

However, it does not mean that the non-deterministic model has no practical usage. It provides a reference to the noise contained by the input data. Thus, in practical cases, we can use both deterministic and non-deterministic models. More specifically, we can use the deterministic model to predict velocity and the non-deterministic model to estimate the noise of the input data.

Besides, in the future, if we can find a criterion to model the input noise and supervise the training process, we believe the non-deterministic model will be able to give more reasonable results without influencing the performance of velocity prediction.

6.3 Ethical Considerations

Though our method has the potential to enhance various applications in vehicle motion control, it is important to consider the ethical implications. And after careful consideration, we think our thesis and its potential future have minimal risks or harm to society. Accurate velocity estimation is crucial for making real-time decisions otherwise errors may pose significant risks to passengers' safety, and this might be the most important concern. In fact, our method acts as an auxiliary method in the whole vehicle motion control system. And at the same time, our method also self-evaluates the uncertainty of its real-time output, which also increases the transparency and accountability of our method. So our method is able to increase the reliability of the system and decrease potential risks. Another concern is the use of vehicle monocular cameras. Our method relies on real-time camera input, and this might raise concerns about privacy or security issues. Actually, in our thesis, we train our model on the legal Volvo Truck dataset and do not record real-time camera input when predicting velocity, so no breach of privacy or other sensitive data. However, since our method is an additional function according to the traditional vehicle motion control system, it consumes more energy when computing on powerful hardware and thus increases greenhouse gas emissions, which should be carefully taken into account from an ecological point of view.

7

Conclusion

The aim of our project is to develop a monocular vision-based ego vehicle velocity estimation method and to estimate the uncertainty of our method. As a result of our thesis, we proposed an efficient technique that is able to perform accurate velocity prediction and gives uncertainty evaluation for every prediction.

In our experiments, we leveraged transfer learning as our basic method. We designed 3 different models: Raft-Projector, Pose-Projector, and Depth-Projector. and evaluate their performance on two different datasets: KITTI and Volvo Truck datasets. After the experiments, we found out that Depth-Projector has the best performance on both datasets. On KITTI, Depth-Projector can achieve 3.60(km/h) MAE and 3.906(km/h) RMSE for forward velocity estimation, and 0.214(km/h) MAE and 0.261(km/h) for leftward velocity estimation. On Volvo Truck dataset, Depth-Projector can achieve 0.526(km/h) MAE and 0.751(km/h) RMSE for forward velocity estimation, and 0.171(km/h) MAE and 0.250(km/h) RMSE for leftward velocity estimation. Based on these test results, we believe Depth-Projector is a powerful technique for our task.

Furthermore, considering our method might be used for practical scenarios, reliability and robustness are crucial. To quantify this feature, we combined our best model with probability theory and enable the model to give confidence in its prediction, i.e. estimate the uncertainty. After the experiment, although the accuracy of velocity prediction was influenced, we found the uncertainty estimation results met our expectations. In practical use, both deterministic and non-deterministic models can be used together, which means the deterministic model predicts the velocities while the non-deterministic model predicts the uncertainty. By taking the velocities and the uncertainty given by our model as a reference, the vehicle motion control system is able to have a better understanding of the environment, enabling more reliable decision-making. This gives our model practical meaning.

Bibliography

[1] V. A. Thurmond, "The point of triangulation," *Journal of nursing scholarship*, vol. 33, no. 3, pp. 253–258, 2001.

[2] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[3] B. Müller, J. Reinhardt, and M. T. Strickland, *Neural networks: an introduction*. Springer Science & Business Media, 1995.

[4] Q. Ke and T. Kanade, "Transforming camera geometry to a virtual downward-looking camera: Robust ego-motion estimation and ground-layer detection," in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, IEEE, vol. 1, 2003, pp. I–I.

[5] B. K. Horn and E. Weldon Jr, "Direct methods for recovering motion," *International Journal of Computer Vision*, vol. 2, no. 1, pp. 51–76, 1988.

[6] K. Yamaguchi, T. Kato, and Y. Ninomiya, "Vehicle ego-motion estimation and moving object detection using a monocular camera," in *18th International Conference on Pattern Recognition (ICPR'06)*, IEEE, vol. 4, 2006, pp. 610–613.

[7] M. Kampelmühler, M. G. Müller, and C. Feichtenhofer, "Camera-based vehicle velocity estimation from monocular video," *arXiv preprint arXiv:1802.07094*, 2018.

[8] S. Umamaheswaran, M. Nair, A. Z. Joseph, N. G. S. S. Srinath, C. L. Priyanka, and P. Sankaran, "Stereo vision based speed estimation for autonomous driving," in *2019 International Conference on Information Technology (ICIT)*, IEEE, 2019, pp. 201–205.

[9] Z. Song, J. Lu, T. Zhang, and H. Li, "End-to-end learning for inter-vehicle distance and relative velocity estimation in adas with a monocular camera," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 11 081–11 087.

[10] H. L. Bandari and B. B. Nair, "An end to end learning based ego vehicle speed estimation system," in *2021 IEEE International Power and Renewable Energy Conference (IPRECON)*, IEEE, 2021, pp. 1–8.

[11] A. M. Mathew and T. Khalid, "Ego vehicle speed estimation using 3d convolution with masked attention," *arXiv preprint arXiv:2212.05432*, 2022.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[13] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, IGI global, 2010, pp. 242–264.

[14] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, no. 1, pp. 1–40, 2016.

[15] D. Erhan, A. Courville, Y. Bengio, and P. Vincent, "Why does unsupervised pre-training help deep learning?" In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2010, pp. 201–208.

[16] S. S. Beauchemin and J. L. Barron, "The computation of optical flow," *ACM computing surveys (CSUR)*, vol. 27, no. 3, pp. 433–466, 1995.

[17] C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow, "Digging into self-supervised monocular depth estimation," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 3828–3838.

[18] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of The 33rd International Conference on Machine Learning*, M. F. Balcan and K. Q. Weinberger, Eds., ser. Proceedings of Machine Learning Research, vol. 48, New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1050–1059. [Online]. Available: `https://proceedings.mlr.press/v48/gal16.html`.

[19] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?" In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/2017/file/2650d6089a6d640c5e85b2b88265dc2b-Paper.pdf`.

[20] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, *Weight uncertainty in neural networks*, 2015. arXiv: `1505.05424 [stat.ML]`.

[21] Y. Gal and Z. Ghahramani, *Bayesian convolutional neural networks with bernoulli approximate variational inference*, 2016. arXiv: `1506.02158 [stat.ML]`.

[22] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, Springer, 2020, pp. 402–419.

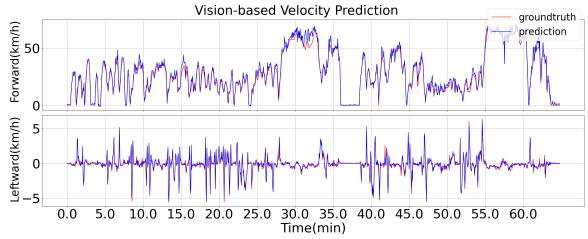
[23] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[24] L. Wei, S. Zhou, P. Zhang, and S. Sun, "Double longitude model-based correction method for fish-eye image distortion," *Chinese Journal of Scientific Instrument*, vol. 36, no. 2, pp. 377–385, 2015.

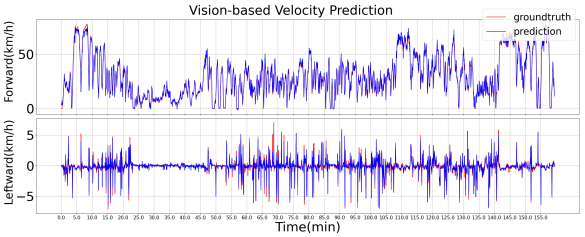
[25] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

A

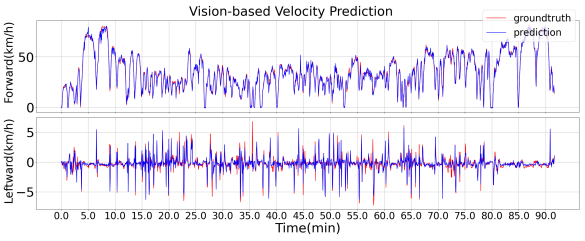
Appendix 1



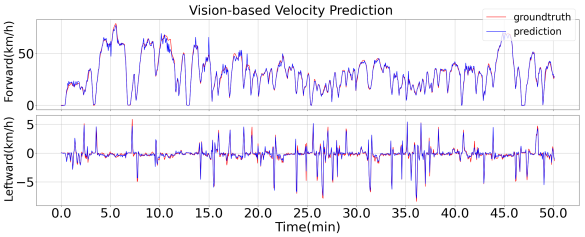
(a) night



(b) snowy



(c) rainy night



(d) rainy day

Figure A.1: Depth performance on different scenarios dataset