

Estimation On Vehicle Motion Based On Video

Group 20

Songjie Chen, Wenchang Wang, Yuchen Tang, Binglin Zhao

Abstract— In this project, we analyzed two different algorithms used to compute optical flow. At the same time, we applied different algorithms and machine learning models to calculate speed depends on optical flow. Except for existing algorithms and model, we also modified the old machine-learning algorithms to try to bring better results and compare the performance of those algorithms.

I. INTRODUCTION

Recently, the self-driving car technology has been rapidly developing. Self-driving currently allows the vehicle to take over the driving task most of the time, but still requires a driver to sit behind the wheel preparing to take over in certain situations. Waymo, the self-driving car project under google, Uber, and many other industries have already put various models into testing. In 2017, Waymo reported only 63 driver engagements in 350 thousand miles of driving. These self-driving taxis are already able to send passengers in urban areas.

Self-driving has many benefits. It can improve safety once the technology is fully developed, by eliminating human error like aggressive driving or distractions caused by fatigue. Computers also react much faster than humans in the face of accidents and contingencies. Self-driving can also reduce labor costs and insurance costs. It also has the potential to maximize fuel efficiency by moderating the drive cycle. Given all that's mentioned above, self-driving has become one of the most perspective fields of research in the present day.

The self-driving system used various sensors, and among these sensors a camera is used to record the surrounding environments and predict

the speed and position of the vehicle. This project is focused on the camera sensor. We intend to use optical flow, which is the apparent motion of an object, surface or edge between picture frames resulted from the relative movement between the camera and the environment.

Currently, there are many algorithms regarding depth perception and optical flow. This project is to improve classical methods so that they are more accurate and more resistant to noises that are caused by light conditions or weather.

Our inputs contain the KITTI dataset with ground truth, stimulation data with ground truth, and a self-recorded dash video. We then used the convolutional neural network to calculate the speed, and compared our performance to multiple classical methods.

II. RELATED WORK

This project is inspired by these following works.

Our inspiration and starter code is provided by Jonathan Mitchell. Also, in “*Non-local Total Generalized Variation for Optical Flow Estimation*”, by René Ranftl, Kristian Bredies and Thomas Pock, they introduced a higher-order regularization term. This regularization term is built on top of the classical second-order Total Generalized variation that generates piecewise affine solutions and is able to combine soft-segmentation cues into the regularization term. In their work, these authors also used the KITTI benchmark as their input. In our project, in addition to the KITTI dataset, which provides accurate ground truths, we also implemented image strings captured by Airsim, which allows us to alter the street condition in our simulation.

One other work we referred to is the “*Large Displacement Optical Flow: Descriptor Matching in Variational Motion Estimation*”, by Thomas Brox, Jitendra Malik, Fellow, IEEE. In contrast to their focus on compensating the failure of the coarse-to-fine heuristics in predicting large movements, we focus on improving the accuracy and resistance to noise, which is more important in

the case of self-driving cars.

We also reviewed the research "SimpleFlow: A Non-iterative, Sublinear Optical Flow Algorithm" by Michael W. Tao, Jiamin Bai, Pushmeet Kohli, and Sylvain Paris. Yet we have a different goal. While the SimpleFlow algorithm uses a relatively sparse sample set to compute the flow in regions that contain relatively smooth movements to minimize the running time and cost, we tend to "patch" existing methods to improve performance.

Our methodology is inspired by "FlowNet: Learning Optical Flow with Convolutional Networks" by A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. Smagt, D. Cremers, T. Brox. Their work suggested the potential of convolution neural networks (CNNs) in optical flow. Convolution neural network has been successful in many computer vision fields other than optical flow, and their work provided insights on introducing this method into optical flow.

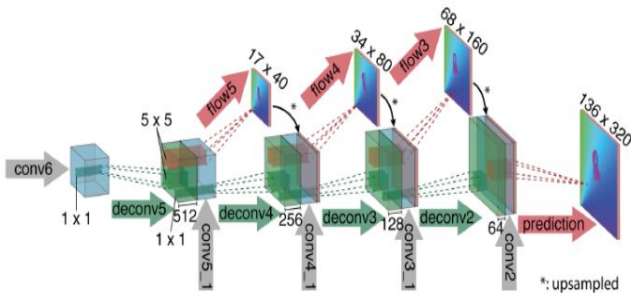


Figure 1. Demonstration of the FlowNet

III. DATASET AND FEATURES

In our project, we implement our training with two different types of datasets. One is the existing complete dataset KITTI[3], a brief introduction of this dataset is displayed below:

KITTI takes advantage of the autonomous driving platform Anniway to develop novel challenging real-world computer vision benchmarks. The dataset including the data of stereo, optical flow, visual odometry, 3D object detection and 3D tracking. KITTI dataset is captured by driving around the mid-size city of Karlsruhe, in rural areas and on highways. Up to 15 cars and 30 pedestrians are visible per image.

The advantages of KITTI are: We have very accurate ground truth provided by GPS localization system and Velodyne laser scanner.; KITTI contains all raw data and provide validate platform. A figure is shown below as an example of KITTI dataset.

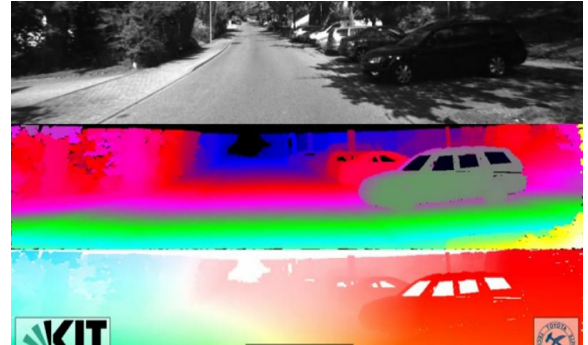


Figure 2. KITTI dataset with ground truth, optical flow

Another dataset we use is picture/video captured from AirSim[2]. A brief introduction of AirSim is displayed below:

AirSim is created by Microsoft AI & Research, it is an open-source simulator for autonomous systems. It is a platform comprised of realistic environments and vehicle dynamics that allow for experimentation with deep learning, computer vision and AI.

The advantages of using AirSim are that we can control variables such as the number of cars, the speed and direction of every vehicle. Also, another advantage is that there is no pedestrian to influence the result. Because the AirSim is not a complement data, we have to simulate our own dataset with the desired environment. Once finished setting up the environment, we used the built-in recording function to record video clips. The video clips are split to images with desired same time interval as our input data set. An example of AirSim simulator is shown below.

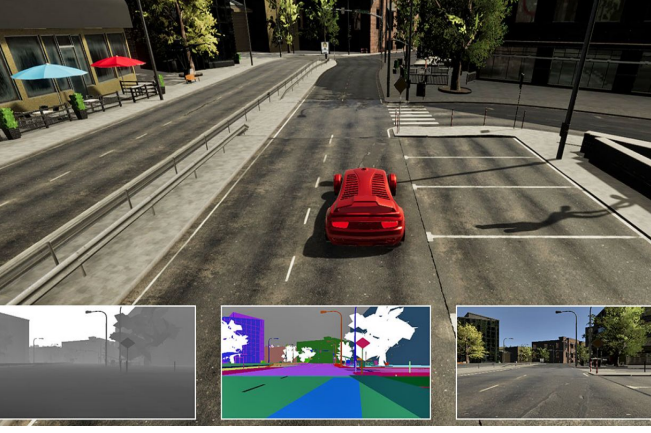


Figure 3. Sample AirSim dataset

IV. METHODS

In order to complete our goal. We implemented two different methods to calculate the optical flow. One is sparse optical flow(Lucas-Kanade algorithm) and the other one is dense optical flow(Sublinear Optical Flow algorithm). When it comes to calculating vehicle speed, we need to add extra calculation based on optical flow result. For Lucas-Kanade optical flow algorithm, we calculate the speed vector by using the least square principles. We first write every pixel inside the window in q_1, q_2, \dots, q_n , then calculate $I_x(q_i)$, $I_y(q_i)$, $I_z(q_i)$, where there is the partial derivative of the image I with respect to position x , y , and time t_i evaluated at the point q_i and at the current time. Then we rewrite them in matrix form $Av = b$, where:

$$\begin{aligned} I_x(q_1)V_x + I_y(q_1)V_y &= -I_t(q_1) \\ I_x(q_2)V_x + I_y(q_2)V_y &= -I_t(q_2) \\ \cdot & \\ \cdot & \\ \cdot & \\ I_x(q_i)V_x + I_y(q_i)V_y &= -I_t(q_i) \end{aligned}$$

Since the system is overdetermined, the matrix can be solved by least squares principle.

The other method of calculating vehicle speed is a little bit more complicated. We use the convolutional neural network (CNN) to calculate speed. There are three parameters need to be fed to the network.(magnitude, hue and saturation). Except for those methods we learned through paper, we

tried to two more parameters to train the CNN module. In the experiment we will test and compare their performances. The detail of each optical flow algorithm and speed calculation algorithm will be discussed individually in the following section.

1. Lucas-Kanade algorithm

Generally, inputs of the Lucas-Kanade algorithm with sequences of images (subsequent frames from a video), it should output an optical flow field and trace the motion of the moving objects. It is achieved by minimizing the mean square error of brightness between two images[4].

Let $T(x)$ to be a template image and $I(x)$ be an input image with $x = (x, y)^T$ as a column vector [1]. For example, when we computing optical flow from time $t = 1$ to time $t = 2$ in a sequence of images, $T(x)$ should be a self-defined sub-region of the image at time $t = 1$ and $I(x)$ should be an image at time $t = 2$.

Define vector of parameter $p = [p_1, p_2]^T$ be the optical flow. Let vector $W(x; p)$ be the parameterized set of warps that takes the pixel in the image of the template. And maps the frame to the sub-pixel location.

$$W(x; p) = \begin{pmatrix} x+p_1 \\ y+p_2 \end{pmatrix} \quad (1)$$

With this equation, what we need to do is to minimize the sum of squared error between two images T and I .

$$\sum_x [I(W(x; p)) - T(x)]^2 \quad (2)$$

The summation is performed over every pixel x in the template image. With respect to equation above, assume that Δp is known, then we can update the parameter iteratively:

$$p = p + \Delta p \quad (3)$$

Repeating step (2) and step (3) iteratively until p converges:

$$|\Delta p| \leq \epsilon \quad (4)$$

2. Sublinear Optical Flow algorithm

Using the color difference between pixels, Simple Flow algorithm can compute probabilistic representation of motion at each pixel which is optical flow. In the computation, it only uses representative local evidence without resorting to global optimization. Instead of matching every pixel on one frame to another based on color, Simple Flow considers blocks of pixels have more discriminative features.[5]

For frame I_t and I_{t+1} . Define (x, y) as pixel position and (u, v) as flow vectors. Firstly, we want to find flow vector (u, v) make scene point (x, y) in I_t is located at $(x + u, y + v)$ in I_{t+1} . To realize this step we want to minimize the error between estimated result and ground truth:

$$E(x, y, u, v) = |I_t(x, y) - I_t(x + u, y + v)|^2$$

Assuming optical flow is smooth, Simple Flow finds flow vectors at each pixel (x_0, y_0) which can represent motion at both pixel (x_0, y_0) and other pixels in a neighborhood N_0 .

$$(u_0, v_0) = \arg \max_{(u, v) \in \Omega} \prod_{(x, y) \in N_0} p(x, y, u, v)$$

Two weighting function is added to compute the relativity between pixels:

$$w_d = e^{-|(x_0, y_0) - (x, y)|^2 / (2\sigma_d)}$$

where w_d is the distance between pixels.

$$w_c = e^{-|I_t(x_0, y_0) - I_t(x, y)|^2 / (2\sigma_c)}$$

and w_c is the color difference between pixels.

Result can be then solved from the following log-likelihood equation:

$$E(x_0, y_0, u, v) = \sum_{(x, y) \in N_0} W_d W_c e(x, y, u, v)$$

3. Three-Parameter Convolutional Neural Network(CNN)

Parameters in the model can be divided into three layers: Sublinear Optical Flow, hue and brightness differences between consecutive picture frames.[6] Such Algorithm are taking into consideration of hue and brightness. Therefore, such method shows robustness on different weather conditions as well as road conditions. The detail of the structures will be shown in implementation section.

4. Five-Parameter Convolutional Neural Network(CNN)

Different from three parameters mentioned above, we added two more parameters to the CNN model. Instead of only considering Hue, brightness we also take the saturation difference in three RGB channels into consideration.

V. EXPERIMENT

1. Algorithm model

In our project, we have done extensive experiments on our algorithms and adjust parameters to produce the best result on both validation sets and testing sets. Firstly, we built model for each algorithm and apply them on video dataset. Then we compared and evaluated each algorithm performance by their accuracy and efficiency. Accuracy is calculated by comparing the final difference between the calculated speed and ground-truth speed. Efficiency is simply calculated by counting the runtime of each algorithm when they run on the same dataset. With the accuracy and efficiency of each algorithm, we can cross-compare the result and check if they meet our expectations. Besides, we also used the result to determine if our self-modified module is successful or not.

A. Sparse optical flow

When implementing naive Lucas-Kanade Algorithm we used Open-Cv resources. The size of the search window at each pyramid level is set to 15*15 to reach the best performance according to our input video set.

```
lk_params = dict(winSize_=(15, 15),
                maxLevel_ = 2,
                criteria_=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

feature_params = dict(maxCorners_ = 500,
                    qualityLevel_ = 0.1,
                    minDistance_ = 7,
                    blockSize_ = 5 )
```

Figure 4. Parameter setting for Lucas-Kanade Algorithm

With this naive method, we implemented Pyramid Lucas-Kanade algorithm application with higher accuracy and robustness. It is accomplished by computing pyramid representation of a generic image I recursively.

Generally, first compute optical flow on deepest Pyramid layer L_m . Then, the result is propagated to the upper level L_{m-1} and replace the original pixel. Keep doing this till the top layer to get final optical flow result [7]. This method can be implemented by function [cv2.calcOpticalFlowPyrLK\(\)](#) from openCV.

B. Dense optical flow

Though, Lucas-Kanade algorithm performs pretty good under conditions of low speed. We want our system to perform more robust under different conditions of brightness as well as different road conditions. In that case, we introduce the following three parameters-CNN with the following architecture[6]:

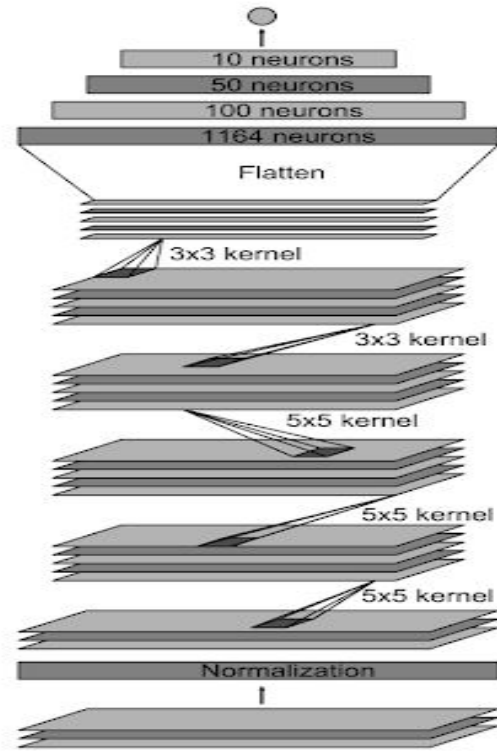


Figure 5. CNN model structure

We use three different input parameters to find the best vision cue input for our network.

B.1 Mean Square Error to find best vision cue input

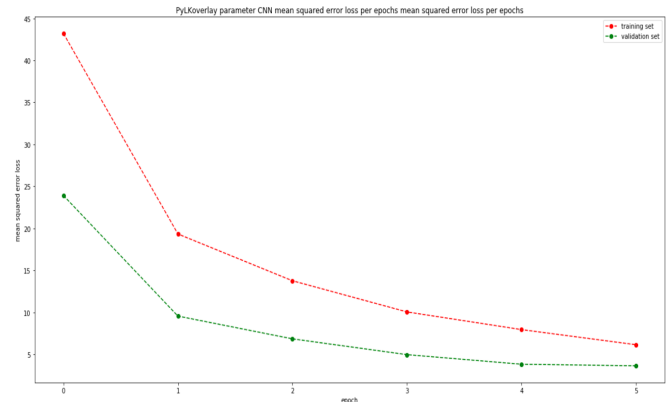


Figure 6. Pyramid Lucas-Kanade Optical Flow overlay(with 3 Dimensional parameter)

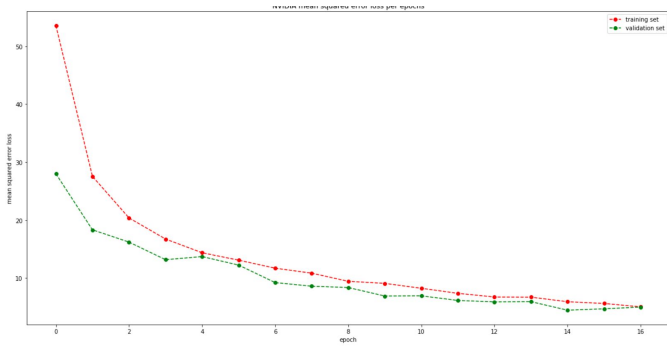


Figure 7. Sublinear Optical Flow Algorithm(with 3 Dimensional parameter)

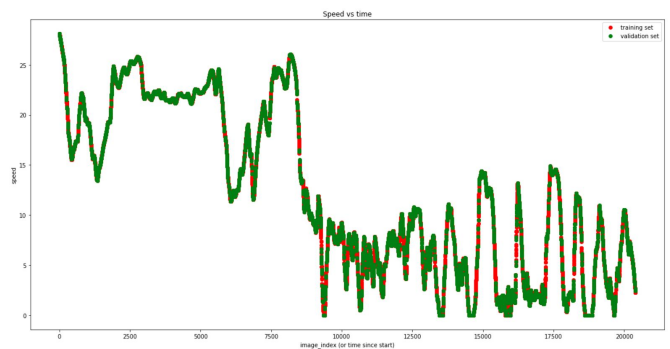


Figure.10 Sublinear Optical Flow Algorithm -- 3 Dimensional parameter

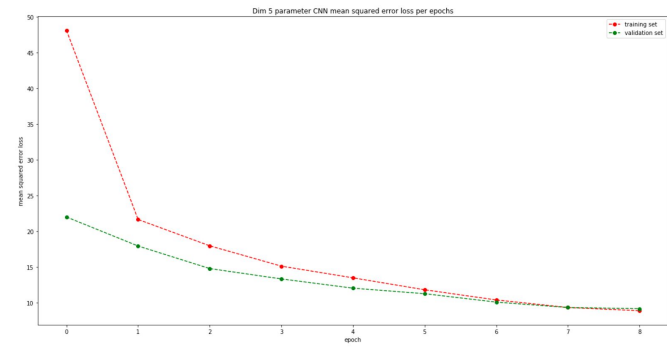


Figure 8. Five image cue differences -- 5 Dimensional parameter

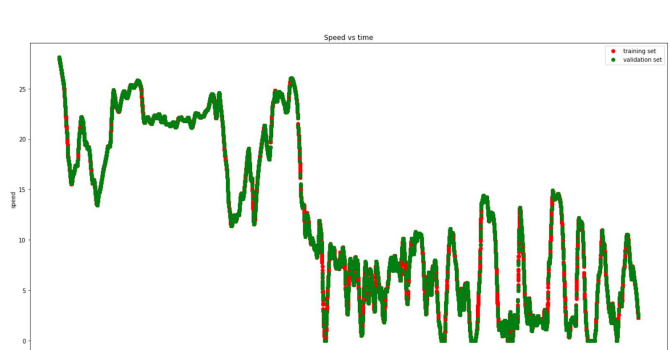


Figure.11 Five image cue differences -- 5 Dimensional parameter

3. Algorithm output

Split the training into train and validation

3.2 Output: Speed vs. Time

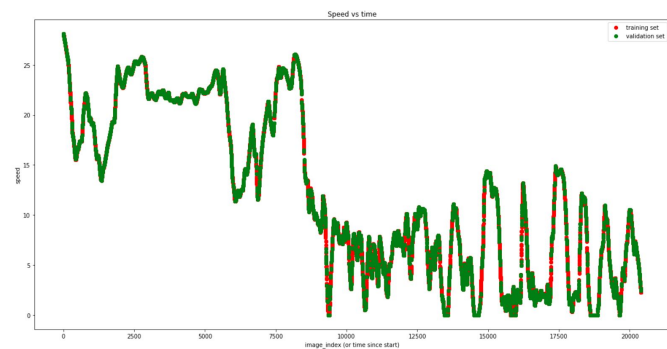


Figure.9 Pyrimid Lucas-Kanade Optical Flow overlay -- 3 Dimensional parameter

3.2. Algorithm output on training data on CNN

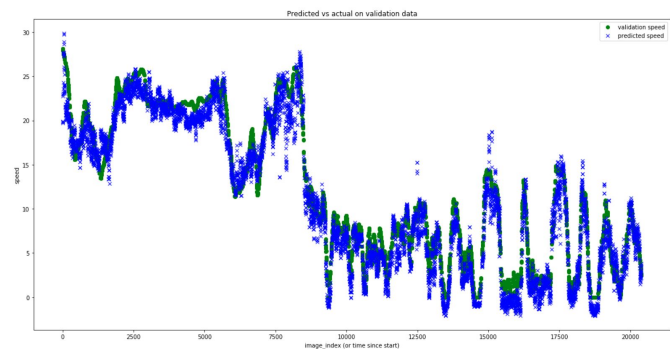


Figure 12. Pyrimid Lucas-Kanade Optical Flow overlay -- 3 Dimensional parameter

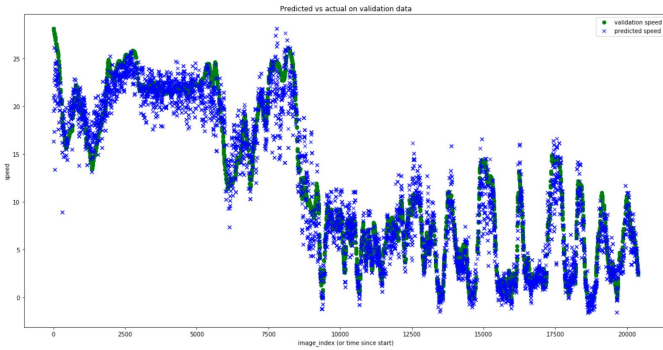


Figure 13. Sublinear Optical Flow Algorithm -- 3 Dimensional parameter

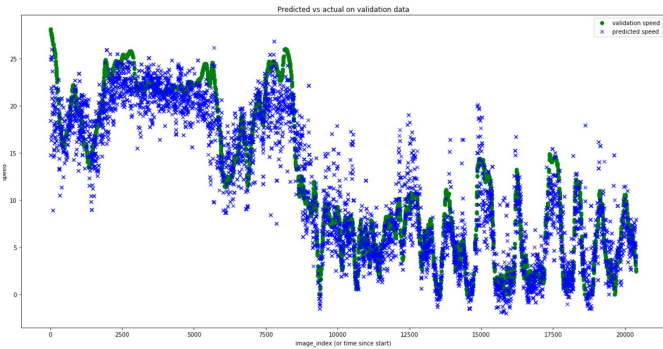


Figure 14. Five image cue differences -- 5 Dimensional parameter

3.3. Algorithm output on predicting test data

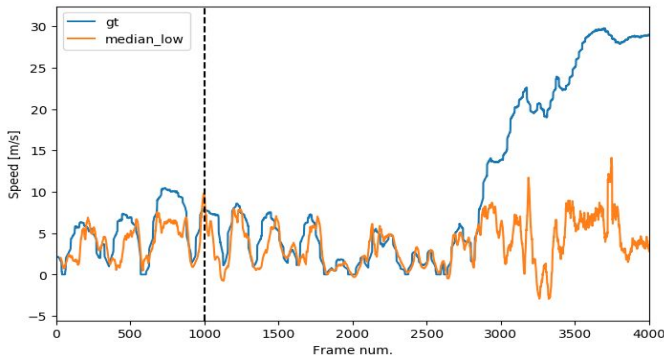


Figure 15. Lucas-Kanade output with given data

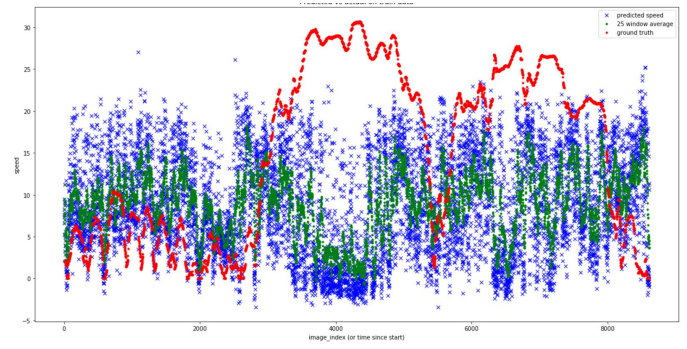


Figure 16. Pyrimid Lucas-Kanade Optical Flow overlay -- 3 Dimensional parameter

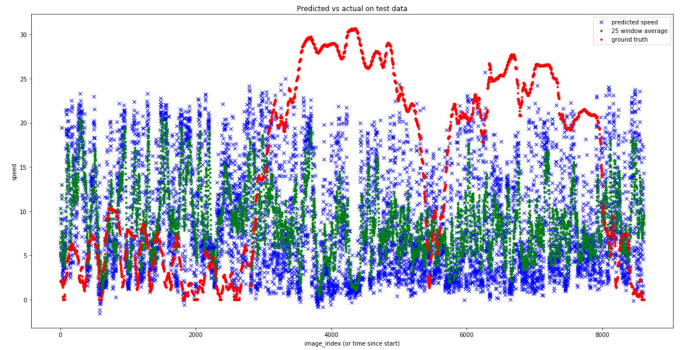


Figure 17. Sublinear Optical Flow Algorithm -- 3 Dimensional parameter

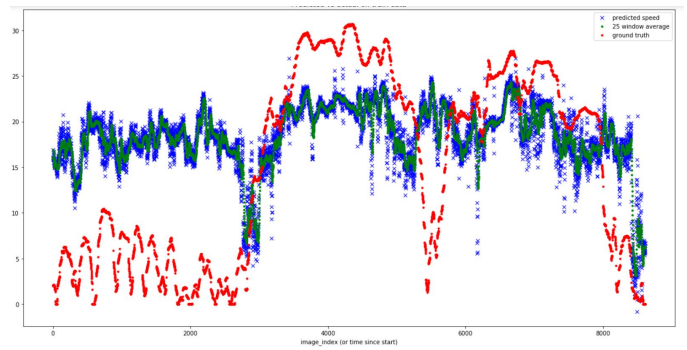


Figure 18. Five image cue differences -- 5 Dimensional parameter

VI. CONCLUSION & FUTURE WORK

After comparing all these four algorithms (Lucas-Kanade naive method, Pyramidal Lucas-Kanade Optical Flow method, Sublinear Optical Flow Algorithm and Five image cue differences method) performance on predicting speed based on vision cue. We found that as we change the feature from images to another, the training process has some really interesting things revealed. Lucas-Kanade naive method is very

robust under conditions of low speed as well as suitable weather and illumination condition.

As the counterpart, using Pyramidal Lucas-Kanade Optical Flow method to feed CNN network also shown some advantage in training processes. It converges much faster than the rest of the feature extraction methods and dense optic flow extraction methods.

Finally, Sublinear Optical Flow Algorithm takes hue and brightness into consideration. Therefore, such a method shows robustness on different weather conditions as well as road conditions. When working on the CNN model, we tried to increase features and layers of the network by adding 2 more RGB saturation difference. However, the increase of the input dimensions in the neural network didn't improve the accuracy of the whole prediction on various datasets.

In the future, we might collect more physical datasets to help us with predicting the speed. Other than that, if we can find a dataset with ground truth of vehicle moving direction, we can use speed information and optical flow to predicting vehicle moving direction. In that case, we can control and assist driving for the drivers with more features.

VII. CONCLUSION

Chen, Songjie:

1. Work on the Lucas-Kanade naive method implementation
2. Find and Developed CNN architecture as well as ensure correct implementation
3. Find and Developed Pyramidal Lucas-Kanade Optical Flow overlay
4. Find and Developed Sublinear Optical Flow Algorithm -- 3 Dimensional parameter into CNN
5. Find and Developed Five image cue differences -- 5 Dimensional parameter into CNN
6. Train all the model and make predictions on training data with ground truth

Tang, Yuchen:

1. Setup the dataset environment for KITTI, setup the dataset environment for AirSim simulator built in Unity3D.
2. Retrieve dataset by using AirSim simulator, use the built-in recording function in AirSim and cut clips in images as input dataset.

Binglin Zhao:

1. Researching on the algorithms could be used to compute optical flow.
2. Brainstorm on improvement of CNN model.
3. Implementing Lucas-Kanade naive method on video dataset.
4. Implementing Pyramidal Lucas-Kanade Optical Flow.
5. Comparing results of different algorithms.

Wenchang Wang:

1. Researching on self-driving background.
2. Brainstorm on algorithms comparison.

VIII. APPENDIX

GitHub:

<https://github.com/Jaychen233/Vehicle-motion-based-on-vision>

IX. REFERENCES

- [1] Andrea Giachetti, Marco Campani, and Vincent Torre. *The use of optical flow for road navigation*, *IEEE Transactions on Robotics*, 1999.
URL: <https://ieeexplore.ieee.org/document/660838/>
- [2] Jose De Oliveira, Rick Duong, *AirSim on Unity: Experiment with autonomous vehicle simulation*, November 14, 2018.
URL: <https://blogs.unity3d.com/2018/11/14/airsim-on-unity-experiment-with-autonomous-vehicle-simulation/>
- [3] Andreas Geiger and Philip Lenz and Raquel Urtasun, *Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite*,

2012.URL:<http://www.cvlibs.net/publications/Geiger2012CVPR.pdf>

[4]Simon Baker, Iain Matthews, *Lucas-Kanade 20 Years On: A Unifying Framework*,2004.URL:https://ri.cmu.edu/pub_files/pub3/baker_simon_2004_1/baker_simon_2004_1.pdf

[5]Michael Tao¹, Jiamin Bai¹, *SimpleFlow: A Non-iterative, Sublinear Optical Flow Algorithm*, November 2, 2012.URL:<http://graphics.berkeley.edu/papers/Tao-SAN-2012-05/Tao-SAN-2012-05.pdf>

[6] Karol Zieba,*End to End Learning for Self-Driving Cars*,April 06,2016.
URL:<https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf>

[7] Jean-Yves Bouguet,*Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the algorithm*.
URL:<https://pdfs.semanticscholar.org/aa97/2b40c0f8e20b07e02d1fd320bc7ebadfd7.pdf>