WRONG-WAY DRIVING DETECTION

by

Aphinya Chairat

A report submitted in fulfillment of the requirements for the course
AT60.993A: Special Study

Examination Committee:	Dr. Matthew N. Dailey, CSIM, AIT (Chairperson) Dr. Mongkol Ekpanyapong, ISE, AIT Dr. YOUR COMMITTEE #2
Nationality:	Thai
Previous Degree:	Bachelor of Computer Engineering Naresuan University, Phitsanulok, Thailand
Scholarship Donor:	Royal Thai Government AIT Fellowship

Asian Institute of Technology
School of Engineering and Technology
Thailand
October 2017

Table of Contents

List of Figures

Chapter 1

Introduction

Humans use their eyes to capture images and send visual information to their brains to see and visually sense the world around them. Computer vision is the science that attempts to understand the world as humans do.

1.1 Overview

According to the estimates of the World Health Organization (WHO), every year, around 1.25 million people die because of road traffic crashes. Between 20 and 50 million more people are disabled because they got in an accident on the roads. They can not do ordinary work as a result of their injuries. Moreover, oftentimes, family members have to take time from work or school to take care of them.

There are many causes of road traffic crashes. The main cause is human error or bad behaviour of drivers such as over speeding, wrong-way driving, driving under the influence of alcohol or other psychoactive substances, and non-use of motorcycle helmets or seat belts. Wrong-way driving happens when a driver wants to use the shortest path to reach a destination, regardless of traffic rules, increasing the risk of accidents.

To prevent this problem, law enforcement is a good solution, and governments already have traffic officers in place to penalize offenders. But strict and ubiquitous enforcement would require many police officers everywhere. Computer vision can enable advanced technological solutions to help improve the situation through automated wrong-way driver ticketing, as one possibility. Therefore, in this study, I consider wrong-way driving monitoring by computer vision combined with a law enforcement system to help solve the problem of wrong-way driving.

1.2 Problem Statement

According to the WHO, statistics on the number of road traffic crashes is high. Governments try to solve this problem by law enforcement, and in many places, they allow automated ticketing. One behaviour leading to accidents is wrong-way driving.

Recently, computer vision has developed many technological solutions that may help. Background subtraction (Piccardi, 2004) is one method of moving object detection. Furthermore, computer vision and image processing combined with machine learning for detection, tracking, and classification are also possible solutions. Forthoffer et al. (1996) present a new an automatic incident wrong-way vehicle detection system using image processing. There has been a great deal of research about moving object detection. For example, Paragios and Deriche (2000) present geodesic active contours

and level sets for the detection and tracking of moving objects.

Optical flow (Horn & Schunck, 1981) is another useful computer vision technology. It implements the idea of measuring the motion of objects in the scene. Monteiro et al. (2007) propose wrong-way driver detection based on optical flow. The basic idea is good; optical flow is good for estimating the direction of motion of an object in the scene. However, optical flow is not very useful for identifying an object, because it only works with points in the image sequence. So while it might work well for individual vehicles with little occlusion viewed from a high height, it is not easy to group moving points into individual objects, in more crowded scenes, and it may give wrong groupings when objects are close to each other.

One of the most important considerations in building a system for law enforcement for wrong-way driving detection is the identification of the vehicle so that we can send a ticket to the vehicle owner.

1.3 Objectives

As the main objective of my research, I would like to develop a system enabling automated detection and processing of wrong-way driving violations with minimal user set up and calibration under crowded traffic conditions. Towards the main objective, in this special study, I will:

1. Study state of the art methods for estimating motion of objects in a 2D projection of a 3D scene under crowded conditions.
2. Study state of the art methods for detecting wrong-way driving vehicles.
3. Explore existing and possible methods for detecting and ticketing wrong-way driving vehicles with a case study prototype.

1.4 Limitations and Scope

The study is limited to the following:

- Study of the possible ways to detect wrong-way driving. I only focus on motion estimation for rigid 3D objects moving parallel to a ground plane.
- The experiment is preliminary and not expected to be a production-ready solution.

1.5 Thesis Outline

I organize the rest of this study as follows.

In Chapter 2, I provide a literature review.

In Chapter 3, I document a preliminary prototype wrong-way driving detection system.

In Chapter 4, I provide recommendation for future research wrong-way driving detection.

Chapter 2

Literature Review

Many efficient techniques based on computer vision are available for classification, object detection, and tracking. Here I review existing work that will be useful for the construction of a wrong-way driving system.

2.1 Background Subtraction

Background subtraction is a commonly-used technique for generating a foreground mask. In background subtraction, a foreground mask is calculated by subtracting the current frame and a background model obtained with a fixed camera. Every object in the background model is considered as part of the background and is to be ignored.

Background modelling has two main steps:

1. Background initialization.
2. Background update, due to possible changes in the scene.

An example is shown in Figure 2.1.

Background subtraction is easy to implement, but it is slow, because background subtraction has to perform its calculation over the whole frame on every frame. Large variance in the background can lead to false negatives, and long-term scene change with a slowly-changing background may not work well.

2.2 Classification

Classification is a general technique for determining what something is (when we do not know its type) or determining where something is (when the type of object we are looking for is known).

2.2.1 Nearest Neighbor Classification

The nearest neighbor decision rule assigns a label to an unclassified sample point. The classification model simply contains a set of previously-classified points. The nearest neighbor algorithm will find the nearest item in the training set and return the label of that item. This method is simple to implement and powerful, requiring no training time. This classification method usually has the best accuracy, if the training set is large enough, but it requires too much memory and compute time with a large training set. Also, as shown in Figure 2.2, if a new data item of class 2 is closer to an item of class 1 than any item of class 2, we would classify the input incorrectly. This can be partly addressed

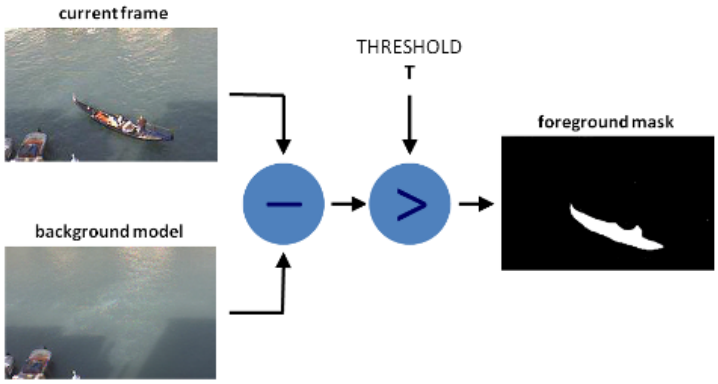

Figure 2.1: Background subtraction example from OpenCV Development Team (2017).

Figure 2.2: Example of nearest neighbor classification.

by the K-nearest neighbor method. K-nearest neighbor classification allows the user to specify K, the number of stored items to compare against. The method finds the nearest K items in the training set and returns the label of the majority within that group of items. Thus, in Figure 2.2, if K = 3, The method will classify the new item as a member of class 2.

2.2.2 Support Vector Machines

A support vector machine (SVM) is a feature-based classifier. Suppose we have a set of given data points, each belonging to one of two classes, and the goal is to decide which class a new data point is in. SVMs use linear hyperplanes for classification. There are many possible hyperplanes that might classify (separate) the training data correctly. Support vector machines choose the unique hyperplane that provides the largest separation of the training data from the hyperplane, as shown for example in Figure 2.3. When the training data are not linearly separable, the optimization of the hyperplane is more complicated, but still can be modeled as a constrained quadratic optimization. Support vector machines can induce non-linear classification boundaries due to the “kernel trick” and do not require a large amount of training data.

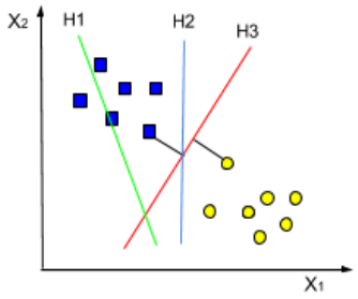

Figure 2.3: Example of large margin classification with a linear hyperplane. H1 does not separate the classes. H2 does separate the classes, but with a small margin. H3 separates the classes with a larger margin.

2.2.3 Classical Backpropagation Neural Networks

Backpropagation is a method for training multi-layer neural networks. Usually, a classical neural network would be feature based. Backpropagation over few layers usually has similar performance to SVMs. The goal of backpropagation is to optimize a set of weights so that the neural network correctly maps arbitrary inputs to a correct output. The optimization algorithm repeats a two-phase cycle, forward propagation and backward weight update. The input is propagated forward through the network, layer by layer, until it reaches the output layer. The output layer is then compared to the desired output using some loss function. The loss is propagated back from the output layer toward the input layer. This method requires experimentation to tweak hidden layer size and learning time.

2.2.4 Convolutional Neural Networks

The methods described thus far are normally feature based, requiring the data scientist to mathematically specify the set of features to be extracted from the input data. This is necessary when the training set size is small. For images, CNNs offer the benefit of automatically generated features, offering better performance than feature-based methods so long as a sufficient amount of training data is available.

A CNN usually consists of one or more convolutional layers with subsampling steps followed by one or more fully connected layers. The input to a convolutional layer is a $m \times n \times r$ image, where m is the height, n is the width, and r is the number of channels in the input. The convolutional layer contains k filters or kernels of size $p \times p \times q$, where p is smaller than the dimension of the image and q can be the same as the number of channels (r) or smaller and may be different for each kernel. The convolutional units share weights across the entire input image patch. Many open source frameworks such as Caffe allow building of CNNs. An example convolution neural network is shown in Figure 2.4.

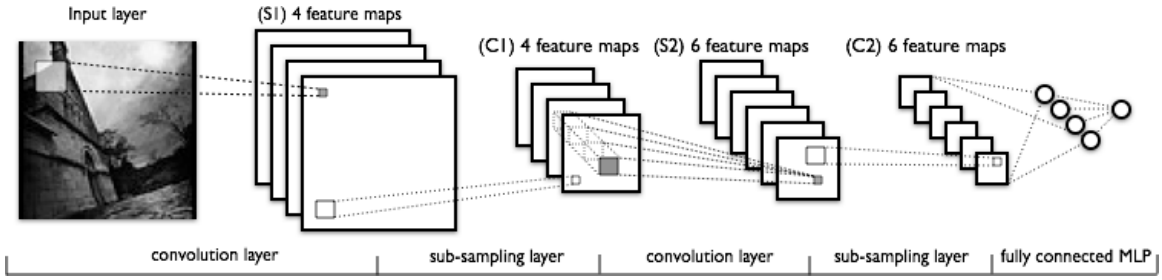

Figure 2.4: Example convolutional neural network (CNN). Reprinted from Deep Learning Development Team (2017).

2.3 Object Detection

Object detection is a very important part of a computer vision system. Detection helps us find objects in an input image. Most detection methods run a scan window over the image and classify each region as a member or not a member of the object class of interest.

2.3.1 Haar Cascades

Object detection using Haar feature-based cascades is one of the most efficient methods to detect an object in the image. Invented by Viola and Jones (2004), this method requires a medium number of input images (positive images and negative images) for training. After training, we use the resulting model to detect objects in other images.

Positive images are images of an object, and negative images are images without that object. In training, we extract the same set of features from both positive and negative images. Each feature is a single value extracted using a fixed binary kernel as shown in Figure 2.5.

A large set of possible sizes and locations of each kernel are considered, giving a huge number of possible kernels, making the training process quite heavy. The trained model might contain a large number of features, which would make it correspondingly inefficient. To increase the efficiency of feature computation, integral images are used as a memoization method enabling calculation of sums of intensities across any rectangular region in constant time regardless of block size.

The new problem is then to find which is a good feature, because a good feature should focus on the essential region and distinctive aspects of the object only. To select features, we use the Adaboost committee-building algorithm. The method applies each and every feature to all training images. For each feature, we find the best threshold for classifying the object as positive or negative over the weighted training set. Each individual feature thus added to the model is a "weak classifier" that has some level of error over its weighted training set. Different weightings of the training set give different optimal weak classifiers. The best classifier is then a weighted sum of the weak classifiers.

Applying the resulting complete "monolithic" committee of weak classifiers to every window in an image will normally achieve good accuracy but is still not a good idea, as it would be unnecessarily

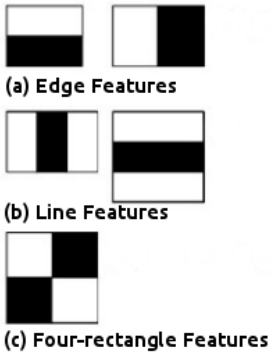

Figure 2.5: Example Haar-like kernels. The sum of the intensities of the pixels in the dark regions is subtracted from those of the light regions. Reprinted from Viola and Jones (2004).

computationally intense. To accelerate the computation, we use a cascade of classifiers to group the features into separate stages and apply each stage sequentially. If a window fails in first stage, we do not need to consider it in the remaining stages. Windows that pass though all stages are classified as an instance of the object of interest.

2.3.2 Histogram of Oriented Gradients

Histogram of Oriented Gradients (HOG) (Dalal & Triggs, 2005) is an object detection technique relying on a holistic descriptor of an image patch. HOG generates a representation of the object is contours. Examples of the same object should produce a descriptor as close as possible to the same descriptor calculated when the object is viewed under different conditions. A support vector machine (SVM) is used to recognize HOG descriptors as representing or not representing the object of interest. To recognize objects at different scales, the image is sub-sampled at multiple sizes. Each of these sub-sampled images is then searched to matches. A sample HOG classifier is shown in Figure 2.6.

2.3.3 Region-based CNNs

Region-based CNNs (R-CNNs) (Girshick, Donahue, Darrell, & Malik, 2014) combine region proposals with convolutional neural networks (CNNs). When an input image is presented, the system extracts region proposals bottom-up to localize and segment objects. A R-CNN creates region proposals using a process called Selective Search. Selective Search, regardless of the size or aspect ratio of the candidate region, warps the set of pixels in a bounding box around it to a constant size. Once the warped proposals are created, the R-CNN passes them through a convolutional neural network in order to compute features. Once features are extracted and training labels are applied, the R-CNN procedure optimizes one linear SVM per class. Then, for each class, the R-CNN scores each extracted feature vector using the SVM trained for that class. Given all scored regions in an image, R-CNN applies a greedy non-maximum suppression method that rejects regions having an intersection-over

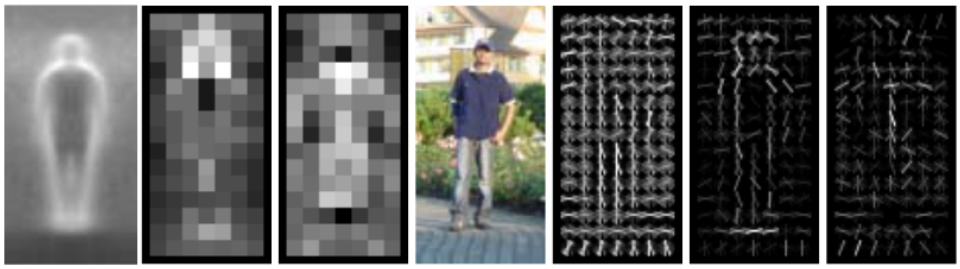

Figure 2.6: HOG classifier. a) The average gradient image over the training examples. (b) Each pixel shows the maximum positive SVM weight in the block centred on the pixel. (c) Likewise for the negative SVM weights. (d) A test image. (e) The test image R-HOG descriptor. (f,g) The R-HOG descriptor weighted by respectively the positive and the negative SVM weights. Reprinted from Dalal and Triggs (2005).

union (IoU) overlap with a higher-score selected region larger than a learned threshold.

R-CNN starts with the ImageNet pre-trained CNN. A SVM training set is created by running the region proposal model over the training set and putting examples with high ground truth IoU into the positive set and other proposals into the negative set. A linear SVM is trained on these positive plus "hard negative" examples. However, fine tuning such a model on new data turned out to be difficult. When the researchers used the same method of selecting fine tuning examples as they did for the original SVMs, the performance was worse than result with the typical method for fine tuning, in which we update the final softmax layer of the fine-tuned CNN. The softmax classifier was trained on randomly-sampled negative examples rather than on the subset of "hard negatives" used for SVM training. As result, RCNN uses softmax and random sampling of negatives for fine tuning.

At test time, each region is classified using class-specific linear SVMs. Linear SVMs are used at the output to make it relatively easy to review the classifier model and fine-tune it. The final step is find the tight dimensions of the object bounding box. The R-CNN runs a simple linear regression on the region proposal to generate tighter bounding box coordinates for the final result. This method yields good results, but it requires a large amount of training data and a large amount of compute power. R-CNNs require a forward pass of the CNN for every single region proposal, and we also have to train three different models: the CNN to generate image features, the classifier to predict the class, and the regression model to find the tight dimensions of the candidate object. These factors make R-CNNs slow. An overview of the system is shown in Figure 2.7.

2.3.4 FAST R-CNNs

The Fast Region-based Convolutional Network method (Fast R-CNN) by Girshick (2015) builds on R-CNNs to improve training and testing speed while also increasing detection accuracy. R-CNNs are slow because they perform a CNN forward pass for each object proposal, without sharing computation. Fast R-CNNs try just run the CNN once per image and share computation over different region proposals. Fast R-CNNs take an input image and multiple regions of interest (RoIs) as input to a fully

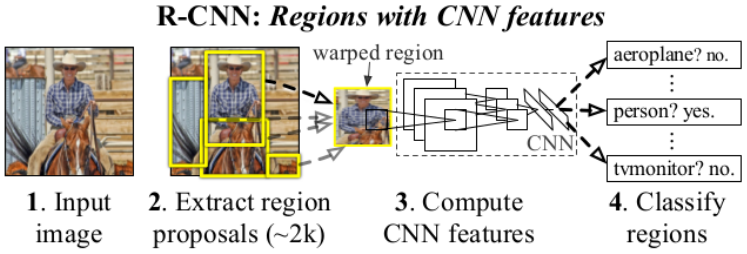

Figure 2.7: Object detection system overview. Reprinted from Girshick et al. (2014).

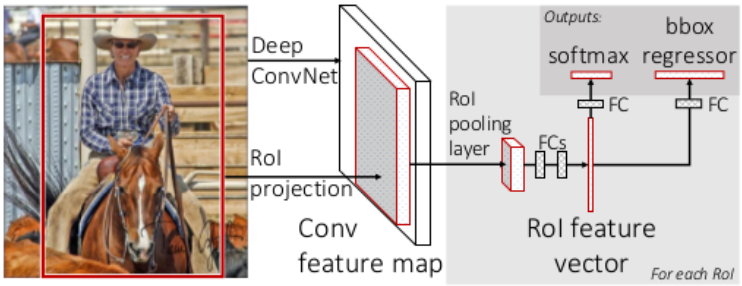

Figure 2.8: Fast R-CNN architecture. Reprinted from Girshick (2015).

convolutional network. Each ROI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers. The second thing that Fast R-CNNs do is join the three models of the R-CNN into one model. Fast R-CNNs replace the SVM classifier with a softmax layer on top of the CNN. It also adds a linear regression layer parallel to the softmax layer to output bounding box coordinates. Thus, the network has two output vectors per RoI: a vector softmax probabilities and a vector of per-class bounding-box regression offsets. The architecture of the fast R-CNN is shown in Figure 2.8.

2.3.5 Faster R-CNN

The faster R-CNN (Ren, He, Girshick, & Sun, 2015) is built from previous work especially, the Fast R-CNN and the basic R-CNN. In previous work, the proposals are created using Selective Search, but this is a slow process that was found to be the bottleneck in the overall process. Faster R-CNN introduces a Region Proposal Network (RPN). A RPN is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. Faster R-CNN reuses the same CNN results for region proposals instead of running a separate selective search algorithm. See Figure 2.9. The Region Proposal Network works by passing a sliding window over the CNN feature map and at each window, outputting K potential bounding boxes and scores for how good each of those boxes is expected to be. The model then passes each bounding box that is likely to be an object to a Fast R-CNN to generate a classification and tightened bounding boxes as previously described.

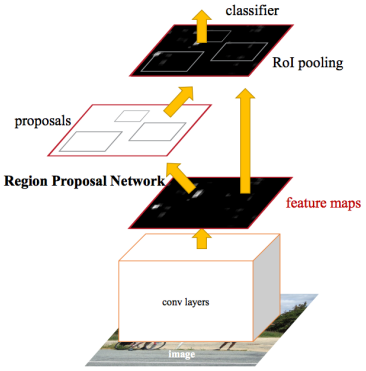

Figure 2.9: Faster R-CNN. Reprinted from Ren et al. (2015).

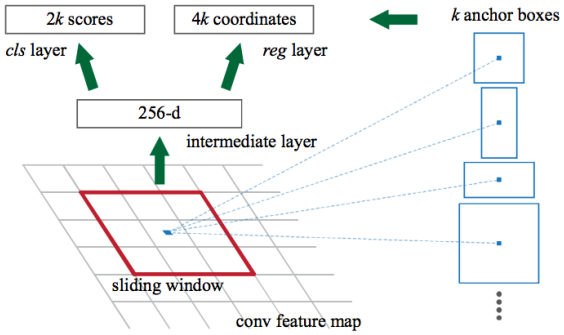

Figure 2.10: Region Proposal Network (RPN). Reprinted from Ren et al. (2015).

2.4 Optical Flow

Optical flow (Horn and Schunck (1981)) is a technique used for estimating the motion of moving objects in a scene by keeping track of features between consecutive frames. "Sparse" optical flow methods start by finding strong corners in the image and then calculating the optical flow for the sparse feature set. The most popular and efficient method is the iterative Lucas-Kanade pyramid method.

Optical flow works on two assumptions:

1. The pixel intensities of an object do not change between consecutive frames.
2. Neighbouring pixels have similar motion.

Consider the motion of a pixel $I(x, y, t)$ (where t is time) that is observed in several consecutive frames. If the object moves by a distance (dx, dy) between two frames taking dt time, under the invariant intensity assumption, we have

$$I(x, y, t) = I(x + dx, y + dy, t + dt). \tag{2.1}$$

We perform a Taylor series approximation

$$I(x+dx, y+dy, t+dt) \approx I(x,y,t) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt.$$

From equation (2.1), since $I(x,y,t) - I(x+dx, y+dy, t+dt) = 0$, dividing by dt, we obtain

$$\frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t}\frac{dt}{dt} = 0,$$

which we rewrite as

$$f_x u + f_y v + f_t = 0,$$

where

$$f_x = \frac{\partial I}{\partial x}\ ;\ f_y = \frac{\partial I}{\partial y};\ f_t = \frac{\partial I}{\partial t}$$

$$u = \frac{dx}{dt}\ ;\ v = \frac{dy}{dt}.$$

The image gradients are f_x and f_y. f_t is a gradient over time. u and v (the motion of the pixel in equation) are unknown. One method to solve this problem is Lucas-Kanade. The Lucas-Kanade method assumes that the spatial displacement of the image content between two successive frames is small. Thus the same optical flow constraints can be assumed to hold for all of the pixels within a window centerd at p (the pixel of interest). Letting $q_1, q_2, \ldots, q_n$ be the pixels within the window, the common local image flow (velocity) vector (u, v) must approximately satisfy

$$f_x(q_1)u + f_y(q_1)v = -f_t(q_1)$$

$$f_x(q_2)u + f_y(q_2)v = -f_t(q_2)$$

$$\vdots$$

$$f_x(q_n)u + f_y(q_n)v = -f_t(q_n).$$

These equations can be written in matrix form $A\nu = b$, where

$$A = \begin{bmatrix} f_x(q_1) & f_y(q_1) \\ f_x(q_2) & f_y(q_2) \\ \vdots & \vdots \\ f_x(q_n) & f_y(q_n) \end{bmatrix} \nu = \begin{bmatrix} u \\ v \end{bmatrix} b = \begin{bmatrix} -f_t(q_1) \\ -f_t(q_2) \\ \vdots \\ -f_t(q_n) \end{bmatrix}$$

The least squares solution for ν can be derived by first transforming the linear system using

$$A^T A\nu = A^T b$$

then obtain,

$$\nu = (A^T A)^{-1} A^T b,$$

where A^T is the transpose of matrix A. In a simplified form, we obtain

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i {f_{x_i}}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i {f_{y_i}}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix}. \tag{2.2}$$

According to the assumption that neighbouring pixels must have similar motion, Lucas-Kanade takes

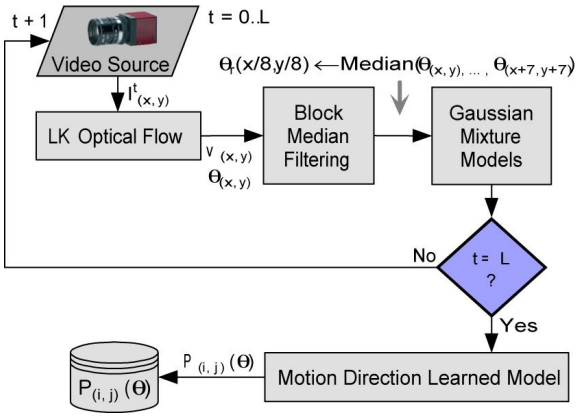

Figure 2.11: Flow chart of the traffic flow direction learning process. Reprinted from Monteiro et al. (2007).

a 3×3 patch around each point. These nine pixels are all assumed to have the same motion, so we calculate $(f_x, f_y, and\ f_t)$ for these nine points. We obtain the least squares solution according to the derivation above.

Large motions will be difficult to identify with this method, as the fundamental assumption of small mention is no longer met. Hence, Lucas-Kanade is applied in a pyramid. At higher levels of the pyramid, small motions are ignored, and large motions of large regions become small motions of small regions that can be identified using the same method as above.

2.5 Traffic Flow Direction Learning

Monteiro et al. (2007) present work on a system whose basic idea is to get the correct direction of motion of vehicles in different lanes is during a learning period when it is assumed that traffic is flowing in the correct direction. Hence, the system learns the correct direction for each point in the scene. The authors propose a learning method that analyzes many frames. A Gaussian mixture model is learned for each block in the image by analysis of vehicles' movement over time. This method is good; it is able to detect vehicles circulating on the wrong side of the road, and it runs in real-time. But this method may have errors in identifying objects and may not work well on more crowded scenes. Learning process has shown in Figure 2.11.

2.6 Object Tracking

Object tracking has been a problem for research for many years. It is still not a solved problem, but there are many object trackers. Object trackers usually need some initialization step, which can be provided manually or automatically using an object detector. Tracking systems must address two basic problem: motion and matching.

- Motion problem: predict the location of an image element using previous positions.
- Matching problem: identify the image element within the designated search region.

2.6.1 CAMSHIFT

CAMSHIFT is a practical application of the meanshift method to tracking of an object using a color histogram. Meanshift iteratively finds the mode of an arbitrary probability density using samples from the density and a kernel. Consider the situation in which we have a set of points. We are given a small window. The basic idea of meanshift is to identify the set of points that fit with in the small window, then move the window to the area containing the highest density of points. But a problem for the standard meanshift method is that when an object moves closer to camera, the object gets bigger, and vice versa when the camera moves away from the object, making the fixed-size window assumption inapplicable. CAMSHIFT tries to solve this problem by adapting the window size and rotation of the target. It updates the size of the window as $s = 2\sqrt{\frac{M_{00}}{256}}$, where M_{00} is the zero-order moment and s is the new window size. CAMSHIFT also calculates the orientation of the best-fitting ellipse for the object. It applies meanshift with the new scaled search window and a previous window location. The method keeps executing this process until convergence is reached, as measured by a threshold on the shift of the window.

2.6.2 Foreground Blob Tracking

Foreground blob tracking analyzes foreground blobs detected by a background subtraction model. Usually, foreground blob detection is not very accurate, requiring smoothing using a filter such as the Kalman filter.

2.6.2.1 Kalman Filter

A Kalman filter is an optimal estimator for the state of a system evolving stochastically over time. It is an algorithm that uses a series of measurements observed over time and is recursive, so that new measurements can be processed as they arrive. The cycle of a Kalman filter is shown in Figure 2.13. Suppose we are tracking a single object. After we detect the object, the detector will give us a candidate location of the object. To predict the next position of the object, we need an object motion model. Furthermore, the detector may not be perfect, so we assume there is noise in the object location, called measurement noise. The motion model is also not perfect, thus we also assume we will have noise in the motion model, called process noise. To estimate the next position of an object, we need three parameters: the object motion model, the measurement noise, and the process noise.

The initial state of the object must be estimated somehow. Usually, we would use the first detection with the inverse of the sensor model as an initial state. Typically, the state includes the object's position and velocity in 2D or 3D, and the detector gives a noisy estimate of object's 2D or 3D

position, usually 2D only. This is the position measurement. Let $x(t)$ be the state vector and, $z(t)$ be the measurement. Given the initial state $x_{0|0}$ and the initial uncertainty about that state expressed as a Gaussian covariance matrix $P_{0|0}$, we begin prediction and correction.

Predict is the step of extrapolating the next state using the motion model. Prediction also updates the uncertainty about the object state. In the state prediction step, an estimate of the next state $\hat{x}_{t|t-1}$ is obtained by multiplying the previous state estimate by the state transition matrix (F). This prediction can be written as

$$\hat{x}_{t|t-1} = F_t \hat{x}_{t-1|t-1}.$$

The covariance prediction is done by multiplying the covariance matrix from the previous iteration by the state transition matrix F and by adding the process noise Q, which can be constant (Q) or dynamic (Q_t). The prediction for P is

$$P_{t|t-1} = F_t P_{t-1|t-1} F_t^T + Q_t.$$

Correct. Here we perform a Kalman filter update, which includes a state update and an uncertainty update. First we obtain a noisy measurement $z(t)$. The noise in $z(t)$ is modeled by a single Gaussian covariance matrix R_t. To update the state estimate using the measurement, we use the measurement sensor model H. We first calculate the prediction error, i.e., the difference between the predicted and observed sensor measurement. This is called the residual or innovation, denoted by $\hat{y}_t$. The innovation or measurement residual is calculated as

$$\hat{y}_t = z_t - H_t \hat{x}_{t|t-1},$$

where $\hat{x}_{t|t-1}$ is the predicted measurement and z_t is the actual measurement. The uncertainty in the innovation residual is

$$S_t = H_t P_{t|t-1} H_t^T + R_t,$$

where $P_{t|t-1}$ is the predicted covariance and R_t is the measurement noise the first update uses the Kalman gain matrix (K), which specifies how much we believe the prediction and how much we believe in the measurement. K_t is calculated as

$$K_t = P_{t|t-1} H_t^T S_t^{-1}.$$

Clearly, K_t is large if $P_{t|t-1}$ is large and is small if S_t is large. The Kalman gain is now used to update the state x and covariance matrix P according to

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t \hat{y}_t$$

$$P_{t|t} = (I - K_t + H_t) P_{t|t-1}.$$

When the correction step is finished, we repeat the prediction step, as shown in Figure 2.12. When K_t is large, we "trust" the innovation more, and if K_t is small, we "trust" the predicted state more.

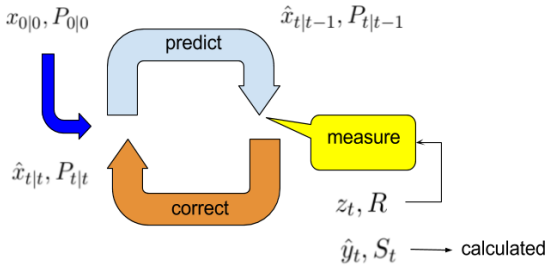

Figure 2.12: Overview of processing by a Kalman filter. An initial estimated of the state after posterior error distribution care recursively updated as new measurements arrive.

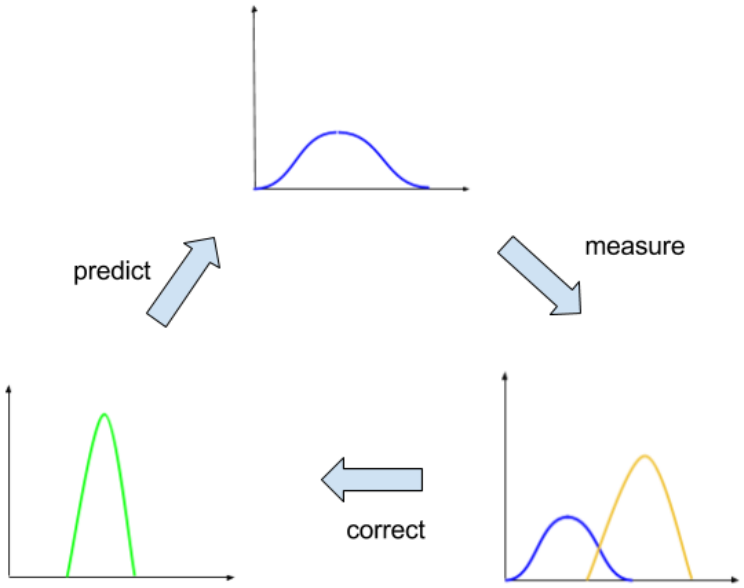

Figure 2.13: The cycle of a Kalman filter. First the filter predicts the next state from the provided state, then if applicable, the noisy measurement information is incorporated in the correction phase. The cycle is repeated.

Chapter 3

Preliminary Prototype

This chapter describes an experiment and shows the preliminary prototype of wrong-way driving detection system.

3.1 System Overview

The proposed system is designed for study possible ways to detect wrong-way driving. This system aims to estimate motion for rigid 3D objects moving parallel to a ground plane. The system consists of 2 parts: motion estimation part and learning part. The workflow of the system is shown in Figure 3.1.

3.2 Motion Estimation (Optical flow)

The detection and motion estimation in this system is based on Optical flow. It looks for corners in image and tags its as a good feature. The good features from two frames (previous frame and current frame) are mapped. Then, we can see a motion of the object. In Figure 3.2 shows motion detection of vehicles on the road.

3.3 Learning (Kalman filter)

We process at image size is 1280×720 pixels

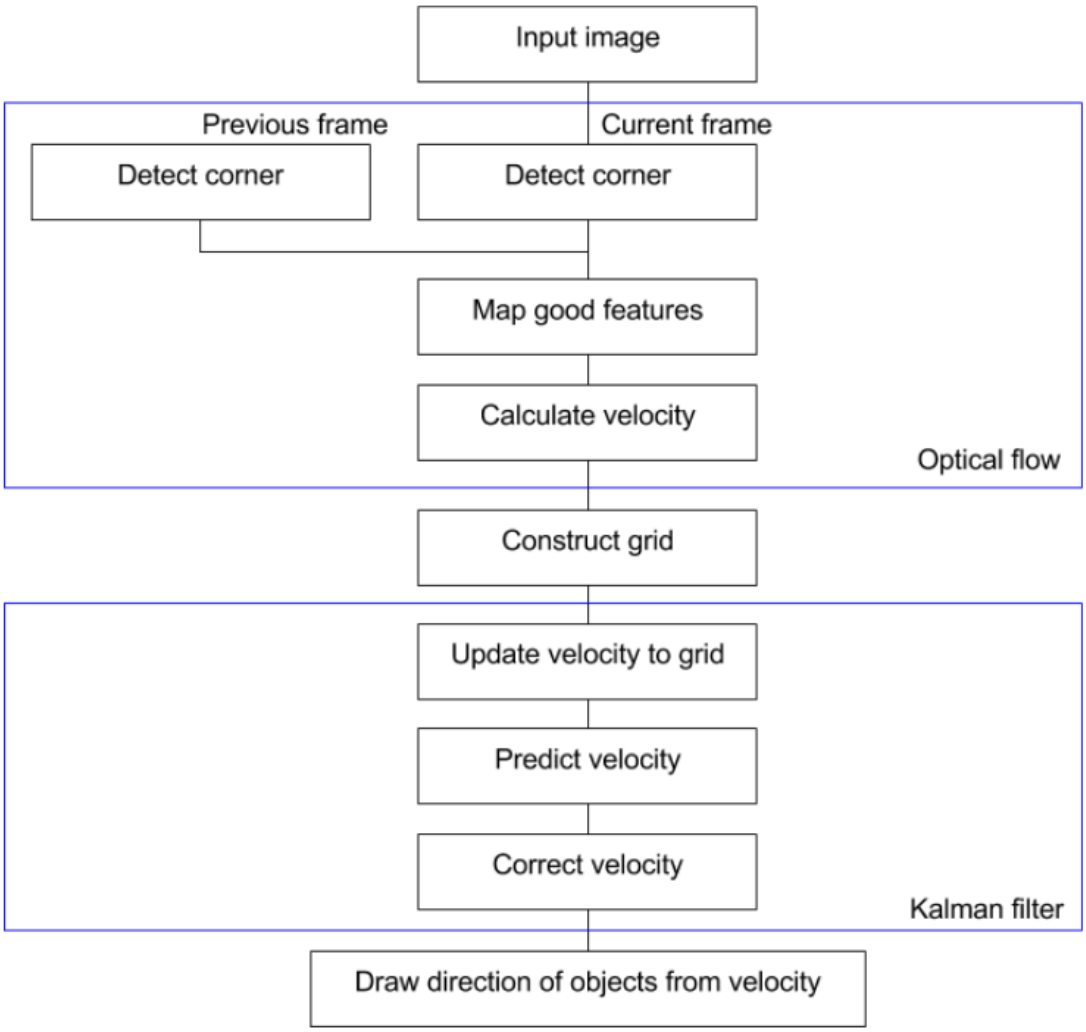

Figure 3.1: System workflow.

Figure 3.2: The result of Optical flow.

Figure 3.3: Direction from learning velocity.

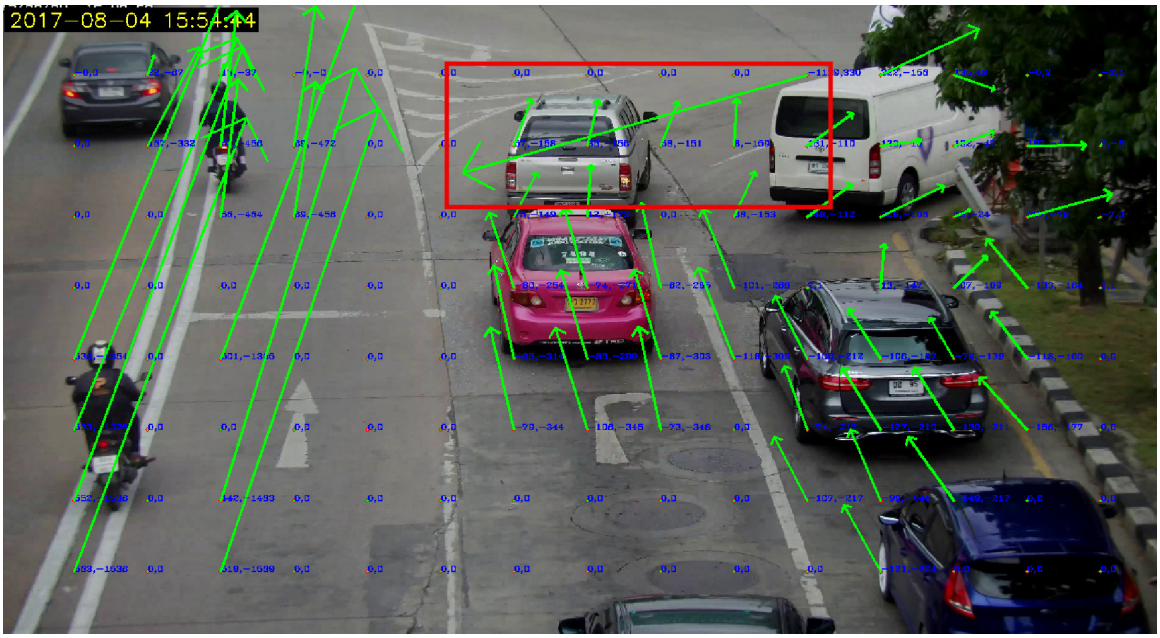

Figure 3.4: The error of the system.

Chapter 4

Conclusion

4.1 Conclusion

In this special study, the system is one of the ways that can get the vehicle direction,but this system still has an error as shown in Figure 3.4. Some directions are wrong and made the learning part confuse. Because optical flow needs two frames to calculate velocity, it can not have velocity in every frames. This is a limit of optical flow. Thus, the system needs to use the predictable values from Kalman filter.

4.2 Future Works

This special study implemented an algorithm for getting vehicle direction. This can improve to be a wrong-way driving detection. The suggestion for improving are filter out the error from optical flow and use a powerful algorithm for learning part.

References

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In *IEEE computer society conference on computer vision and pattern recognition (CVPR)* (pp. 886–893).

Deep Learning Development Team. (2017). *Deep learning documentation.*

Forthoffer, M., Bouzar, S., Lenoir, F., Blosseville, J., & Aubert, D. (1996). Automatic incident detection: Wrong-way vehicle detection using image processing. In *World congress on intelligent transport systems: Realizing the future.*

Girshick, R. (2015). Fast R-CNN. In *Proceedings of the IEEE international conference on computer vision (CVPR)* (pp. 1440–1448).

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 580–587).

Horn, B. K., & Schunck, B. G. (1981). Determining optical flow. *Artificial Intelligence*, *17*(1-3), 185–203.

Monteiro, G., Ribeiro, M., Marcos, J., & Batista, J. (2007). Wrongway drivers detection based on optical flow. In *IEEE international conference on image processing (ICIP)* (pp. V–141).

OpenCV Development Team. (2017). *The OpenCV reference manual.*

Paragios, N., & Deriche, R. (2000). Geodesic active contours and level sets for the detection and tracking of moving objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(3), 266–280.

Piccardi, M. (2004). Background subtraction techniques: A review. In *IEEE international conference on systems, man and cybernetics* (pp. 3099–3104).

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (pp. 91–99).

Viola, P., & Jones, M. J. (2004). Robust real-time face detection. *International Journal of Computer Vision*, *57*(2), 137–154.