CONVENTIONAL AND LEARNING APPROACHES FOR OBJECT RECOGNITION AND TRACKING

YANG XUEFEI

MPhil

The Hong Kong Polytechnic University

2019

The Hong Kong Polytechnic University
Department of Electronic and Information Engineering

Conventional and Learning Approaches for Object Recognition and Tracking

Yang Xuefei

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Philosophy

June 2018

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____

YANG XUEFEI

_____

Abstract

Nowadays driver assistant system is popular in both research and application field. An efficient driver assistant system will alleviate the driver's burden, provide essential warnings, and increase the overall driving safety. While a wide range of sensors such as radar and ultra-sound devices are available for developing driver assistant system, vision-based analysis remains to be a robust and inexpensive approach.

Driver assistant system, especially collision warning system, is also important for Light Rail Vehicles (LRVs). However, there lacks research on vision-based collision warning for LRV in the academic field. We develop this LRV Close-up Monitoring System for the Hong Kong Light Rail, which aims at providing warning signals once the system detects any frontal vehicle approaching certain safety distance. The challenges lie in the vast change of environmental conditions and scales of the front vehicles. As a real-time real-world application, the system is required to make fast and reliable detections in a variety of situations with very limited computation time.

We design a hierarchical multi-module structure to achieve the above objectives. Each module adopts various orthogonal or semi-orthogonal features to detect vehicles under certain circumstances. We also improve the detection accuracy by designing a verification module that checks the low-confident detections by totally different sets of features. These studies specifically show how multiple orthogonal features could increase discriminability as well as maintain high robustness. The system achieves high performance with no missing detections and few false alarms in our field tests.

In our further research, we aim at enhancing individual modules by adopting machine learning techniques. We propose a modified decision tree algorithm to form a shadow detection module. The shadow detection module aims at recognizing the shadow part of the LRVs in an early stage to accelerate the detection process. Our proposed modified decision tree classifier takes each binary node as a weak classifier and combine all weak classifier predictions to obtain the final prediction and confidence measures. Our evaluation shows that the proposed detector significantly reduces the computation time by exploiting simple intensity pair features in binary test design, while giving the best

detection accuracy (the highest F-score) by combining the predictions from all decision nodes. We also study LRV detections using deep learning. We improve the accuracy of the vehicle detection module by adopting the Faster RCNN algorithm. Specifically, we propose a novel Adaptive ROI detection scheme to deal with remote-ranged vehicles. Compared with a direct implementation of Faster RCNN, experimental results show our proposed algorithm has achieved a significant improvement with a 48% (87.7-38.9)% increase of recall rate for “remote-range” detections, while maintaining an excellent performance for close-range detections.

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Professor Wan-Chi Siu and Dr. Yui-Lam Chan, for their constant encouragement and support. They have offered me invaluable guidance during my study. I also would like to thank Dr. Lun for his support and assistance in my study.

Special thanks to Mr. Hui-Wai Lam, Mr. Jun-Jie Huang, Mr. Meng Yao, Mr. Zhi-Song Liu, Mr. Chu-Tak Li, Mr. Wei Kuang, Ms. Zi-jun Wang and other members in Centre for Multimedia Signal Processing. My heartfelt gratitude goes to my classmates that we have made a good progress together.

Last but not least, I am deeply indebted to my parents and friends for their continuous company and love.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

CMS .....Close-up Monitoring System
CNN ..... Convolutional Neural Network
HOG .....Histogram of Orientated Gradient
IoU.....Intersection over Union
LBP.....Local Binary Pattern
LRV ..... Light Rail Vehicle
RCNN..... Region-based Convolutional Neural Network
RPN ..... Region Proposal Network
SIFT.....Scale-Invariant Feature Transform
ROI ..... Region-of-Interest
SSD..... Sum of Squared Distance
SURF .....Speeded Up Robust Features
SVM ..... Support Vector Machine

Chapter 1 Introduction

1.1 Driver Assistant System

Intelligent transportation system has been a popular topic in both academic and application fields. It is reported that every year over tens of thousands of people die of traffic accidents [1]. Thus, it is extremely important developing reliable vehicle detection systems that can alleviate the driver's burden, provide essential warnings, and increase the overall driving safety. A wide range of sensor techniques has been applied to the driverless system. Radar is one of the commonly used sensors [2, 3]. Radio waves are transmitted and collected to analyze the range, angle, and velocity of any objects in the front. While Radar signals are robust to illumination and weather conditions, the detection field-of-view is relatively narrow, and the radio signals can be quite noisy, requiring additional filtering techniques. Lidar tends to replace Radar and becomes a leading technology in driver assistant system, due to its purer signals and wider sensing range [4-6]. However, the cost of Lidar sensors remains high. Unlike Radar and Lidar sensing technology that analyzes returned signals reflected by the front object, and classifies vehicles based on speed and size of the object, vision based detection system directly recognize vehicles from raw images. The hardware requirement for vision based system is simple and inexpensive, mostly one or multiple cameras. Although vison based detection could be sensitive of illumination and weather change, it is believed that raw images can provide richer information, and deal with more complex situations due to the advance in computer vision techniques.

1.2 Vision-based Vehicle Detection

Vision based driver assistant system can be categorized into monocular vision based vehicle detection and stereo vision based detection based on the amount and type of cameras. Our research only involves monocular vision based system due to its low cost and practical values in industrial projects. The detection can be achieved by either appearance based approaches that recognize vehicles by modeling the appearance features of target vehicles, or motion based approaches that analysis the motion of the moving target through a group of frames [7]. However, due to the lack of additional disparity and

depth data for monocular camera system, the motion based approaches are not as popular as appearance based approaches.

1.2.1 Vehicle Detection Framework

This hierarchical structure is certainly promising for vehicle detection systems. The structure can be roughly described as a two-stage detection process [8]: 1) the hypothesis generation stage, and 2) the hypothesis verification stage. The hypothesis generation stage searches for the whole frame, trying to locate all candidate regions where a vehicle might exist, and then generate a proper region of interest for further exploration. Fast detection is required at this stage, so many of the hypothesis generation methods exploit simple features that describe local properties of the vehicle.

1.2.2 Appearance Features for Vehicle Detection

The earlier appearance features used for vehicle detection are often simple rear-view features that consider the edge and symmetry properties of vehicles, such as shadows, defined as a rectangular-shaped region underneath the vehicle, darker than its adjoining road surface [9-12], or symmetry pair of vertical edges along the left and right side of the rear view [13-16], or a pair of rear-lamps that is especially discriminative in nighttime conditions [17].

Shadow recognition is an important hypothesis generation process. It functions as a hypothesis generation of any locations that could possibly have a vehicle. The aim is to accelerate the detection by recognizing part of the train first so that a finer detection could be performed only at these locations of interest, instead of sliding windows over the entire frame. In [9] the shaded area is extracted from the paved road by an adaptive threshold, estimated from the average gray level of the free-driving-space (the lowest central homogenous region in the image), and further verified by examining the horizontal edge between the shadow and the paved road. [10] Takes very similar approach, but improves the threshold generation part by modeling the free-driving-space as normal distribution, and thus the approach can determine the threshold more intelligently based on mean and standard deviation of road pixels. The successful extraction and modeling of the free-driving-space (road region) plays the key role in both methods. Thus although both are effective shadow detectors for on-road vehicles under ideal illumination, situations like

complex-structured road, unevenly illuminated road, and also nighttime road, could affect greatly the performance of the detectors. Another commonly used appearance feature, especially in nighttime videos, is the rear lamps/taillights [11, 17]. Taillights are strong features often exist in pairs and share symmetry properties. Vehicle color [18] has also been exploited.

The recent advances in more general and robust feature extraction has boosted the vehicle detection to exploit those general features with machine learning techniques. One of the commonly used features is the Histogram of Oriented Gradient (HOG) feature. Proposed as a general feature describing the human contour [19, 20], the HOG feature sees a great success in pedestrian detection. HOG features are highly discriminative for objects with special edge patterns, and are also robust to illumination changes, thus HOG feature is also applied to many other detection and recognition tasks such as vehicle detection [10]. Haar wavelet features [21] calculate the contrast of neighboring rectangular regions with the help of integral images. The most advantage of Haar-like features over the HOG features are the fast-computational speed. Thus, a number of vehicle detection systems adopted this feature for fast detection [22-28]. Other features like SIFT [29] are not commonly seen in intelligent transportation system due to its extensive requirement of computation [30].

1.2.3 Classification of Appearance Features

The classification methods for vehicle features develop with the classification for general objects. SVM [31] is a good classification tool compared with Neural Network, which often has too many parameters to tune and are vulnerable to local optimum. [32] extracts Haar-like features and uses SVM for classification, while [23] and [33] use HOG features plus the SVM classifier. Another widely used classification tool in vehicle system is the AdaBoost classifier [21]. [34-39] are vehicle detection systems that adopted a combination of AdaBoost and Haar-like feature. Decision tree was firstly proposed by Breiman et al. [35], and is now a commonly used data mining algorithm. The idea of decision tree is to solve a complex problem by testing some simple questions which are organized hierarchically, partitioning the problem to a more specific region of the decision space

according to the answers to the questions and making a decision when the problem reaches a region where the response is confident enough.

Decision tree is a commonly used classification tool and has been successfully applied to many computer vision tasks [40].

Deep learning has been receiving constant attention in the past few years and has achieved the state-of-the-art performances in various fields such as image classification [41], segmentation [42], and object detection [43]. RCNN [44] applied convolutional neural network to classify region proposals generated by auxiliary objectness detection methods [45], and has achieved significant increase in detection accuracy on VOC dataset. Faster RCNN [46] further improves the detection speed by integrating region proposal network (RPN) and classification network (Fast RCNN) into one unified, end-to-end trainable net. However, Faster RCNN does not perform well in detecting small objects [47]. This is because for small objects, the feature map is too coarse after several pooling layers, and hence not enough information would remain during the ROI pooling process in the Fast RCNN network, and the classification afterwards will thus be degraded.

1.3 Rail and Light-rail Vehicle Detection

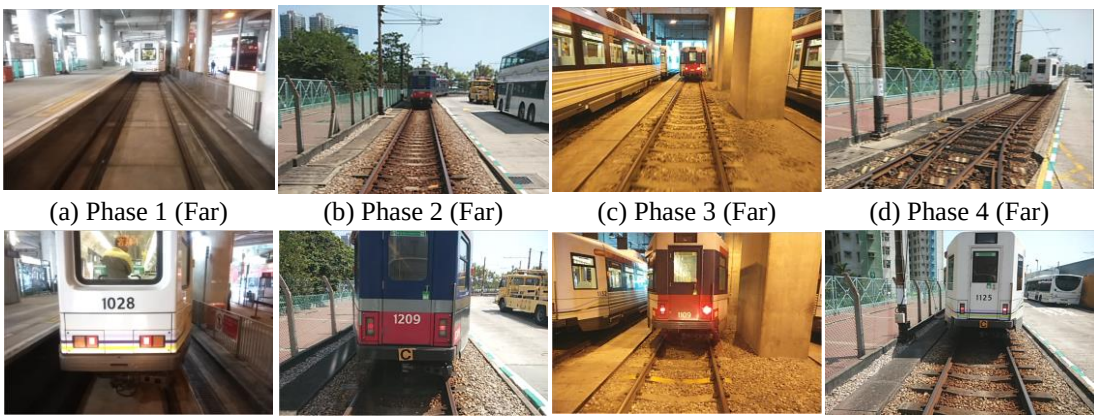
Light Rail Vehicle (LRV) is a popular public transport in urban cities due to inexpensive, highly convenient, and green characteristics. Light rail transportation helps to mitigate traffic congestions, especially at commute peak. Therefore, it is extremely important to ensure the safety for LRV systems. Like trains, there are usually more than one vehicles running on the same rail at the same time, but the interval between two consecutive trains is very short. Because of the huge inertia, the LRVs cannot stop in a short distance, which makes it important to control the interval to avoid collision risk. Some of the traditional safety guarantee systems are based on trackside infrastructure elements, like axle counters, track circuits, which are expensive to install and maintain. Compared with the traditional approaches, the hardware requirement for vision based system is simple and inexpensive, mostly single or multiple cameras, but the information it can capture is rich. Therefore, it is beneficial and important to develop vision-based vehicle detection techniques for driver assistant systems in light rail.

1.4 Introduction of Light Rail Vehicles Close-up Monitoring System

Our project focuses on developing a driver assistant system, the first level of vehicle autonomy, for light-rail vehicles (LRV) operated in Hong Kong. The reason is to reduce the collision risks caused by driver drowsiness or distraction. The objective of our project is to provide accurate collision warning signals to the LRV driver whenever the distance of the front vehicle exceeds certain threshold. Our project uses monocular camera only to achieve vehicle detection, tracking, distance estimation and software-level decision making. The advantage is the low cost and high flexibility. The system can be easily implemented by a single smartphone, and therefore is capable of extensive field tests and trial runs, and even commercial-scale production. Although the project only aims at achieving the initial level of autonomous driving, once proved effective and reliable, the consequent design of autonomous control can then be done.

We are developing a vision-based driving assistant system for light rail vehicles. The aim is to achieve highly reliable recognition and tracking of any possible vehicle in the front through the camera mounted on the moving train. The distance can be calculated based on detection results and warnings of collision will be given when the distance becomes close.

The challenges lie in the vast change of environmental conditions and scales of the front vehicle. As a real-time real-world application that runs 24 hours a day, the system is required to make fast and reliable detection in a variety of situations, ranging from daytime to nighttime, from well-illuminated outdoors to gloomy indoors, from bright sunny weathers to cloudy and or rainy ones, and more difficultly, in some extreme situations where the backlight and headlight are too strong for the vehicle to be recognizable. As a collision warning application, the system is expected to be extremely sensitive for close-range detection as well as performing steady detection and tracking from a far distance. Figure 1 and Figure 2 show some examples of environmental variation and some challenging situations.



(a) Phase 1 (Far) (b) Phase 2 (Far) (c) Phase 3 (Far) (d) Phase 4 (Far)

(e) Phase 1 (Near) (f) Phase 2 (Near) (g) Phase 3 (Near) (h) Phase 4 (Near)

Figure 1 Examples of scaling variations of trains

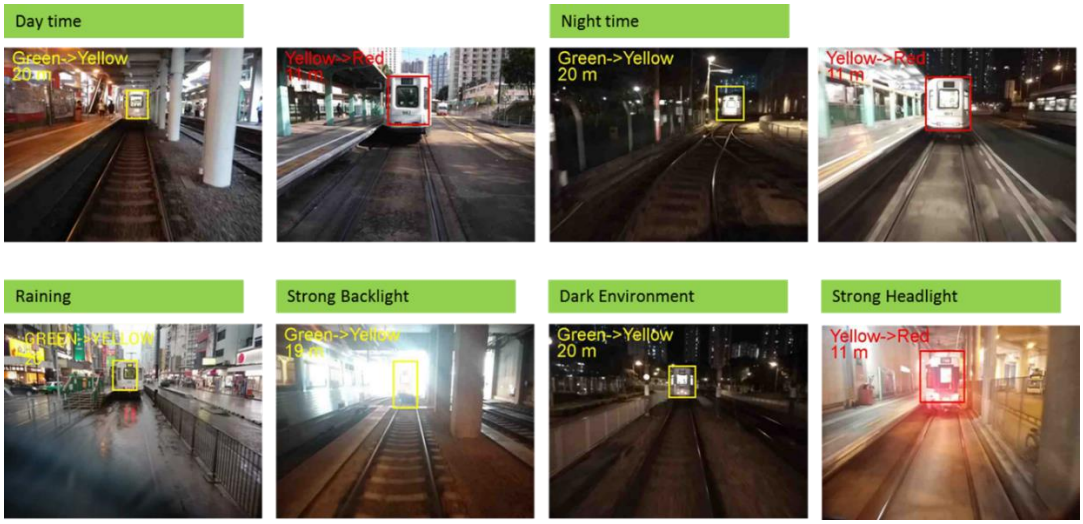


Figure 2 Examples of illumination variation of trains

1.5 Organization of Thesis

This thesis is organized as follows. Chapter 2 briefly reviews the common features and classification tools used in vehicle detection systems. Chapter 3 introduces the complete framework and basic processing modules of our LRV Collision Warning System. Chapter 4 proposes two examples of improving individual modules, hence to enhance system performances. Chapter 5 proposes further improvements through machine learning approaches. Chapter 6 gives the conclusion of this study.

Chapter 2 **Technical Review**

2.1 Appearance Features for Vehicle Detection

2.1.1 Histogram of Oriented Gradients

Histogram of Oriented Gradients (HOG) has become more and more popular since the success of pedestrian detection in 2005. The HOG feature is insensitive to illumination conditions because it is based on the edge information. HOG feature is efficient and discriminative. Therefore, it has been adopted in various object detection algorithms. As shown in Figure 3, Gamma correct process is conducted to normalize the image. Second, the gradient information is obtained by using Sobel 1-D [-1,0,1] operator at horizontal and vertical direction separately. Then, as shown in the following equations, the gradient magnitude and orientation can be calculated in (1), where g denotes the gradient magnitude, θ the gradient orientation, and g_x and g_y denote the gradient along x and y axis respectively.

$$\begin{aligned} g &= \sqrt{g_x^{\,2} + g_y^{\,2}} \\ \theta &= \arctan\frac{g_y}{g_x} \end{aligned} \tag{1}$$

Next, the orientation is divided into 9 bins and the hypothesis patch is partition into several cells. For each pixel position, voting the magnitude to it related bin forms a 9-bin gradient histogram. A sliding window covers some neighbored cells forms a small area called block, and histograms of these cells are concatenated together to form a large one. Then, a L-2 normalization process is conducted as shown in (2) to limit the value from 0 to 1, where bin_i denotes the gradient sum of the i^{th} bin. Finally, collect all the histograms of the blocks to form the final histogram that is the HOG.

$$bin_{i,normalized} = \frac{bin_i}{\sum_{block} bin} \tag{2}$$

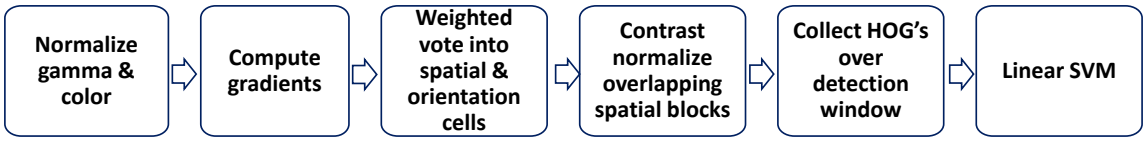


Figure 3 Flowchart of HOG descriptor

2.1.2 Haar-like Features

Haar-like features are commonly used for object detection and face recognition. The concept using intensity differences of adjacent rectangular regions to represent the pattern of a patch. Figure 4 represents some example calculations of Haar-like feature values. The sum of the pixels lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles. The feature response in Figure 4 (A) and Figure 4 (B) are the difference of sum of pixels between two horizontally and vertically adjacent rectangles, while the feature response in Figure 4 (C) is calculated by the sum within two outside rectangles subtracted from the sum in a center rectangle, and feature response in Figure 4 (D) is the difference between diagonal pairs of rectangles. The reason of developing these kinds of features for face detection comes from the observation that the eye regions are normally darker than the nose region on a typical face. Similarly, the reason of applying Haar-like features for vehicle detection comes from the observation that some vehicle parts are normally brighter than other vehicle parts such as shadows.

The Haar features are extremely fast to compute with the help of integral image, which is a pre-computed image map that stores the summation of all pixel intensities for each pixel location. For an image of size 24*24, the exhaustive set of rectangle features at 11 scales will have a number of 37525. However, not all features are useful to discriminate a face, and thus the AdaBoost learning is used as a feature selection process.

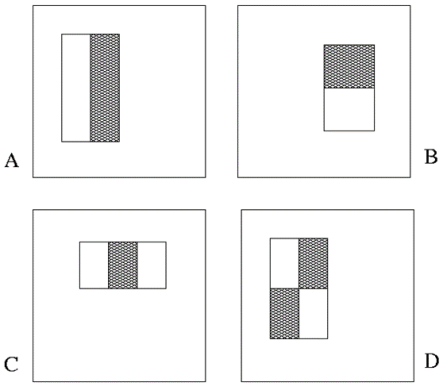


Figure 4 Examples of Haar-like Features (The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles.)

2.2 Classification of Appearance Features

2.2.1 SVM

Support Vector Machine (SVM) is a widely used classifier formally defined by a separating hyperplane. It tries to find a split function which can maximum the margin among classes. As shown in the following figure, the split function (green line) tries to gap the two class (orange and blue) which gives the largest the distance between the two support vectors. However, sometime the classes cannot be gapped by a line split function. By using different kennels, the data space can be mapped into a separating hyperplane, then the SVM can find the split function with the maximum margin.

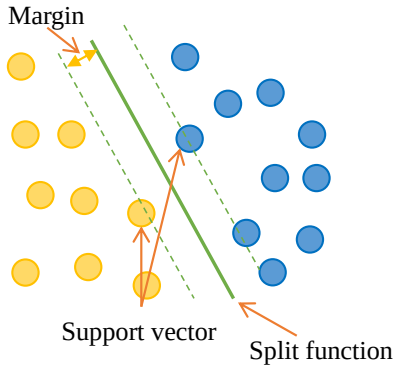


Figure 5, Support vector machine

2.2.2 AdaBoost

An AdaBoost classifier is formed for each stage by combining many weak classifiers (decision stumps in this case) learned in a boosted manner. The features learned in earlier

boosting round are the most representative features describing a face, such as the contrast between eyes and nose regions, while the features selected in later boosting round will focus on discriminating hard examples.

The study of the Boosting algorithms involves two questions: 1) How the weights of the training examples are updated at the end of each boosting round, and 2) How the predictions of each weak classifier are combined. AdaBoost, adaptive boosting, is one learning algorithm that tries to answer the two questions.

Let $\{(x_j, y_j) \mid j=1,2,...,N\}$ denote the original set of N training examples. Multiple base classifiers will be learned sequentially from datasets resampled from this original set. The AdaBoost algorithm assigns an importance score for each base classifier C_i defined by:

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right) \tag{3}$$

Where ε_i denotes the error rate of classifier C_i. When the error rate is large ($\varepsilon_i \to 1$, for example), the importance score has a large negative value, while when the error rate is small ($\varepsilon_i \to 0$), the importance score will have a large positive value. It can be viewed as an evaluation of the performance for each individual weak classifier, and the classifier that provides more accurate predictions has a higher important score.

The error rate ε_i measures the prediction error of base classifier C_i over all weighted training examples. The definition is as follows:

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right) \tag{4}$$

where $\delta(p) = 1$ if p is true, and $\delta(p) = 0$ otherwise, and w_j denotes the weight for the j[th] example.

The importance score α_i is also used to update the weight assigned to example (x_j, y_j) during the t th boosting round. The weight updating scheme is defined as:

$$w_j^{(t+1)} = \frac{w_i^{(t)}}{Z_t}\begin{cases}\exp^{-\alpha_t} & \text{if } C_t(x_j) = y_j \\ \exp^{\alpha_t} & \text{if } C_t(x_j) \neq y_j\end{cases} \tag{5}$$

where Z_t is the normalization factor used to ensure the sum of all example weights is 1. The weight updating scheme will increase the weights of incorrectly classified examples and decrease the weights of the correct ones.

Finally, as shown in (6), the prediction is made by combining the results from all base classifiers, where C^* stands for the final prediction result, and α_j denotes the importance score of the j^{th} class. AdaBoost does not adopt the majority voting scheme. Instead, each base classifier is weighted by its importance score α_i (3). Thus, the prediction could penalize models with poorer accuracy.

$$C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta\left(C_j(x) = y\right) \tag{6}$$

2.2.3 Neural Network

Neural Network algorithm was inspired by animal brains in 1943. As shown in the figure, the whole network consists many neurons, which form the input, hidden and output layers. First, as shown in (7) and Figure 6, the set of data is sent into the input layer. The output y of each neuron is a linear combination of the input data x_i. w_i is the learnable weight, θ is the small bias of the activate function $f(\cdot)$. Then, the hidden layer consists of several neurons that will further process the data. Next, the output layer collects and processes the result of the hidden layer, and outputs the result. Backpropagation algorithm is used to transfer the estimation error to the intermediate neurons, which means the network parameters can be updated after comparing with the target.

$$y = f\left(\sum_i w_i x_i - \theta\right) \tag{7}$$

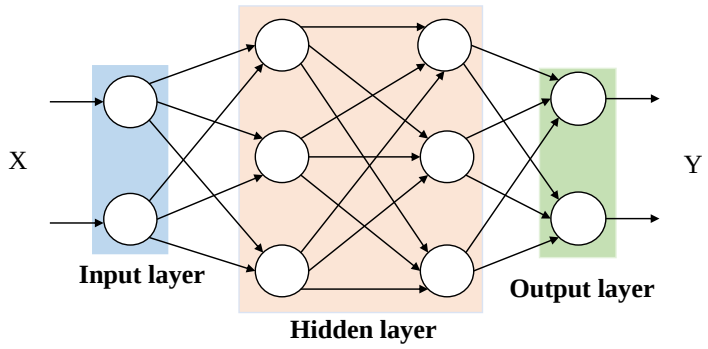


Figure 6. A simple Neural Network structure

2.2.4 Decision Tree and Random Forest

Random forests is an ensemble learning method which enable fast and accurate classification. It is a commonly used classification tool and has been successfully applied to many computer vision tasks. There are multiple decision trees in random forests. As an ensemble method, the random forests approach reduces prediction variance by combing the prediction results from multiple predictors (i.e. decision trees). The averaged prediction from un-correlated decision trees could achieve low bias and low variance even if the prediction result from a single decision tree usually has low bias and high variance.

Inserting randomness helps to reduce the correlation among base decision tree classifiers. The "randomness" comes from a randomly selected subset of the whole training data to train each decision tree, as well as the process that decision trees randomly generate binary test parameters. For each decision tree, a randomly selected F features are used to split each node of that tree. Instead of examining all possible features to find the optimal split, the decision to split a node is determined from these F selected features.

Decision tree classifier solves a classification problem by asking a series of simple questions about the feature space. Each time an answer is received, a follow-up question is then asked until the conclusion about the class label has been achieved. The series of questions and answers are organized in a tree structure, consisting of nodes and directed edges.

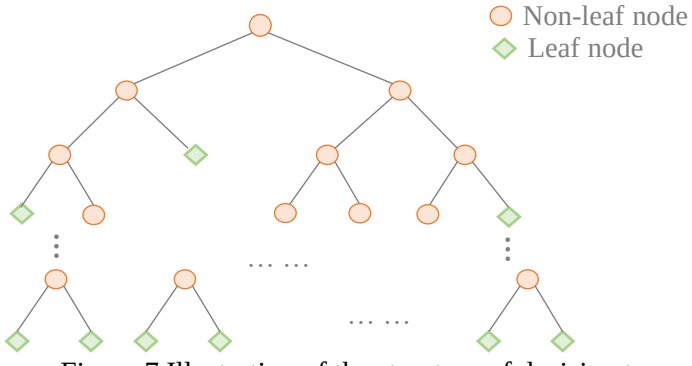


Figure 7 Illustration of the structure of decision tree

2.2.4.1 Leaf and Non-Leaf Node

As illustrated in Figure 7, a binary decision tree consists of non-leaf nodes and leaf nodes. The non-leaf node is a node that has one parent node and two child nodes. Each non-leaf

node stores a binary test that performs the classification and partitions the input data into its left and right child nodes. A leaf node is the end of the tree structure and cannot be further split into child nodes. Each leaf node stores a prediction model determined by the data arriving at that node.

2.2.4.2 Binary Test

Binary test is the test to separate the incoming data into two sections and forward them to two child nodes accordingly. Each non-leaf node stores a binary test that performs the classification and partitions the input data into its left and right child nodes. The test could be either linear or non-linear, and could involve more than one dimensions of attributes. Essentially, the learning process is to find the optimal binary test for each non-leaf node through a pool of candidate binary tests.

Two major concerns can be raised for the decision tree learning algorithm. The first one is how the split function be selected for each non-leaf node. Proper evaluation tool should be applied to compare the goodness of each possible split. Another question is when a node should not be further split. A stopping condition is needed to terminate the tree growing.

Impurity measurement is used for evaluating each candidate split. Let $p(i|t)$ denote the fraction of records belonging to class i at a given node t.The impurity of this node can be evaluated by Shannon entropy $E(t)$:

$$E(t) = -\sum_{i=0}^{C-1} p(i|t)\log(p(i|t)), \tag{8}$$

where C is the total number of classes.

Note that when there is only one class for all data in the node, the entropy is zero, indicating the node is the purest for such data distribution, while a node is the most impure when each class occupy the same percentage.

An information gain $\Delta_{\inf o}$ can be defined as the subtraction of entropies of child nodes from the parent node entropy:

$$\Delta \inf o = E(S) - \sum_{i \in \{left, right)} \frac{|S^i|}{|S|} E(S^i) \quad (9)$$

where *E(S)* represents the entropy of parent node *S*, and $E(S^i)$ represents the entropy of child node *i*, where *i* belongs to either left or right child.

A large information gain indicates the data becomes purer after the split, and thus the larger the information gain is, the better the performance of a split. A split that could maximize the information gain of node *S* is selected as the best split from a pool of candidate splits.

2.2.4.3 Prediction Model

The prediction model stored in each leaf node is estimated by posterior probability. according to Bayes' Theorem. The class distribution of sample data fallen into this leaf reflects the class it predicts. The class that has the highest posterior probability is the final predicted class, and that probability is the confidence score of this prediction.

2.2.4.4 Training

During the training stage, an optimal binary test will be learned by evaluating a pool of candidate tests for each non-leaf node. A prediction model indicating whether the node is positive or negative is also assigned to each leaf node.

A decision tree can grow in a greedy strategy by making a series of locally optimum decisions about the features of the training data. A brief tree growing process considering each node could be stated like this:

1. For each node, decide whether it needs further split or not.
2. If no further split needed, the growing of this node will be stopped, and the node itself becomes a leaf node.
3. If further split needed, select a proper split function for the node.
4. The split function can involve one or multiple feature variables. The function can be linear, non-linear, or even more complex, according to the user design. In fact, the split function can be regarded as a weak classifier, and the entire decision tree is a boosting of many weak classifiers.

5. Then split the data set into two subsets according to the split function

2.2.4.5 Testing

After training, each non-leaf node stores its best binary test binary test parameter and each leaf node stores the learned prediction models.

During the testing stage, the input data will be continuously classified into left or right child nodes according to each binary test result, until reaching to a leaf node. The prediction model stored in that leaf node will map the input data into either positive or negative class.

2.2.5 Deep Learning

2.2.5.1 Region proposal Convolutional Neural Network (RCNN)

RCNN, Region-based Convolutional Neural Network, is one of the first several algorithms that applied deep learning framework on object detection. RCNN first takes out regions that probably contain object information and warps them into image regions of fixed size. It then forwards each extracted image region through a ConvNet, which can be regarded as a feature extraction network. The extracted features, i.e. the output after several conv layers, will be fed into an SVM, which classifies the features into different classes. Finally, a linear regression model (bounding box regression) will be applied to each class to slightly tune the offsets of the bounding box of the region proposal.

Although the RCNN has achieved quite high detection rate on most standard object detection database, there is one drawback. Every image region is forwarded into the ConvNet, and since the number of extracted image regions is not small (around 2k regions for each image), the computational cost is too high. Also, since the image regions are overlapped with each other, many computations in the convolutional stage are in fact redundant. Thus, an improved framework, Fast RCNN is proposed.

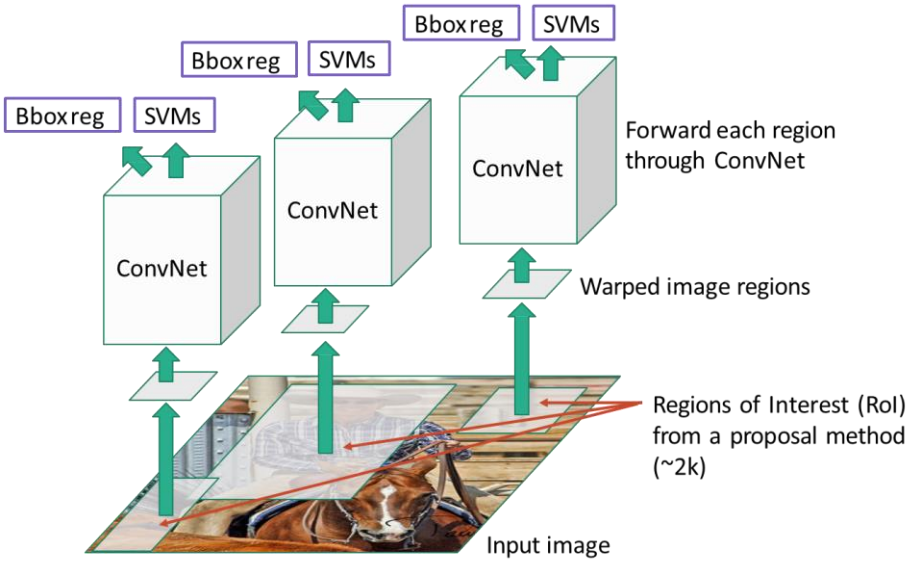


Figure 8 RCNN framework (figure credit to Ross Girschick)

2.2.5.2 Fast RCNN

The essence of Fast RCNN is to share computations through the ConvNet for all region proposals, or in other word, feed the whole image, instead of extracted region proposals, into the ConvNet. The idea of "region proposal" is presented at the end of all convolutional layers (the 5th conv layer in this algorithm). Each region proposal in the original image can be mapped into a small region on the conv5 feature map (the 5th conv layer). ROI pooling is applied on this mapped region, so that each ROI after pooling can have the same feature length.

Another improvement of Fast RCNN is the combination of the final classification and the bounding box regression. At test time (Figure 9), the ConvNet features are sent to two output layers, softmax classifier and bounding box regressor, the result combined is the final object detection result. It is similar to RCNN at test time. The training stage is different. For RCNN, the bounding box regressor is trained for each class. This means that the two procedures are separated, and the bounding box regression is simply an ad-hoc process to adjust the already extracted bounding box. However, in training time (Figure 3), each training region is labeled with a class label, and four bounding box coordinates, and the training is a multi-task training that optimizes both object class and coordinates.

This multi-task training is also inherited by Faster RCNN algorithm, which will be introduced later. Thus, a more detailed explanation on multi-task training will be presented then. It has been proved that without selective search (the region proposal generation procedure), Fast RCNN can achieve over 100 times faster speed compared with RCNN. It also can be seen that in the Fast RCNN framework, the most time-consuming process is not the deep learning process, but the region proposal generation process. It is worth wondering whether the region proposal generation can be skipped. Thus, comes the Faster RCNN framework.

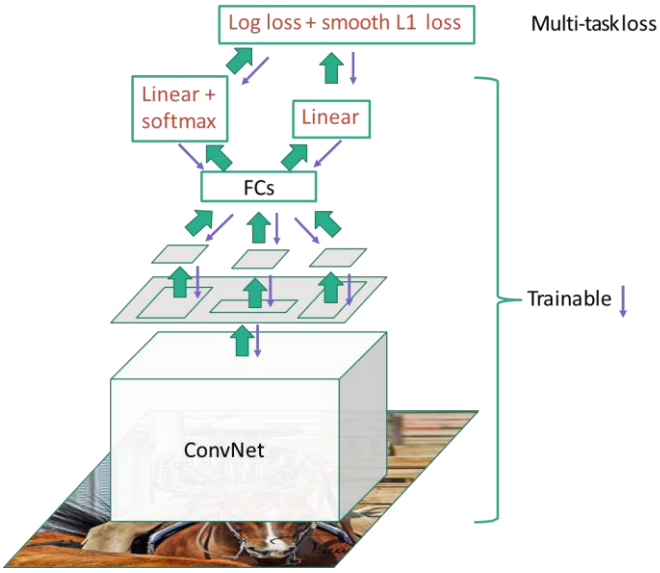


Figure 9: Fast RCNN framework at training time (figure credit to Ross Girschick)

2.2.5.3 Faster RCNN

Faster RCNN is an end-to-end object detection network. The input of the network is a whole image, and the output are the detected objects with their bounding box coordinates. Unlike RCNN, Faster RCNN does not need pre-hoc or post-hoc processes.

As illustrated in Figure 10, The Faster RCNN consists of two important networks: the Region Proposal Network (RPN), and the Fast RCNN network. RPN is responsible for generating region proposals, while Fast RCNN is responsible for the classification based on the region proposals extracted from RPN. Both networks share a same set of fully convolutional layers.

The input of the RPN is the entire image, and the output is a set of object proposals with objectness scores. The region proposals are generated by sliding a small window over the feature map output from the convolutional layers. At each slide location, *k* anchor boxes with same center point, but different size and aspect ratio are generated. Each anchor box is associated with one objectness score that estimates the probability of object and non-object, and four coordinates that encode the bounding box. Thus, the output of RPN to Fast RCNN network for each sliding window would be *2k* objectness scores and *4k* bounding box coordinates.

The RPN is trained by optimizing both classification loss (the probability to be object or non-object) and the bounding box loss, so a multi-task loss function is applied as in (10), where i is the index of an anchor, p_i is the predicted objectness probability for anchor i, p_i* is the ground truth objectness label of anchor i. p_i* is zero if the anchor is labeled as negative (non-object), and one if the label is positive (object). An anchor is labeled as positive if its IoU with any ground truth box is larger than 0.7, or if it has the highest IoU score among other anchors. It is labeled as negative if none of the IoU overlap score is larger than 0.3 for all ground truth boxes.

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i (p_i, p_i^*) + \delta \sum_i p_i^* L_{reg}(t_i, t_i^*) \quad (10)$$

The first term in the summation is the classification loss, which is a log loss over the predicted anchor label with the ground truth anchor label. The second term is the bounding box loss, which is a L_1 loss of the predicted bounding box with the ground truth bounding box.

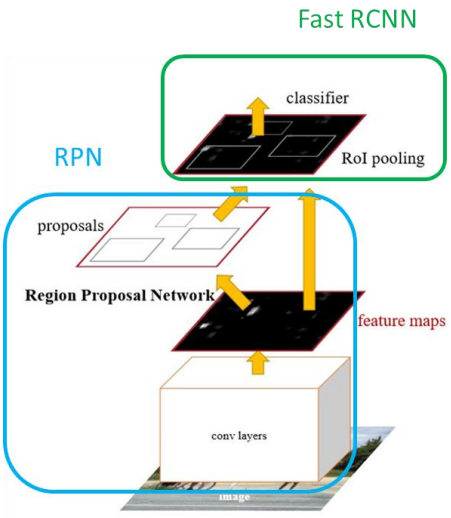


Figure 10 Faster RCNN framework

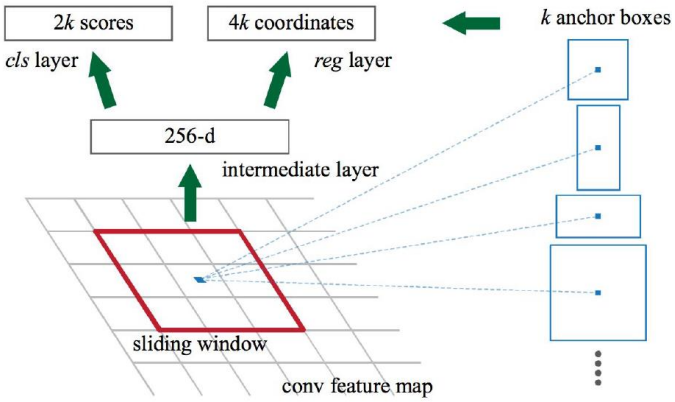


Figure 11 Generating object proposals in RPN of a Faster RCNN framework

Chapter 3 System Structure

3.1 Framework of the Close-up Monitoring System

3.1.1 The Overall Framework

The objective of the light rail vehicle collision warning system is to provide warning signals once the system detects any frontal vehicle inside certain safety distance. A brief description of the system framework is shown in Figure 12. The system first performs Railway Detection to extract a pair of railways in the front to provide guidance for vehicle detection. Then, according to the detected railways, an LRV Detection Module with various sub-modules using multiple object detection algorithms is performed to detect if any light rail vehicle exists inside the region of interest. Once the vehicle is detected, a Distance Estimation module will be activated to estimate the real distance between the camera and the front vehicle, according to the geometric relationship of the camera calibration and the location of the detected vehicle.

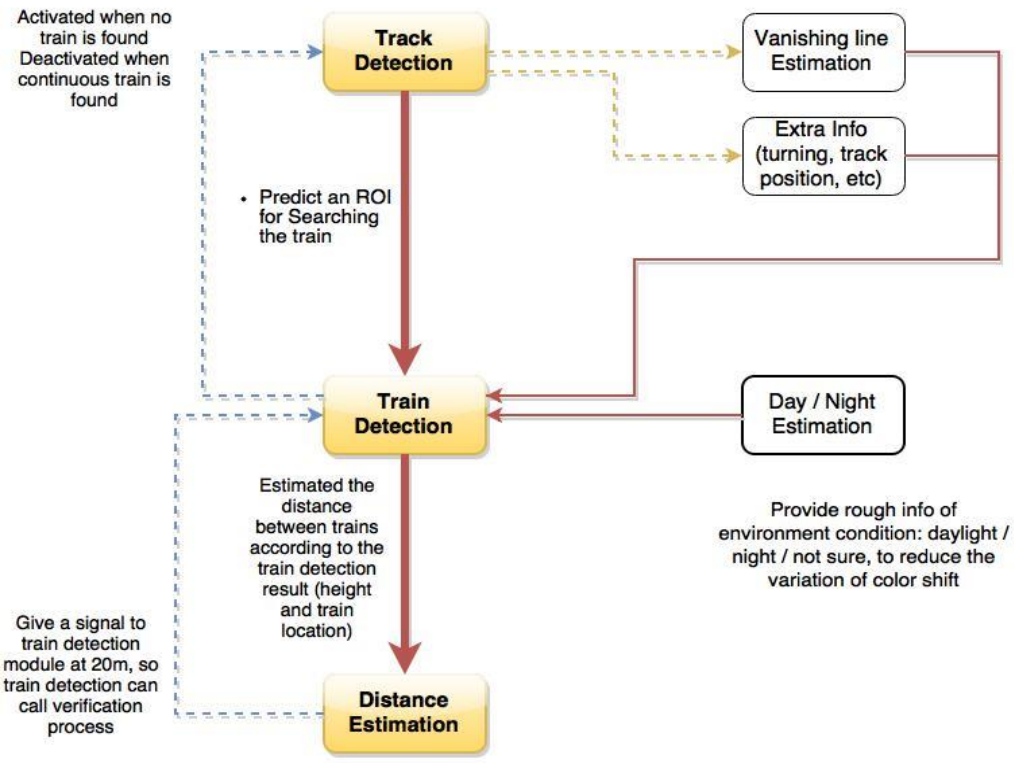


Figure 12 Relationship of Train Detection Module and other modules

LRV Detection Module is the core module in the whole project not only as it provides the essential information of the train situation, but also as it connects with other function modules and plays an important role in combining different modules to make fast and reliable decisions. The detailed relationship between LRV Detection Module and other modules is shown in Figure 12.

The Train Detection Module takes the results of the Railway Detection Module to roughly estimate a reasonable searching region for train detection. This would largely reduce the computation effort of searching the whole frame, and thus increase the detection speed. The Track Detection Module can also provide extra information that helps securing the train detection results such as the vanishing line position, which is an estimated position where the left and right track intersects. The LRV Detection Module also feeds back the Railway Module with the detection result, and the Railway Module would deactivate itself if it receives continuous successful train detection, so as to save computation power.

After conducting LRV detection, the LRV Detection Module would pass the results to Distance Estimation Module where the distance of the front train is calculated based on the received bounding box size. The Distance Estimation Module also feeds back with the records of the distance to the Train Detection Module, and the LRV Detection Module would decide whether to call the verification process based on the distance. Normally the verification would be conducted when the distance is decreasing to 20 meters, which is also the criterion of releasing critical warnings to the driver.

3.1.2 Railway Detection

Railway Detection is the first processing module in the LRV collision warning system, and the major objective is to provide a reliable detection ROI. Thus, the most essential criteria for railway detection is its robustness to various environmental conditions as well as low computational complexity. In order to meet such requirement, a novel railway extraction algorithm using angle alignment measure techniques has been proposed by our group. It is a layer wised railway detection algorithm, where a railway pair is extrapolated using angle alignment measure for each layer from the bottom to up until terminated.

The railway detection algorithm can be divided into two sections: the bottom layer railway detection section, and the iterative upper-layer railway extrapolation. The bottom layer railway detection divides the bottom layer into blocks, and uses HOG to identify the initial rail-area blocks that have obvious dominant gradients. Then it performs Hough transform to search for straight lines. Considering the color distribution for upper and lower regions of the railway line should be similar, the straight lines with inconsistent color histograms would then be eliminated. Finally, an optimal line-pair will be identified with symmetrical considerations. No railway is detected if no optimal line pair is found. After finding the railway lines for the bottom layer, the optimal line pairs for each upper layer will be extrapolated sequentially. A line is considered to be potential railway line if it is a well-connected line. A line is identified as well-connected line if the gradient orientation of each pixel along this line is equal to the slope of the line. The iterative upper layer railway extrapolation is to extrapolate the most well-connected line for each upper layer, until no well-connected lines could be found.

3.1.3 Light-rail Train Detection

The LRV detection module is the core to the LRV collision warning system. The challenge lies in dealing with severe variation in the vehicle appearance due to various illumination and weather conditions and the constant changing of distances between the front vehicle and the camera. In order to sustain a robust and highly reliable performance, we adopt a multiple feature fusion approach in detection-level design, and a hierarchical modular structure approach in system-level design. The features we use include structural features such as edges and corners, and color information such as color histogram in HSV space, and also special patterns commonly possessed by vehicles, such as shadow. We also design a hierarchical structure for efficiently combining sub-modules (such as Rough Search Module, Far-distance Detection, Night-time Detection, Tracking, etc.) and a buffering mechanism that allows the system shift between modules if the current detection is not convincing. These designs allow our system to achieve extremely high robustness with zero missing detections through field tests. However, the system design is not the focus of this paper, and therefore will not be further explained in detail.

3.1.4 Distance Estimation

Distance estimation in our system is measured using geometric information. As shown in Figure 13, the blue dot represents the camera, with a height of H_C measured from the floor. D is the horizontal distance between the camera and the real front LRV. H is the height of the LRV. The camera has a focal length of f, and a CMOS pixel ratio of d. AB is the image plane of the camera. x is the distance between the image plane bottom to the floor. h is the LRV height in pixels on the image plane, and y_b is the y coordinate of the LRV bottom to the bottom-left corner of the image plane. According to the similar triangle theorem, we can develop the following sets of equations.

$$\frac{d \cdot y_b + x}{H_C} = \frac{D - f}{D} \quad \Rightarrow \quad D = \frac{f \cdot H_C}{H_C - (d \cdot y_b + x)} \tag{11}$$

$$\frac{d \cdot h}{H} = \frac{f}{D} \quad \Rightarrow \quad D = \frac{H \cdot f}{h \cdot d} \tag{12}$$

Equations above shows two different approaches to calculate the distance D. For each frame, if the LRV is detected, two estimated distances will be calculated using the above approaches. If the two estimates are close to each other, the LRV bounding box is believed to be well fit and the distance is accurate. Otherwise the Distance Estimation Module will send a message to the LRV Detection Module to ask for a more accurate bounding box result.

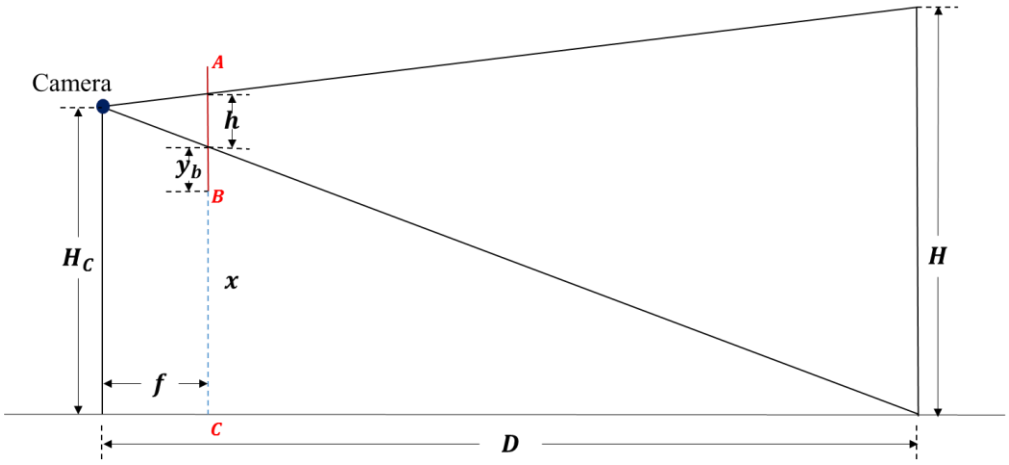


Figure 13 Geometric relationship of camera calibration, image plane, object height and distance

3.2 Framework of the Light-rail Vehicle Detection

The LRV Detection Module aims at detecting and tracking four types light rail vehicles for each video-captured frame. The module outputs a Boolean value indicating the existence of train for current frame, and also the exact bounding box location if the LRV is detected. The objective is to design and implement a real-time application that performs accurate train detection with multiple train types under a variety of environmental conditions. In order to fulfill the requirement, there are several principles we should follow when designing the detection framework. The Train Detection Module should be able to:

1. Perform successful detection when there is train in the front
2. Provide accurate bounding box of the train if detection is made (otherwise it will influence the accuracy of the distance estimation)
3. Keep tracking of the train once successful detection has been made
4. Ensure no missed detection of trains closer than 20m
5. Try not to make false detection when there is no train in the front
6. Verify the detection result as the train comes closer and eliminate false results even if the detection was wrong at the first time
7. Finish processing in a limited time

The difficulty of fulfilling the above requirements lies in dealing with complex situations caused by the change of train appearance, weather condition, illumination, or background condition, etc.

For example, the appearance of the train changes greatly as the distance decreases from far to near. When the distance is far, the train is complete with left, top, right and bottom boundaries distinguishable from the background. It also has a pair of long railway lines that could help locate the train position. All four types of trains process similar edge patterns at the distance, so they could share the same detection method. However, because of the distance, the feature of the train is not clear and thus not reliable. While at a closer distance, normally when the train is less than 15 meters away, the appearance starts to change. There is no more complete shape and even no top and bottom boundaries when the train gets closer. The railway information would be little or null. However the features of each type of train become clear and contain more information such as colour, corner,

edge, etc. Different train type will have different set of features, and will not share a common detection method.

Another example would be the change of color and intensity caused by different illumination condition. The figures shown below are all red trains, but in different environmental conditions varying from dark night, dim indoor lightning, daylight with shadow casting on, to bright sunlight. The red and blue color of each train is very different from one another. This variation would greatly influence the performance of detecting methods using color as their main object feature.

In order to overcome the difficulties and achieve the objectives discussed above, the Train Detection Module is divided into different functional blocks, each aiming at one detection problem in one specific case. The connection of each block and the overall decision making is of great importance for the whole module to perform fluent and reliable detection.

3.2.1 The Overall Framework

A number of different functional modules are designed in order to handle the varieties in scaling and appearance of the train. For example, the Far-train Detection module aims at recognizing any trains at a distance from 40m to 15m, while the Near-train Detection modules detect trains closer than 15m. The trains in our task could be roughly divided into two types according to their body color, and we believe it is more efficient to recognize them with different sets of features, and thus two near-train detection modules are designed. All functional modules are arranged in a hierarchical structure to perform the detection and tracking of the trains, as shown in Figure 14 and Figure 15.

The first-time detection (Figure 14) starts after the railway detection. By examining the railway detection result, the system could have a rough understanding of whether it is a far-train case or near-train case, and corresponding detection modules will be applied. Once the train is detected, the detection record will be kept and the tracking mode will be activated. Note that although the detection relies on the railway detection result, the system can still function normally without the railway information. In such case, a Shadow Detection module is performed to quickly scan the frame and tell the system whether or

not a possible train exists. If yes, the train detection module can be applied to perform finer detection within the detected Region of Interest (ROI), instead of over the whole frame in a sliding window approach, which will be extremely time consuming.

The Shadow Detection module is an important module at the early stage of the hierarchy. The major function of this module is to detect a small shadow-like area at the bumper region of the train, which we believe is a quite discriminative feature indicating the existence of a train. The advantage of this module is its fast detecting speed. It rejects most of the locations in the ROI that are unlikely to have a train, while keeps a few locations that could possibly have a train by very limited computation efforts. Thus, the Shadow Detection module is adopted in the detection hierarchy, prior to any other finer detection modules. (The flow diagram in Figure 14 does not show this module before the Near-train Detection modules because shadow detection function has been embedded into near-train detection modules.) If no railway detected, or the detection result does not seem reasonable, another shadow detection module will be triggered, which has the same functionality with the previously described one, but instead of conducting patch detection within the region of interest, it performs detection from the bottom of the entire frame.

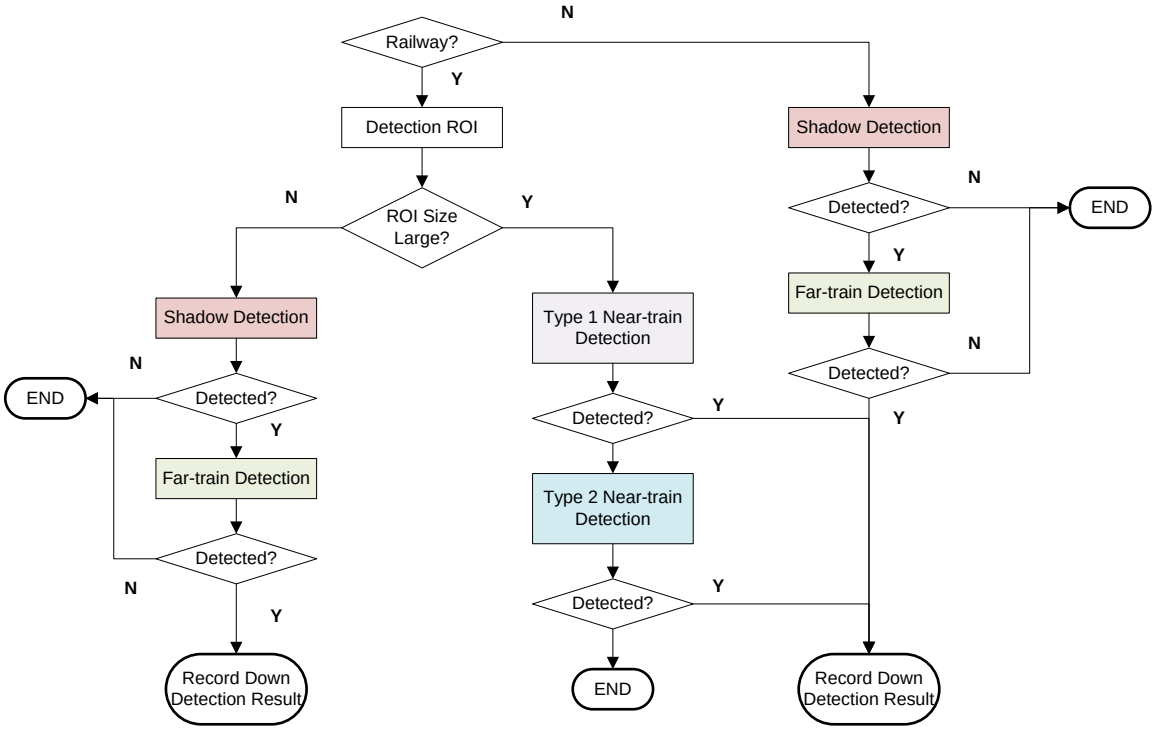


Figure 14 Flow Diagram of First-time Detection

After a successful detection, the tracking mode will be automatically activated. Tracking systems for far-distant-trains and close-distant-trains are designed separately, and are selected according to the detection result. Figure 15 only shows the flow diagram of far-train tracking system. The Far-train Tracking function that performs object tracking for continuous frames is the core function in the system.

In addition to the core function, we have also added two more groups of functions, forming an exceptional case handling scheme and a re-detection buffering scheme that could significantly improve the reliability of the whole system. The exceptional handling scheme is activated when the tracking result is not trustworthy. We have kept a score for each tracking indicating our confidence for the result, and once this score is lower than certain threshold, the tracking is regarded as a low-confidence-tracking, and a verification function that measures the similarity between the tracking result and the real train will be conducted. If the verification test has been passed, the tracking will be continued, and if not, the tracking will be regarded as an unsuccessful tracking. The exceptional handling scheme can solve the shifting problems in a tracking system. When the tracking algorithm is not accurate enough, the tracked object location might be shifted from the real object location, and by tracking the shifted object, the error could increase gradually, leading to a high probability of losing a track. The exceptional handling scheme prevent such case by immediately identifying any low-confidence-tracking, and evaluating the result through a set of similarity tests, so that further actions could be taken based on the similarity. In addition to solving the shifting problem, the scheme can also reduce the number of false positive detections by identifying and rejecting the falsely detected backgrounds at the verification stage.

The re-detection buffering scheme will be activated once the tracking fails at the current frame. It is unfair to immediately claim no train found in this frame and reset all results to their initial status, especially when we already have a number of successful tracking results beforehand, since we believe that the train cannot suddenly disappear. It is reasonable that we give the system a chance to re-detect the train while copying the previous tracking results. However, the re-detection chance cannot last infinitely, in case there is really no trains in the frame. Based on the above considerations, we introduce a re-detection

buffering scheme. As long as the number of previously successful detection/tracking exceeds certain threshold, we will allow the re-detection. If the train is re-detected, the tracking will be continued and the records will be updated as normal. If the re-detection fails again, a record of failed detection will be stored. The re-detection will be allowed until the number of failed detections exceeds certain limit, and the system will claim no train found at that time. The re-detection buffering scheme has greatly reduced the chance of missing detections and thus increased the overall accuracy of the whole system.

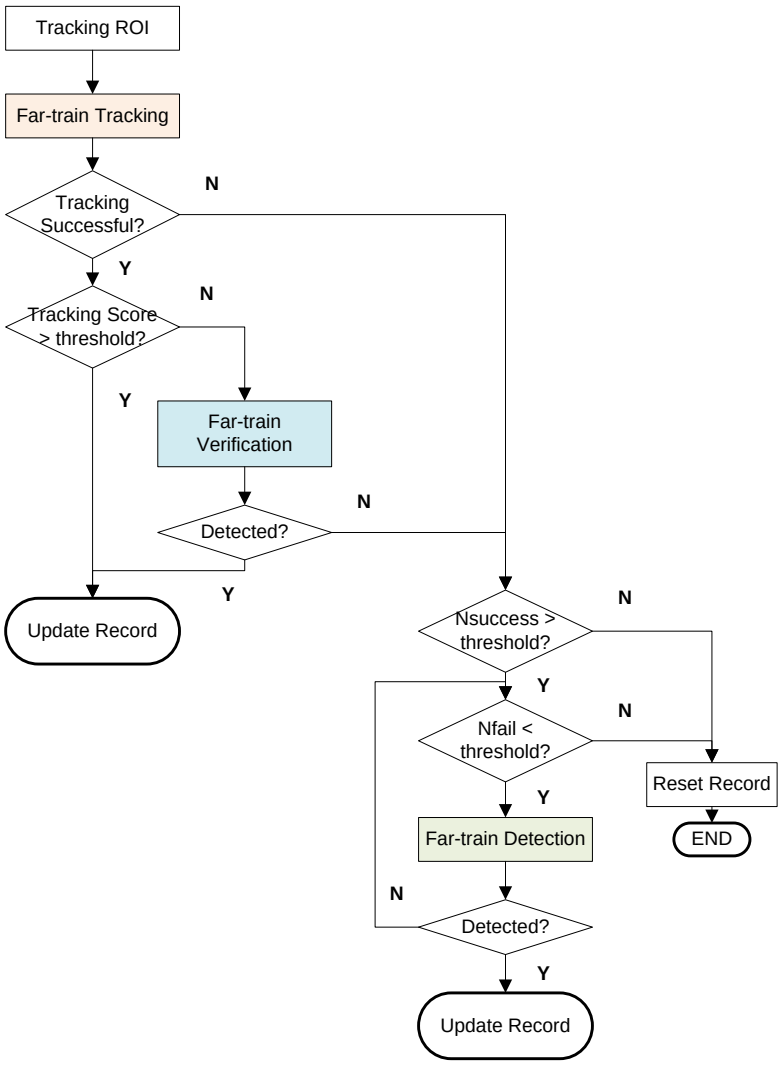


Figure 15 Flow Diagram of Tracking

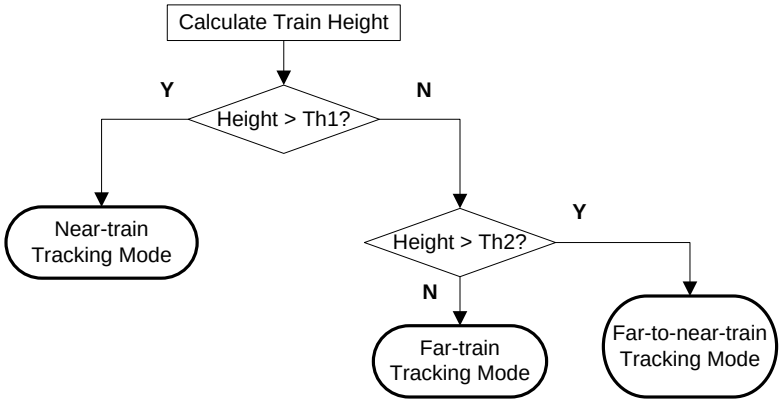


Figure 16 Flow Diagram of Transiting Between Modules

3.2.2 Modular Design of Vehicle Detection System

3.2.2.1 Initialization Module

At the beginning of the Train Detection Module, an initialization is conducted based on the railway situation of current frame and the detection results of previous frame to return an index indicating the most suitable detecting method for this frame, the corresponding functional block is then called to perform the detection. In order to make the illustration clear, the initialization is first divided into six cases based on Railway Result, Near-Train Record, and Far-Train Record. Each case will then be further explained, and an overall initialization flow diagram will be displayed after the explanation.

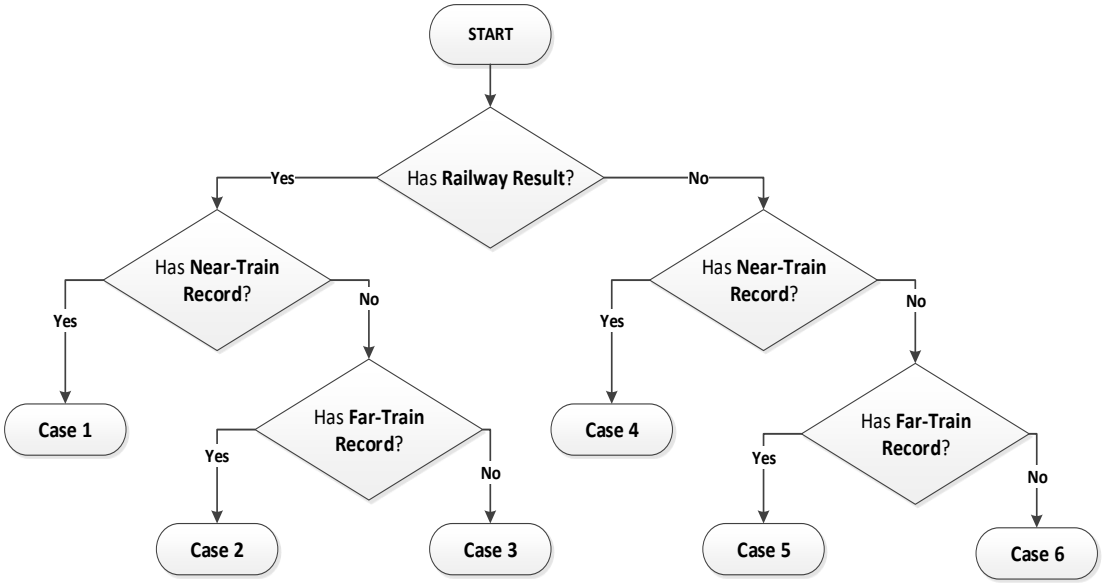


Figure 17 Initialization (Six cases based on previous detected results)

Case 1 and Case 4:

Both Case 1 and Case 4 have the Near-Train Record, meaning that the train at a close distance was successfully detected at previous frame, so the action of this frame is just tracking the previous detection result. Since Near-Train Record also stores the train type, the initialization result would then be Near Train Tracking of that certain train type.

Case 2:

Case 2 has the Railway Result, and the Far-Train Record, and no Near-Train Record. Having Far-Train Record means that in the last frame a far-distanced train was detected. It is reasonable to follow the record and call the Far Train Tracking block, but in order to be more cautious, a simple comparison is made between the Railway Result and the Far-Train Record beforehand. If the two results match, then the initialization result is Far-Train Tracking. However if they do not match, then the record is possible to be corrupted or the previous detection could be wrong, so it is passed to Case 3 where no Far-Train Record is used and the detection purely relies on the Railway Result.

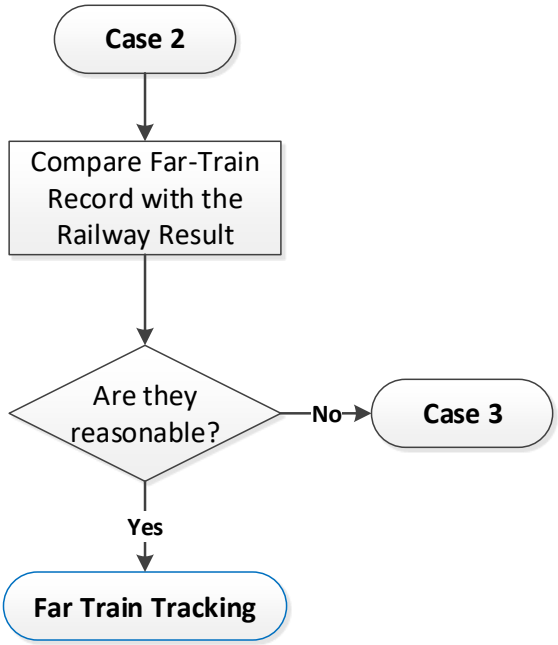


Figure 18 Initialization (Case 2 further explanation)

Case 3:

Case 3 does not have Far-Train Record or Near-Train Record but only the Railway Result, meaning that Case 3 would be a first-time detection based on the railway situation. The decision is made according to the length of the railway pair: if the railway is long, it is either the case of no train or far-distanced train, so after a quick rough searching process, the detection is either terminates or sent to far train detection; if the railway is short, there is likely to be large trains, so the full search method is used. Since we do not know the train type yet, so the three type-specific full search methods would take turns to be called.

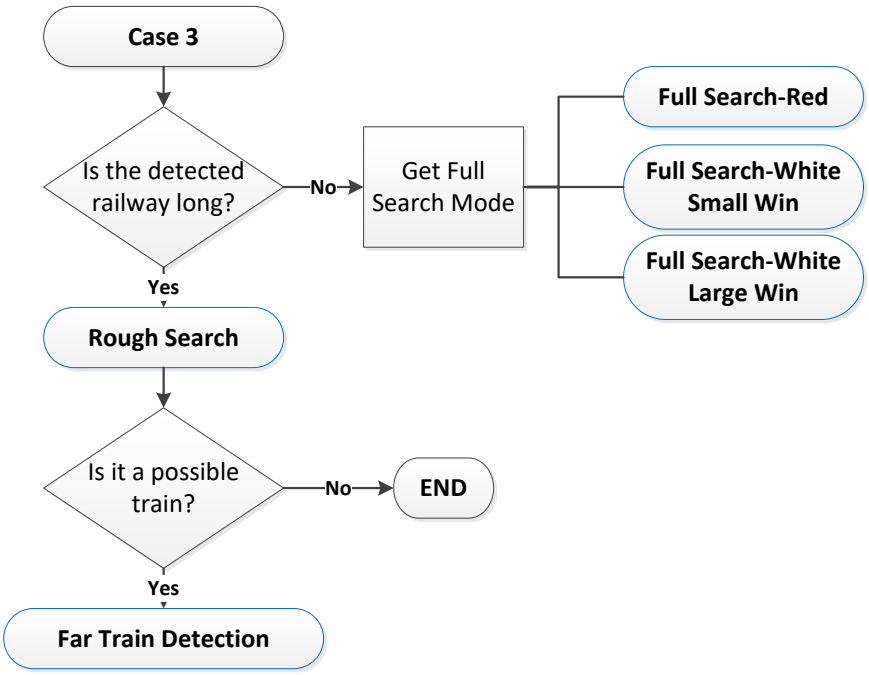


Figure 19 Initialization (Case 3 further explanation)

Case 5:

Case 5 has no Near-Train Record, no Railway Result, but has Far-Train Record. Clearly it is in the tracking mode of a far-distanced train. However, there might be the case when the train comes too close that far-train-detection method cannot keep tracking. In such case the initialization would call the Far-to-Near Train Detection methods instead.

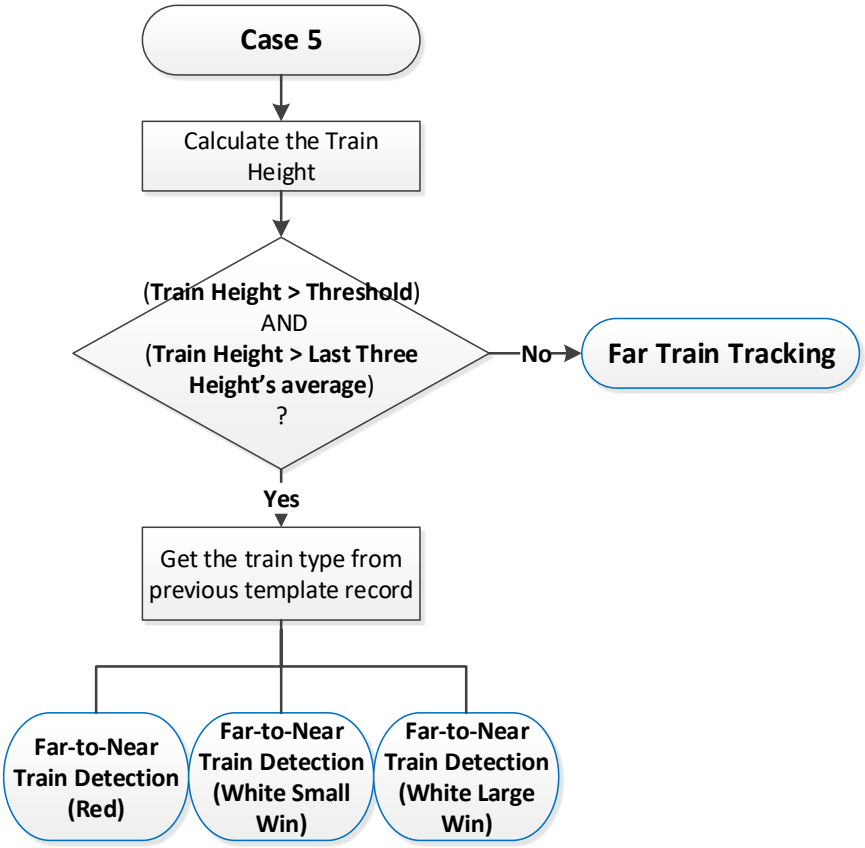


Figure 20 Initialization (Case 5 further explanation)

Case 6:

Case 6 has none of the three results, so the only solution left for it is conducting the Full Search. Similar to Case 3, the Full Search method for different type of train takes turn to be called.

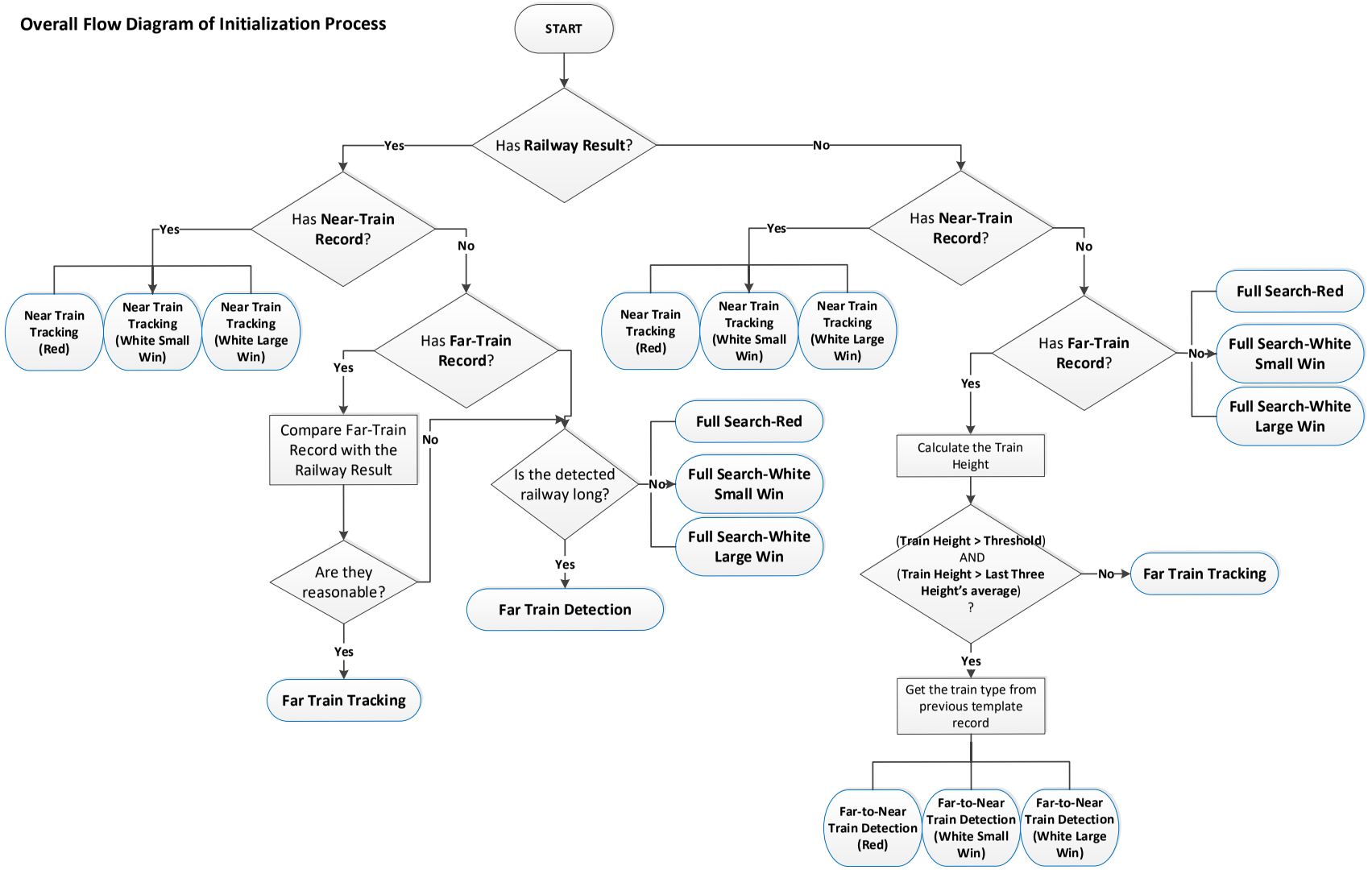


Figure 21 Overall Flow Diagram of Initialization Process

3.2.2.2 Shadow Detection

The purpose of this block is to quickly locate the possible location of the train based on the track information provided, and conduct a fast and simple test indicating whether a possible train exists.

There are two observations of the property of a possible train: 1). The bumper is right above the end of the track pair. 2). The bumper area has very sharp contrast and distinguishable color distribution. Thus, the Rough Search Block tries to use a small rectangular area right above the track to estimate the possible existence of the train.

It first defines an initial searching area based on the tail of the detected track as shown in the figure below. The searching area will keep shifting upwards until it returns a true result or meets the stopping criteria. The decision of whether this area contains a train is made based on the intensity, color and edge information within the searching block.

By observation, the most discriminative part is not the shadow underneath the train. Instead, the rectangular patch that contains the transition boundary of the train-body to the train-bumper (Figure 22 (e)) provides much more information of the existence of the train, and should thus be used as the “shadow” patch to be recognized in the shadow recognition process.

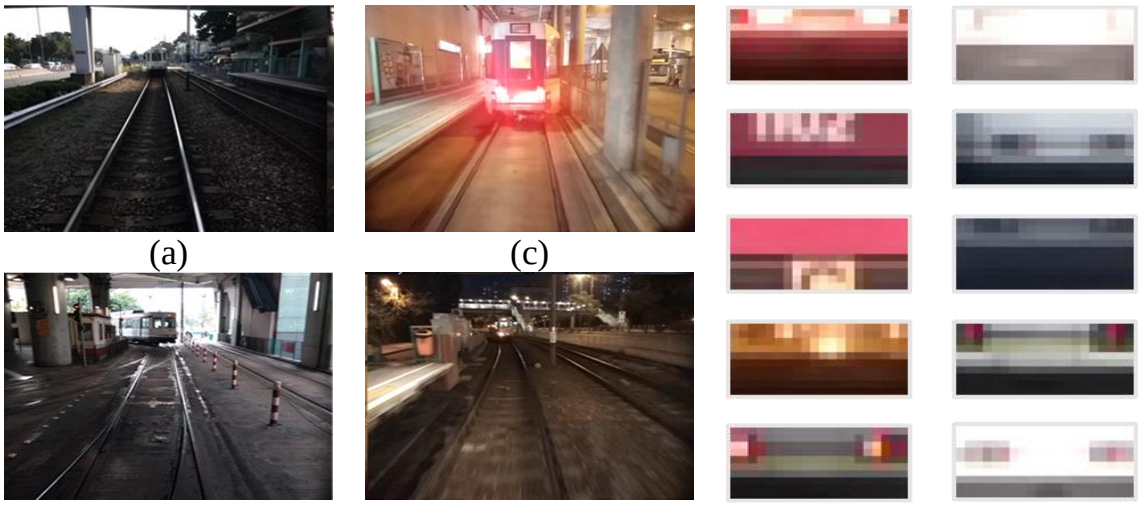


Figure 22 Train shadow under different environment ((a) uneven illumination (b)complex texture (c) strong tail-light (d) night and (e) example shadow patches)

Based on the above observations, we propose a multiple feature based cascaded detector that can effectively and efficiently detect the shadow patch of the train in various scaling, illumination, and weather conditions. We also propose a decision tree based shadow patch detector that further improves the detecting accuracy and speed. Subsequently we propose a modified decision tree classifier that takes each binary node as a weak classifier. The final prediction and confidence measures are obtained by combining all weak classifier predictions.

High reliability is essential for any vehicle detection system. A reliable detection system should perform almost no missing detections, and at the same time as few false alarms as possible. This requirement is the same for shadow detection, since it acts as an early stage coarse detection module in the whole train detection system. From the sample shadow patches in Figure 22(e) we can see that due to the non-ideal illumination, the influence from tail lights, and the difference of rear-view appearance, and the intra class variation of the shadow patches are high, making it hard to be modeled by one or two simple features and thresholds. More complex models, however, will make the detection slow, especially when the classification is performed at each candidate location as the detector slides over the entire frame. Observing that 1) the number of non-shadow patches is much larger than shadow patches in each frame, and 2) some non-shadow patches can be easily differentiated while some are more difficult, it is our strategy to design a cascade detection

structure with multiple stages. Each stage, representing a different level of differentiation difficulty, rejects a certain amount of non-shadow patches, and only the testing patch that passes all tests will be classified as positive, as shown in Figure 23

Among all patches being tested, most of the non-shadow patches will be rejected by the first stage, so no further classification is needed, and thus the computation time will be greatly reduced. The features used for classifying shadow/non-shadow patches should have the most obvious and discriminative features. Observing that all true shadow patches have sharp horizontal edges, the feature used for stage 1 classifier is the histogram of oriented gradients. All background patches without strong horizontal edges are rejected at this stage. However, non-shadow patches that have horizontal edges cannot be differentiated. They will be tested in stage 2 using another set of features. Color is an important piece of information, and the histogram of Hue value in the HSV color space can be used as the features for this stage. Still, there could be background patches that have similar edge and color distribution. We designed a new feature called line-contrast for discriminating such negative patches at the third stage. It compares the average intensity differences between horizontal lines, since most lines above the bumper have larger intensity values as compared with lines below the bumper. Different from comparing two single neighboring lines, this feature compares the intensity difference of one line with a bunch of contiguous lines below it. This approach increases the discrimination power of the line contrast feature since shadow patches always contain multiple low-intensity lines.

The classifier used in each stage uses some linear decision rules with weights and thresholds manually designed and tuned by experiments and observations.

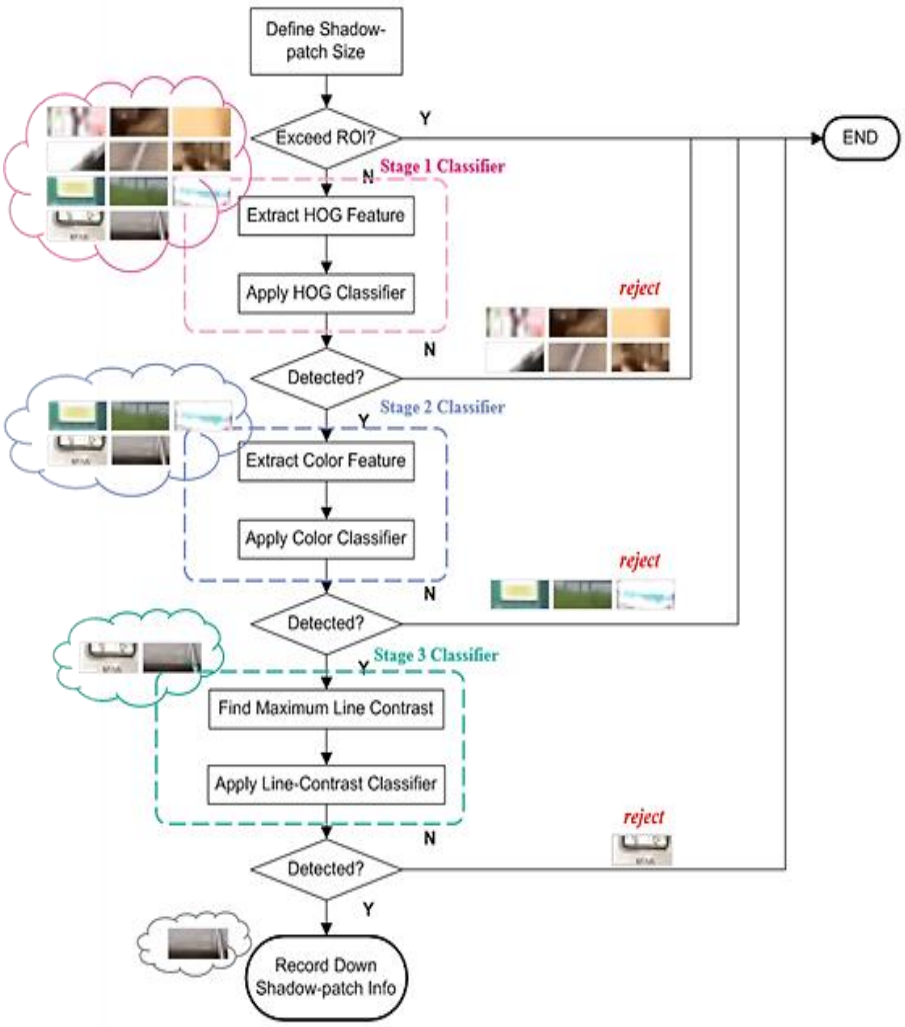


Figure 23 flow diagram of proposed 3-stage-cascade shadow detector, and illustration of negative patches rejected by each stage

3.2.2.3 Far Train Detection Module

Far Train Detection aims at detecting and localizing the train at far distance (>15m). The main detecting method is to compare the similarity between the HOG features of the testing patches and the HOG of 23 templates stored in advance. After a successful detection, the following detections will simplify the detecting process by selecting fewer testing patches and templates based on previous detection result, and thus greatly reduce the computation time. Still, false alarm cases might exist if the background has very

similar edge pattern with the train, and a further verification test is needed to ensure the accuracy of the detection. This is why the Far Train Verification is conducted every time when the train distance approaches 20m.

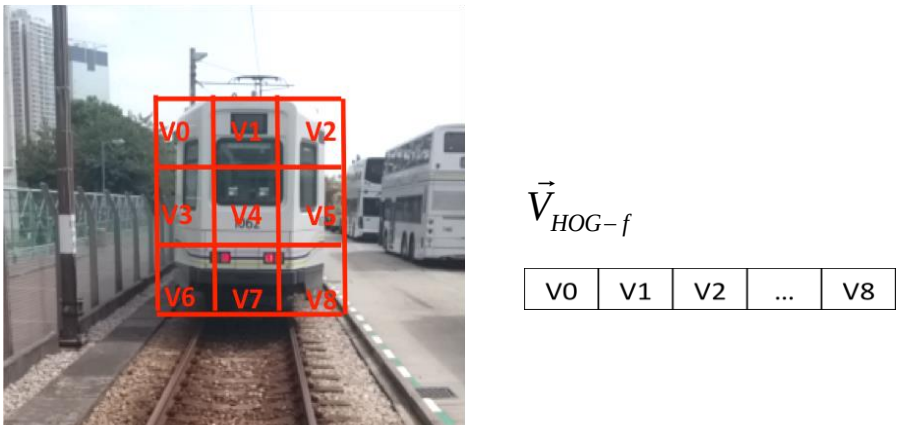


Figure 24 HOG feature extraction of testing patch

As shown in Figure 24, The testing patch is divided into 9 cells, each cell containing one HOG vector. The HOG feature of this patch is built by concatenating 9 vectors together, denoted as $\vec{V}_{HOG-f}$.

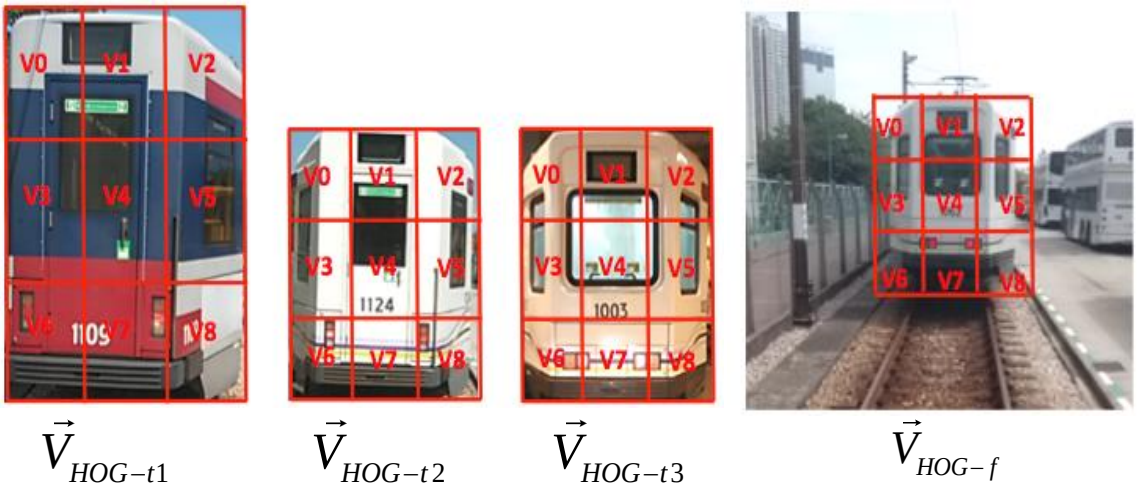
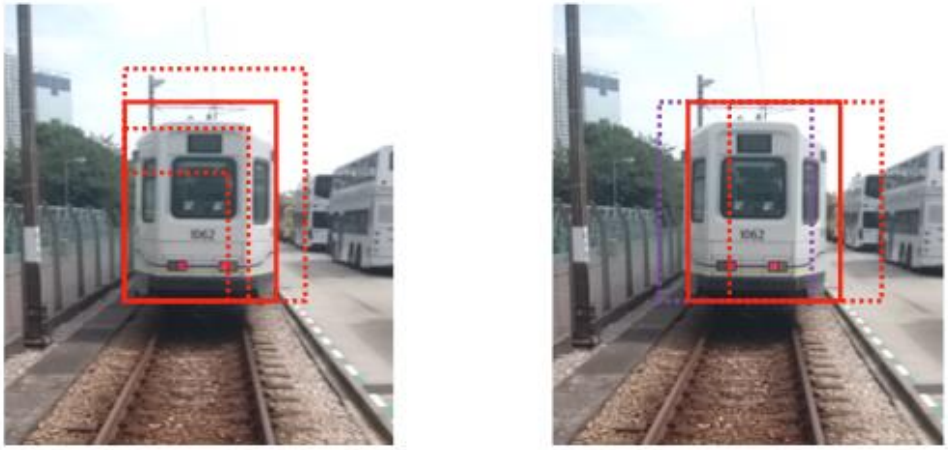


Figure 25 Comparing HOG features extracted from testing patch with HOG features

The HOG feature of each template has already been calculated and stored in the program since start ($\vec{V}_{HOG-t1}$, $\vec{V}_{HOG-t2}$,etc.), and is now compared with the HOG of the testing patch.

The sum of squared differences (SSD) between two HOG features is calculated as the similarity measurement. The template is matched when the SSD value is smaller than a certain threshold, and the best match is defined by the template with the minimum SSD value.

Refinement is still needed after finding the match for the testing patch. The testing patch is shifted horizontally and vertically with limited steps, and also resized by enlarging and shrinking the patch with one corner fixed each time, forming a group of candidate patches. Each candidate patch is compared with the matched template and the one with the least sum of squared difference is selected as the object patch.



(a) Resizing (b) Shifting

Figure 26 Shifting and Resizing the testing patch ((a). resize the patch into smaller and larger size with left-lower corner fixed; (b). shift the patch left and right horizontally)

3.2.2.4 Close-range Train Detection

The shape of close-distance train is quite different from remote-distance train, and it cannot provide the complete framework of the train when distance is too close. However, the local features are clearer and more reliable to detect large trains. Unlike the global features that treat the train as a whole, the local features focus more on each detailed part of the train, and thus some features suitable for one train type may not suit for another train type, so according to the type of the train, the Near Train Detection is divided into three different detection blocks: Red Train Detection for Phase2 and Phase 3 Trains, White

Train with Small Window for Phase 4 Trains, and White Train with Large Window for Phase 1 Trains.

Close-range Red LRV detection

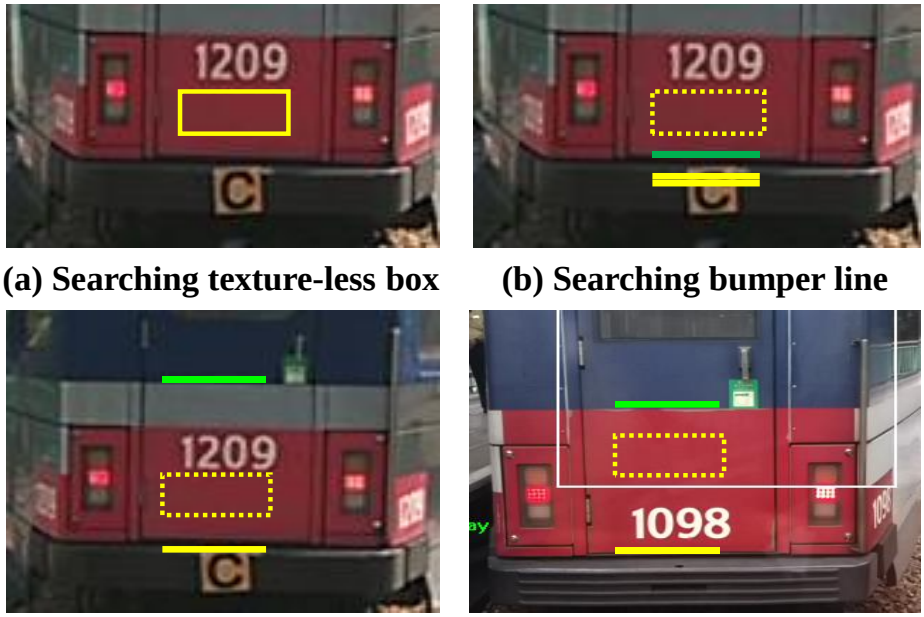
(a) White Train (Large Window)

(b) White Train (Small Window)

(c) Red Train

Figure 27 Window features of white and red trains at close distance

Unlike the close-distanced white trains, which have clear boundaries between the window and the white train body, the window boundary of the red train is not so obvious, especially in the daylight condition, when the window is normally black and the color around the window is blue (Figure 27). The intensity difference is always large for white-black or red-black comparison, while small for blue-black case as shown in the red train. Even in the RGB color space, the difference in B value for these two colors is too small to make reliable detection. This observation eliminates the chance of using the window lines as local features to detect near trains as used in white train cases. However, the red train still has an important feature: it is Red. It is quite rare to find such a large texture-less red area in the background or in other types of trains. In addition to this, the boundary of the upper-blue and lower-red areas, and the bumper line of the train are also good features defining a near red train.



(a) Searching texture-less box **(b) Searching bumper line**

(c) Searching blue line **(d) Searching blue line**

Figure 28 Illustration of Near Red Train detection

Firstly, this module searches for a red, texture-less box. For each pixel in the testing box, we compute its horizontal and vertical differences with neighboring pixels: $dy = I(x,y) - I(x,y+1)$ $dx = I(x,y) - I(x+1,y)$. The average is computed afterwards: $DX = \frac{1}{N}\sum dx$, $DY = \frac{1}{N}\sum dy$. The box with both DX and DY smaller than certain threshold is considered as a texture-less box. A pixel is defined as "red" when its color angle, computed by RGB values, lies in some pre-defined limit, and the whole testing box is considered as red if more than 60% of its pixels are "red" pixels.

Then it starts to search for the bumper below the box. We define the horizontal line contrast as average R channel intensity of green line minus the average R channel intensity of the blue line (multiple blue lines are involved while only one green line is involved). Then we search for a horizontal line giving the maximum contrast within the ROI, and this line is thus considered as the bumper line.

Then Search for the blue line. Although there are two types of red trains, one with grey band between blue and red area while one without, it is interesting that the blue line of both trains can be found using the same method. This is because the difference between

grey-blue and the difference between red-blue are both large in the R channel, and this is also the reason we use R channel pixel values more frequent than pixel intensity. First, we use a linear equation to predict the position of the blue line based on the bumper position. Then we search for the blue line around the predicted region with the maximum contrast described in bumper detection.

Close-range White Trains

(a) Phase 1 (no light) (b) Phase 1 (with light) (c) Phase 4 (no light) (d) Phase 4 (with light)

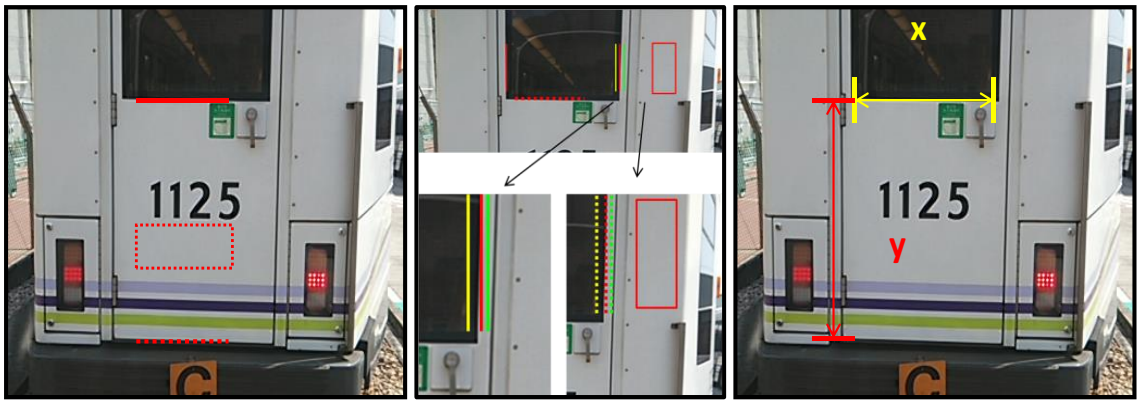
Figure 29 Window features of Phase 1 and Phase 4 trains with different lighting condition

There are altogether four types of trains: Phase 2 and Phase 3 trains are red trains, sharing a same detecting algorithm; while Phase 1 and Phase 4 trains are all white trains, but with separate detecting functions. The main reason why we use different methods is the windows of the train look very differently for the two types.

As shown in Figure 29, the window of Phase 1 train is wider than Phase 4 train. This would cause trouble if we use the window width estimating the train boundary, or use the bumper position estimating the window region. Another difference lies in appearance of the corner. Phase 1 train has round corners while corners of Phase 4 train are right angle. Finally, Phase 1 train shows a thicker black band around the window when the light inside the train is on, while the black band of Phase 4 train is much thinner. These differences in windows matter because window is a key feature for white train detection at close distance, just like color red being the key feature for red train detection. Thus it is necessary and straightforward separating white train detection into detection for trains with small window and trains with large window.

The first several steps of White Train (with Small Window) Detection are quite similar to the ones of red train detection, since boxes of texture-less area and clear bumper line are common features shared with all types of close-range trains. The only difference is the pixels in the box mostly being white pixels. The window is then searched within a region of interest linearly predicted by the position of the bumper line. Three boundaries including left, right and bottom are searched in order to make sure it is the pattern of window detected, other than some random vertical lines generated by some background. Finally the bounding box of the whole train is estimated by the position and the size of the window.

The detailed detection algorithm is described as follows:



(a) Texture-less box and bumper line **(b) Left and right window boundaries** **(c) Relationship between window width and window-bumper width**

Figure 30 Illustration of Near Train detection for white trains with small window

1. Searching for a white, texture-less box
2. Searching for the bumper line
3. (Searching method is the same as in Red Train Detection, results shown in Figure 30(a).)
4. Searching for the bottom line of the window (Figure 30(a))
5. The searching region is estimated by the bumper line position. Similar to the bumper line detection, the line with the maximum horizontal contrast is regarded as the window bottom line.
6. Searching for the left and right boundary of the window

The searching method is still finding lines with maximum contrast. However, this time the vertical contrast is computed and compared. After finding a candidate boundary line, some more features are examined to avoid false detection from texture-rich backgrounds. First, the pixel-pair difference is tested for two lines left and right to the candidate boundary, correspondently marked in yellow and green in Figure 30(b). If the candidate boundary passes this test, a thick rectangular region parallel to the boundary is then examined. The region with variance smaller enough and white pixels more than 60% is considered as a valid white region. Only the candidate boundary that passes the above two tests is regarded as the window boundary of the white train.

Verifying the train with the window width and bumper-window distance (Figure 30(c))

After detecting the window, the width of the window (denoted as x) and the distance between the bumper and the window bottom line (denoted as y) are calculated, and the relationship between x and y is tested to see if it satisfy the relationship of a possible Phase 4 train. Finally, the entire bounding box of the train is estimated based on x and y values.

White Train with Large Window

As described before, the window of this type of train has many unique features. The band around the boundary of the window is black and thick, making it a reliable and recognizable feature for any illumination conditions with trains at any distance closer than 20m. The black band consists of two vertical straight lines, one horizontal straight line, and two round corners. Based on this observation, the feature detector is designed as a five-cell, window-shaped polygon, shown in Figure 31.

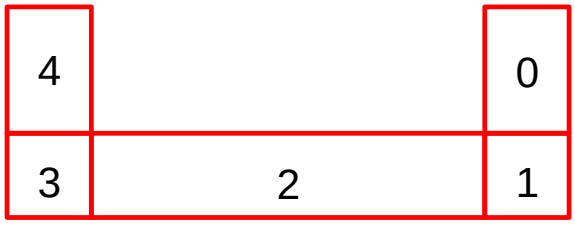
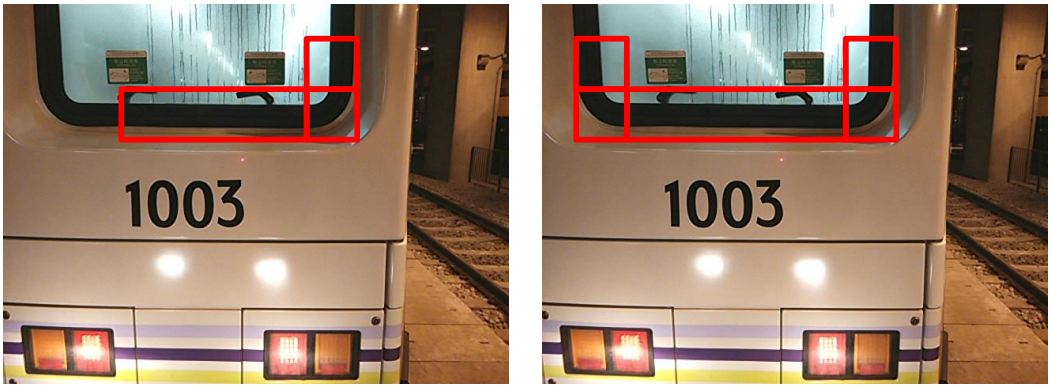


Figure 31 5-cell HOG window detector of white trains with large window



(a) Searching for right portion **(b) Searching for left portion**

Figure 32 Search scheme for white trains with large window

Each cell contains a 9-bins' histogram of edge orientations. The testing patch with histograms in all five cells satisfying the matching criteria is considered as the real window. The matching criteria are illustrated as follows:

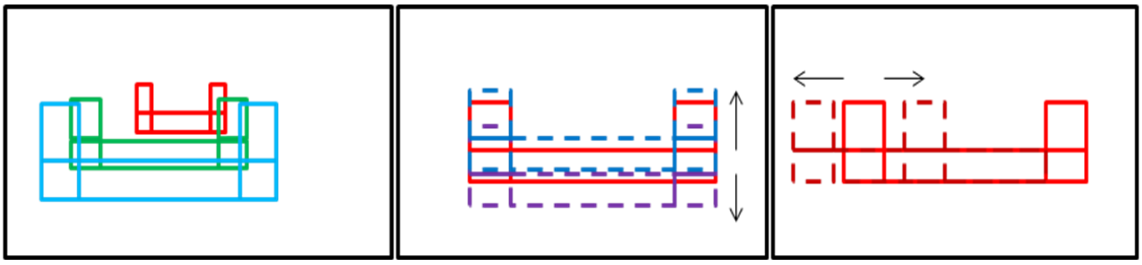
Step1: Searching for the right portion of the window

The right portion is considered to be matched if Cell 0 contains strong vertical edge; Cell 2 contains strong horizontal edge; While Cell 1 contains equally distributed 20°-40°,40°-60°, and 60°-80° edges.

Step2: Searching for the left portion of the window.

After the right portion is found, the 5-cell polygon is extended to the left to search for positions that match with the left-window criteria. The left portion of the window is considered to be matched if Cell 4 contains strong vertical edge; Horizontal edge dominates Cell 2; And Cell 1 contains equally distributed 100°-120°,120°-140°, and 140°-160° edges.

The way of searching the candidate 5-cell-polygon patch includes an initial search of a set of resized polygons to find a rough position of the candidate patch (Figure 33 (a)), a refinement search of a set of vertically shifted polygons to further secure the position of the right window portion (Figure 33 (b)), and a final search of left and right extended polygons to find the left portion of the window (Figure 33 (c)).



(a) Initial Search **(b) Refinement Search** **(c) Final Search**

Figure 33 Full Searching steps for detection of white trains with large window

3.2.2.5 Buffering Mechanism

Imagine a situation where we already have 100 successful detection of the train in front of us. However, due to some reason (probably a sudden change in illumination, or a large movement of the target object), we cannot detect the train in the 101th frame and neither can we quickly regain the detection. Then, at least in a certain period, we lose the detection. The buffering mechanism is thus introduced into our system to avoid such situation. It counts the number of successful detection made previously. Once the tracker fails at detecting the object, the buffering mechanism would give the tracker a chance to copy previous detection results while keep trying to regain the detection until it reaches the buffering allowance.

The mechanism is introduced into all tracking methods including Far Train Tracking, Near Train Tracking-Red, Near Train Tracking-White Small Win and Near Train Tracking-Large Win. Here only the Near Train Tracking for Red Train is taken as an example. There are several counters important to the flow of the mechanism:

1. successful count (stores the number of previous successful detections, and will be reset to zero once the detection fails)
2. buffering count (the times of allowing current frame to copy the results from previous frames)
3. failing count (the number of continuous failing detections, reset to zero when successful detection is made)

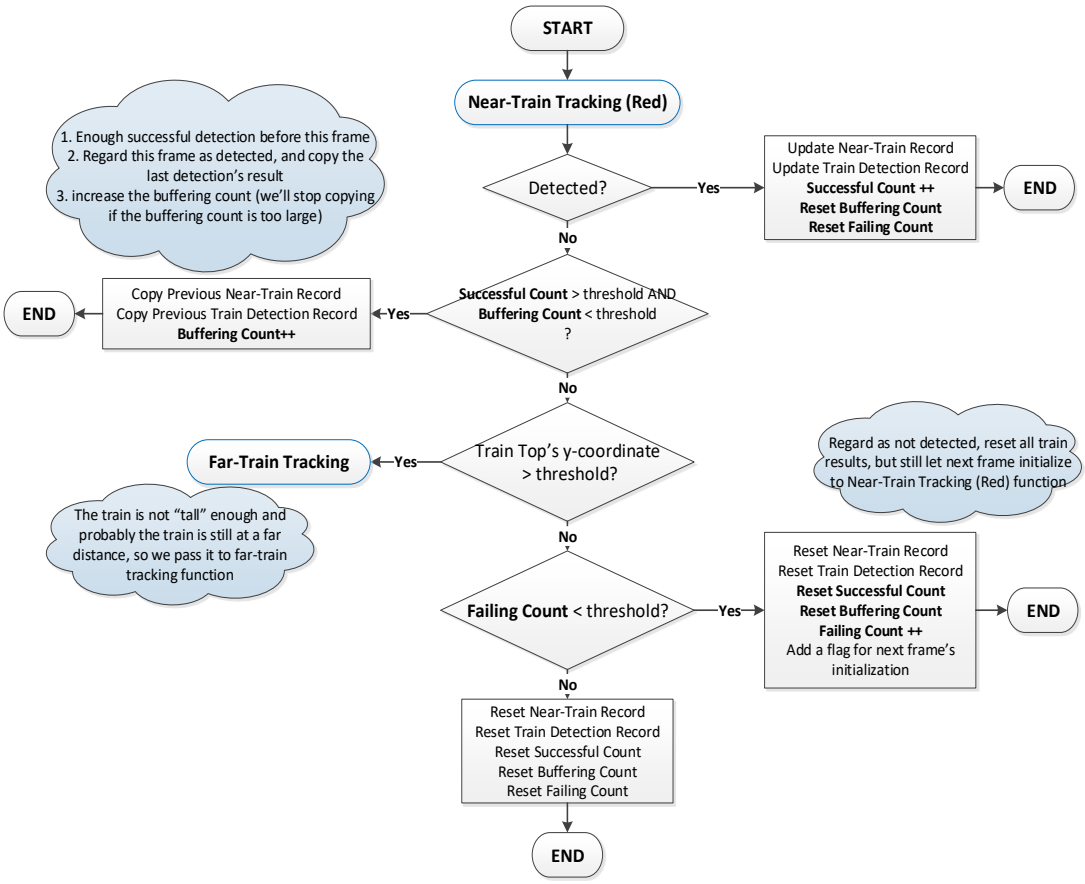


Figure 34 Near Train Tracking for Red Train with Buffering Mechanism

Block-Switching Mechanism

After initialization, a function block is assigned to perform the detection. However, the initialization could be wrong and the assigned method may not be the most suitable method for this situation. Thus, switching between blocks is needed. Block-switching mechanism is embedded in many detection processes. In fact, the example in the buffering mechanism itself contains a block-switching function when the detection switches from Near Train Tracking to Far Train Tracking under certain circumstances (Figure 34). Here is another example of switching between Far Train Tracking and Far Train Verification.

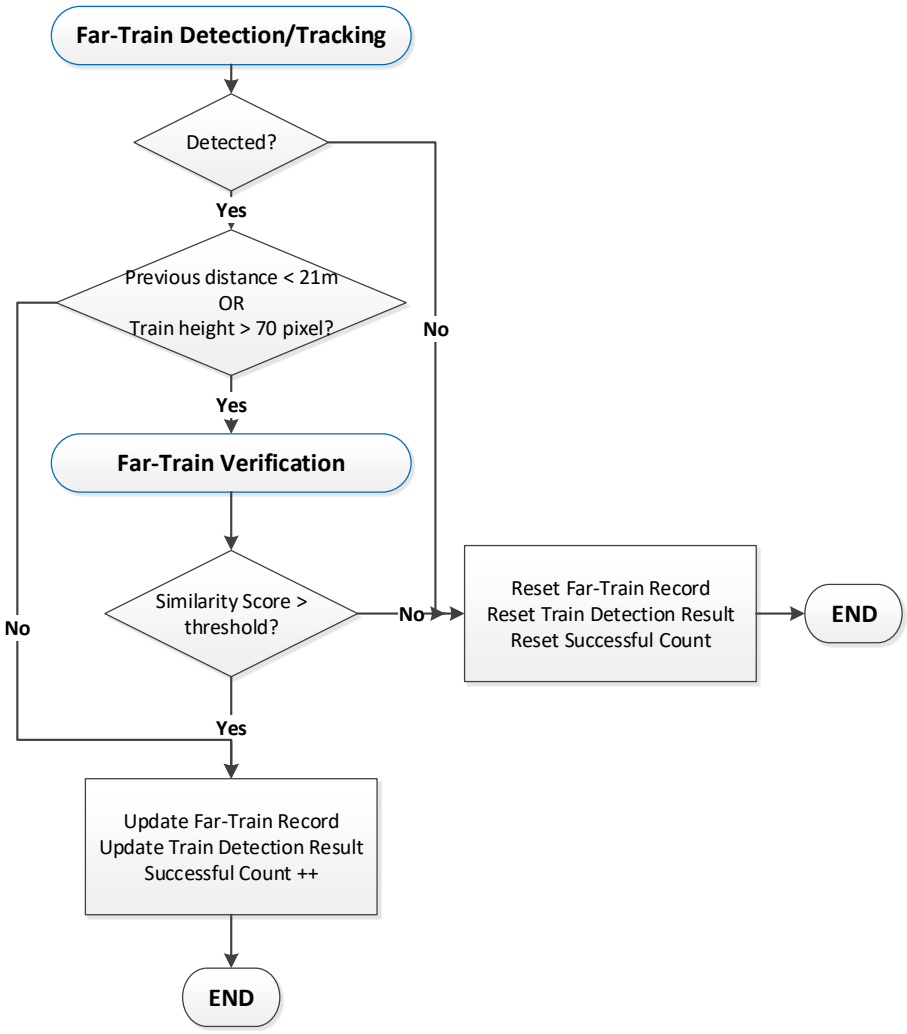


Figure 35 Block-switching mechanisms for Far Train Detection and Far Train Verification

Chapter 4 Enhancing System Performance

4.1 Far Train Verification Module

4.1.1 Introduction

Sometimes the background contains very similar edge patterns to the train. For example, in the indoor condition where the ceiling, floor, and the pillars construct very sharp horizontal and vertical edges, the HOG feature of such area will have SSD small enough to be called a match with one of the templates, and thus false detection would occur. In order to solve such problems, the “Far Train Verification Block” is introduced. This block aims at filtering out false detections by testing local features of the detected patch.

4.1.2 Feature Extraction

We select a set of features helpful to distinguish positive and negative samples to form a similarity test. The similarity test includes the following features: The difference between the height estimated from vanishing line and the height detected (larger difference indicating less possibility to be a train); the maximum contrast we obtained when finding the bumper; the ratio of valid pixel pairs from two lines above and below the bumper; the average difference (absolute value) of two lines above and below the bumper; the existence of smooth rectangular blocks (either red or white) above the bumper; and the pixel intensity variance (smoothness measurement score) of that block.

4.1.2.1 Edge

One obvious feature is the bumper line. A genuine train has a distinguishable bumper line where pixels below it are black and pixels above it are either red or white. Both red and white pixels have a large R value compared with black pixels in the RGB color format, so a possible solution would be comparing the R value differences between pixels above the bumper line and the pixels below.

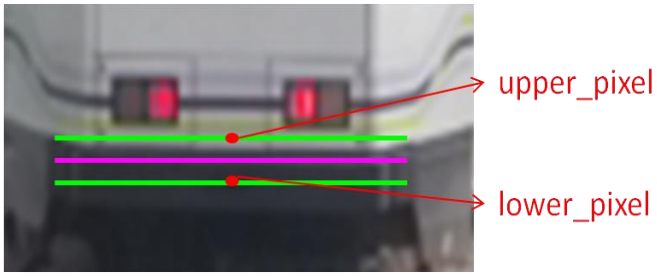
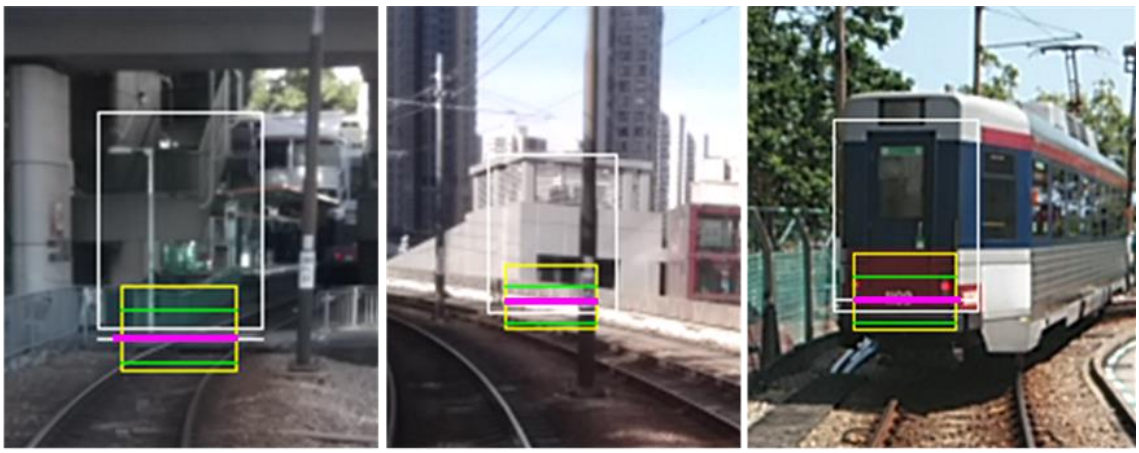


Figure 36 Bumper line (pink) and lines above and below (green) the bumper line

Figure 36 illustrates the position of three lines. The line marked in pink denotes the bumper line, and two parallel green lines denote the lines above and below the bumper. The pixel on the upper green line is denoted as upper_pixel, and the pixel right below this upper_pixel on the lower green line is denoted as lower_pixel. The upper_pixel and its corresponding lower_pixel define a pixel pair, and the difference of this pixel pair is defined as *R(upper-pixel)-R(lower pixel)*. The pixel pair with a difference larger than certain threshold is called a valid pixel pair. The idea of testing the bumper line feature is to calculate the ratio of valid pixel pairs over the total number of pixel pairs on this bumper line. Normally a ratio larger than 50% is considered as a pass on this feature test. Some false detected background patches like Figure 37 (a) and (b) are successfully eliminated using this test.



(a) Falsely detected background patch | (a) Falsely detected background patch | (c) Genuine train

Figure 37 Testing valid pixel-pair ratio for falsely detected background and genuine train

However, an important question is then asked: Will this test affect the detection of the genuine train? Or in other words, can all genuine trains pass this test? A negative example would be the red train shown in Figure 37(c), where the tail of the train is casted inside the shadow, leaving the object to be detect very dark. In fact, the average pixel pair difference in this case is only 35, making almost all pixel pairs non-valid pixel pair. The genuine train thus fails the test. One solution is to compare relative difference instead of absolute difference. The new difference is defined as $\frac{R(upper\ pixel) - R(lower\ pixel)}{R(upper\ pixel)}$. With this change, the feature test could filter out most falsely detected background patches without affecting the detection of real trains.

4.1.2.2 Texture-less Rectangle

Unfortunately, the bumper line feature test is not useful for all false detection cases. The examples on Figure 38shows a common situation when the light and shadow accidentally form a perfect bumper line that passes the above test without any difficulty. This forces us to find another feature distinguishable to the genuine train. By observation, there is always a smooth rectangular area without any texture right above the bumper on the genuine train, while most false detected backgrounds contain complex textures. Thus, the solution is searching around the bumper line for a pre-defined rectangular area with horizontal and vertical variance small enough to be regarded as texture-less. We can also examine the color of the detected area, since it can only be red or white for a genuine train. However, the color is always sensitive to environmental changes, and is therefore to be treated more carefully. Note that the smooth rectangular area can only be found when the train is close enough, otherwise the rectangular size would be too big for a far-distanced train, and that is why we only activate this Far Train Verification process when the train distance is around 20 meters.

Figure 38 Testing texture-less area above bumper for falsely detected background patches

4.1.3 Similarity Test

Similar to the above two features; we select a set of features helpful to distinguish positive and negative samples to form a similarity test. The similarity test includes the following features: The difference between the height estimated from vanishing line and the height detected (larger difference → less possible to be a train). The maximum contrast we obtained when finding the bumper. The ratio of valid pixel pairs from two lines above and below the bumper. The average difference (absolute value) of two lines above and below the bumper. The existence of smooth rectangular blocks (either red or white) above the bumper. The pixel intensity variance (smoothness measurement score) of that block. The detailed flow diagram of extracting those features is shown as below. Figure 39 shows the test to extract the difference of detected height and estimated height. The estimated height is determined empirically. This feature is useful because false detections often occurs due to similar patterns from buildings or sky bridges, where the estimated height would differ much from the detected height. Figure 40 shows test to find valid bumper and the edge-related features described in 4.1.2.1.Figure 41 shows the test to check the smooth and texture-less area as described in 4.1.2.2.

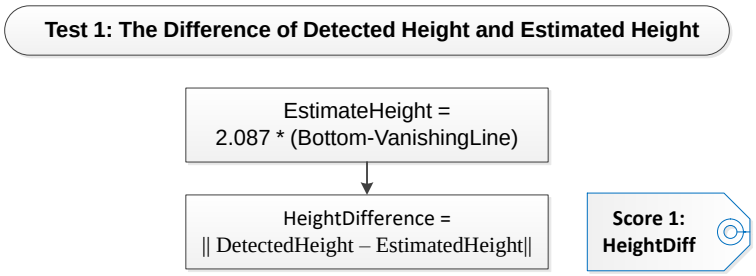


Figure 39 The difference of detected height and estimated height

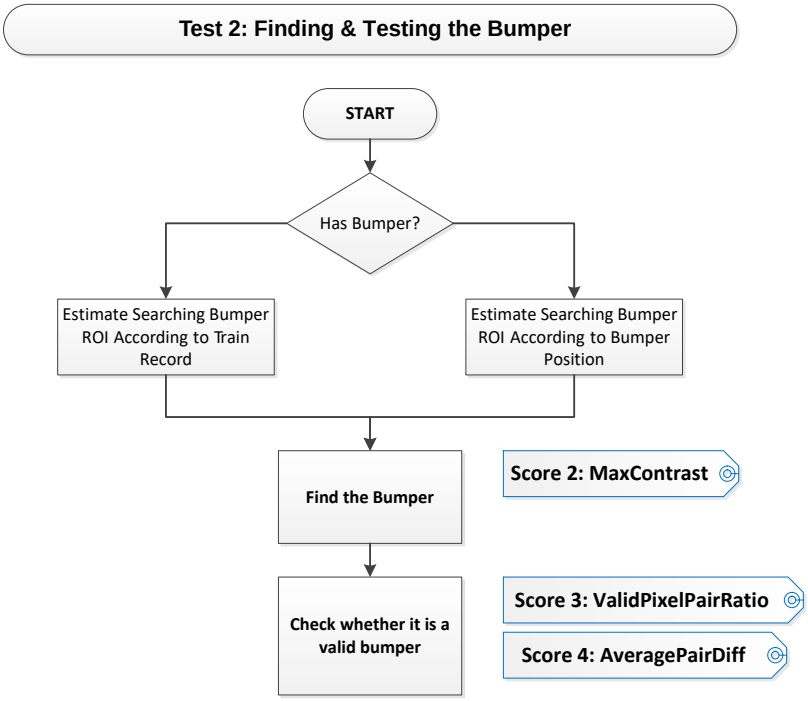


Figure 40 Finding and testing the bumper

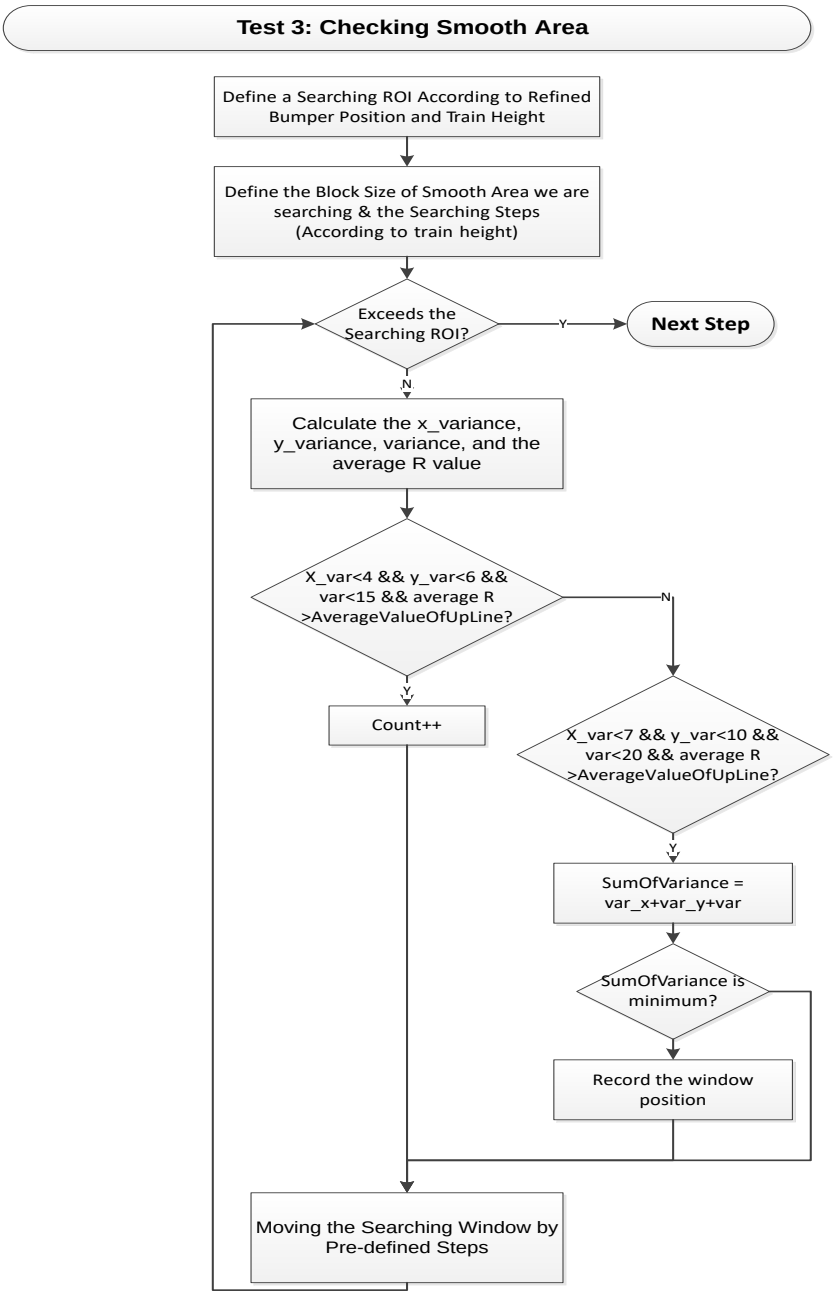


Figure 41 Checking smooth area

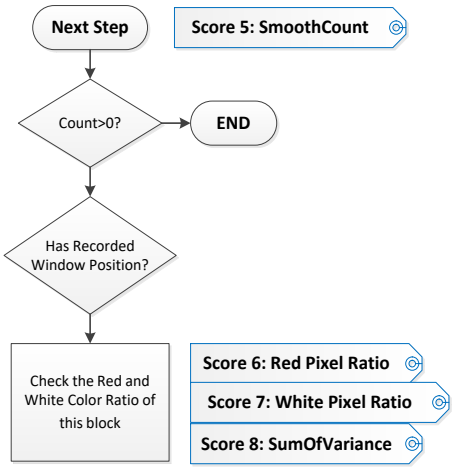


Figure 42 Extracting testing features for Far Train Verification

4.2 Close-range Red Train Detection Module

4.2.1 Introduction

The Close-range Red Train Detection Module we introduced in Chapter 3 could perform good detection of near red train in a comfortable environment. However, it fails at generalizing to a variety of environments where the color shifts greatly with different illumination conditions. As illustrated in the overview section, the red color on the train changes greatly from indoor to outdoor, from day to night, and even from day time with bright sunlight, to day time with dark shadow. Below shows a collection of red color areas in different situations. We can see the difficulty in defining the color red whatever color space is used. Besides the color shift, the average maximum contrast calculated for finding the bumper line and the blue line also varies much with different illuminations. For example, as shown in Figure 43 and Figure 44, the contrast value is small for indoor or in shadow areas, but large under sunlight. Thus setting the threshold high in one case will cause missed detection in another case, but decreasing the threshold might also lead to false detection.

Figure 43 Example of close-range red trains in different lighting conditions

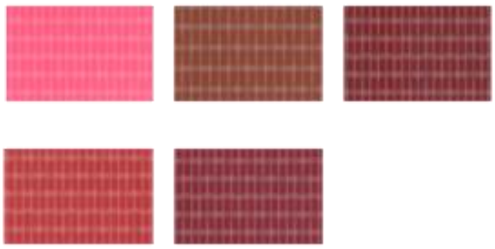


Figure 44 Example of red rectangular boxes in different lighting conditions

It is always the struggle of designing the testing criteria whether stringent or loose for a changing environment. One extreme leads to missed detections, while the other leads to false recognition. The solution is to use more features with less strict thresholds, record the score of each feature and distinguish the object from backgrounds through carefully designed decision rules involving all the features and the scores.

4.2.2 Feature Extraction

4.2.2.1 Texture-less Rectangle

The red train still has an important feature: it is Red. It is quite rare to find such a large texture-less red area in the background or in other types of trains. We evaluate the texture-less by calculating the variance of the intensities of all pixels inside the rectangular region R. *N* is the number of pixels in R. A simpler computation would be the x-variance and y-variance as defined in the following equation:

x-variance is the mean difference of horizontal neighboring pixels.

$$\Delta x = \left| I(x,y) - I(x+1,y) \right| \tag{13}$$

$$x_{\text{var}} = \frac{1}{N} \sum \Delta x, \quad for\ all\ (x,y) \in R \tag{14}$$

y-variance is the mean difference of vertical neighboring pixels.

$$\Delta y = \left| I(x,y) - I(x,y+1) \right| \tag{15}$$

$$y_{\text{var}} = \frac{1}{N} \sum \Delta y, \quad for\ all\ (x,y) \in R \tag{16}$$

Figure 45 shows an example of rectangular boxes and their x-variance and y-variance. It clearly shows the left box from background has much higher variance values than the right box from the smooth area of the red LRV.

Figure 45 Example of rectangular boxes and their x-variance and y-variance.

4.2.2.2 Color

Due to different illumination conditions under different environments, we define three kinds of red colors: The Normal Red, Dark Red, and Indoor Red. Pixels inside the testing box are classified according to their RGB and HSV values. For those with hue values between 340 and 360, if the R value is larger than 128, it is classified as Normal Red, otherwise Dark Red (with different confidence values relating to the R value). For those with hue values between 0 to 20, the classification is Indoor Red. Otherwise, the pixel is classified as non-red pixel. The final Red Indicator and Red Confidence for the whole box is calculated in the following steps. First, each pixel inside the rectangular box is scanned. Then each pixel is classified into “red pixels” and “non-red pixels” according to its RGB and HSV value. Each pixel is assigned with 2 values: red indicator that indicates the type of the red color, and red confidence that indicates how confident we are to classify the color to be “red”. Finally, the red indicator and confidence of the whole box can be

determined by summing up each pixel's information. Figure 47 shows examples of red indicator and red confidence in different conditions.

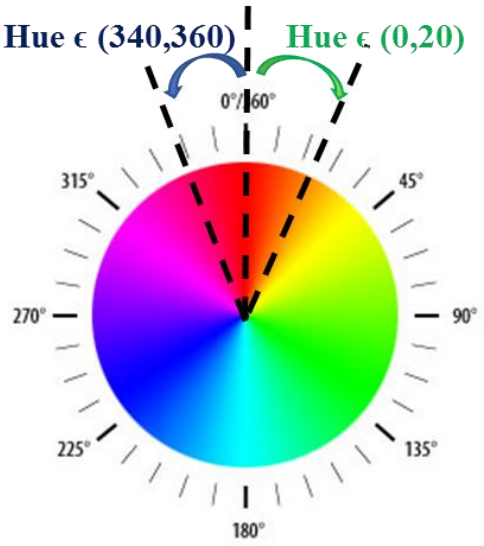


Figure 46 Range of "Red Color" in HSV color plane

Figure 47 Example of red indicators in different conditions

4.2.2.3 Edge and Line Contrast

We define the horizontal line contrast as average R channel intensity of green line minus the average R channel intensity of the blue line (multiple blue lines are involved while only one green line is involved). Then we search for a horizontal line giving the maximum contrast within the ROI, and this line is thus considered as the bumper line. Although there are two types of red trains, one with grey band between blue and red area while one

without, it is interesting that the blue line of both trains can be found using the same method. This is because the difference between grey-blue and the difference between red-blue are both large in the R channel, and this is also the reason we use R channel pixel values more frequent than pixel intensity. First, we use a linear equation to predict the position of the blue line based on the bumper position. Then we search for the blue line around the predicted region with the maximum contrast described in bumper detection.

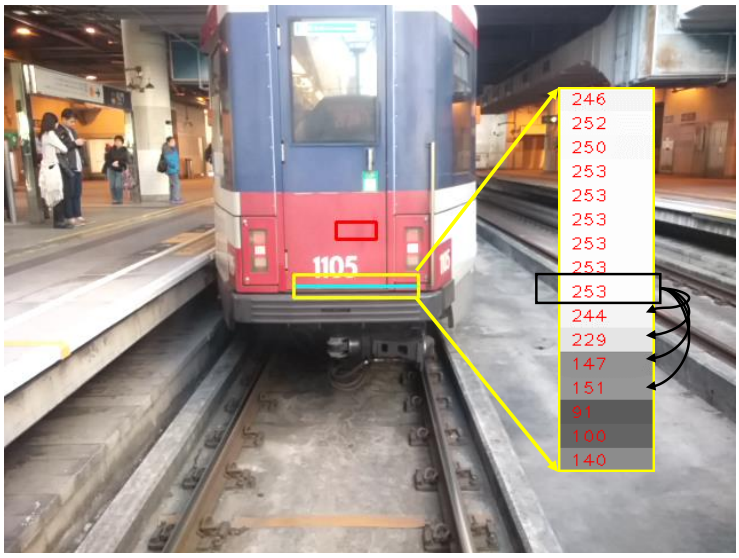


Figure 48 Example of bumper line

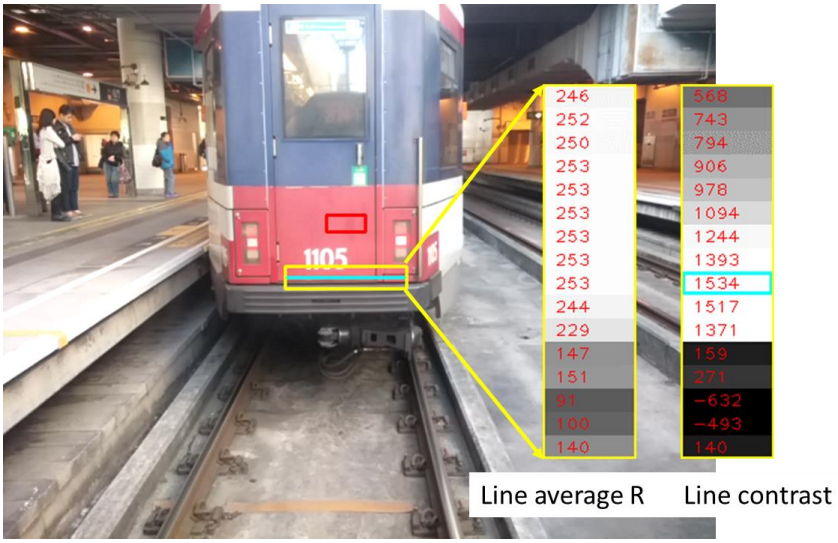


Figure 49 Example of line average and contrast

4.2.2.4 Other Features

Besides the above features, there are also many other features that could be useful to differentiate a red train from backgrounds, especially in indoor or night-time conditions. Figure 50 shows a complete set of features we use during our searching and classification process. Feature 4 is a typical pattern for night-time trains where the train contains very bright white-colored lines caused by the reflection of lights from consecutive trains. Feature 8 is the tail-light from vehicles at night-time. This is a more discriminative feature in dark illuminated conditions where the color and line contrast features might be severely influenced.

Feature 1: Red Pixel Ratio (the percentage of red pixels over the total pixel number in the square)

Feature 2: Red Indicator (based on the hue value of the red pixels, define three types of red: normal red, indoor red, and dark red)

Feature 3: Bumper Contrast (the maximum horizontal contrast when searching for the bumper)

Feature 4: Reflection (whether the train contains bright white-colored lines caused by the reflection of lights at night)

Feature 5: Blue line Contrast (the maximum horizontal contrast when searching for the blue line)

Feature 6: Red box Ratio (the percentage of red pixels in the red box)

Feature 7: Red box Position (the aspect ratio of the red box)

Feature 8: Train light (whether or not the train has a pair of train lights)

Figure 50 Extracting features for close-range red train

4.2.3 Searching Scheme

The flow diagram of searching and extracting the above features is shown in Figure 51. We divide the entire image or the entire RoI into multiple vertical stripes, and start to search for the smooth rectangular region from the bottom of each stripe. If a texture-less rectangular box is found, we then check the color of this box, and extract the red indicator and the red confidence features. If the ratio of red pixels passes certain threshold, then we continue to search for the bumper-line below this rectangular area, and similarly to search

the blue-line above this area. If the line contrasts of these two lines are higher than certain threshold, then we could locate the boundaries of the entire red box. The features we mentioned beforehand have already been extracted during the searching process. Thus, a similarity test will be conducted to verify whether the extracted features are good enough to describe a close-range red train. If this set of features could pass the similarity test, then the search would be terminated, and a valid close-range LRV is detected. Otherwise, we will continue searching other texture-less rectangular regions until a set of features that passes the similarity test is obtained. Finally, if none of the such set of features could be found, then it is safe to say that there is no close-range LRV in this particular frame.

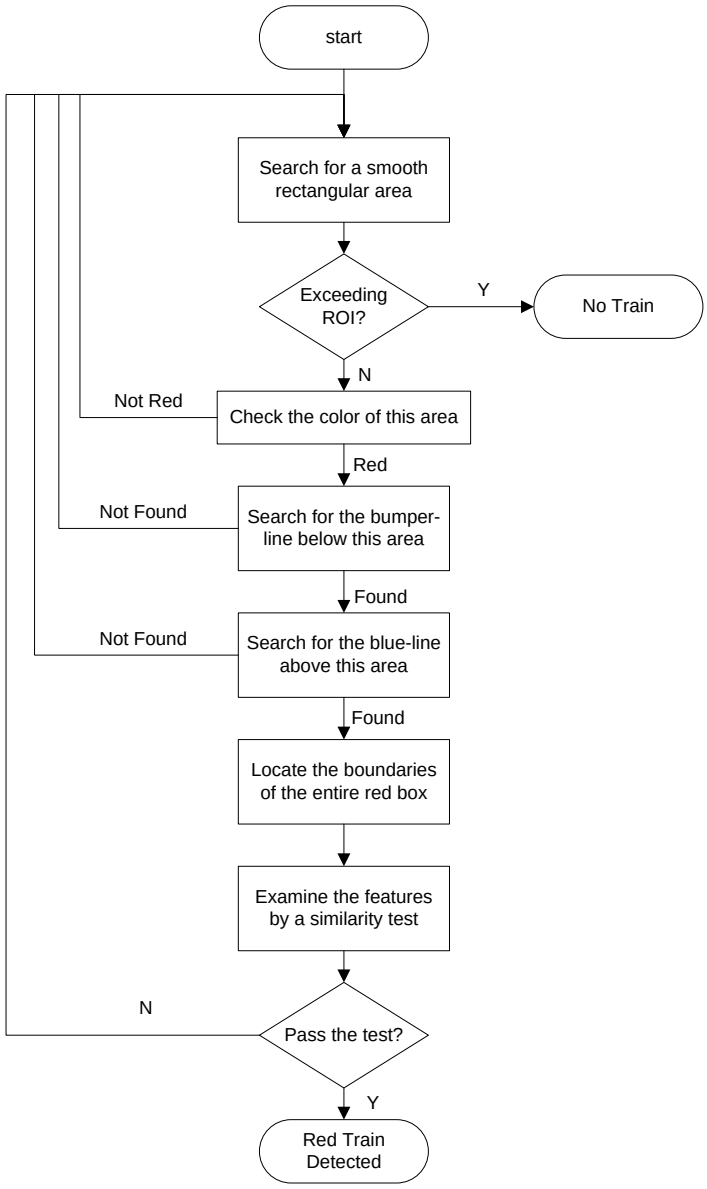


Figure 51 Flow diagram of the searching scheme

4.2.4 Similarity Test

It is always the struggle of designing the testing criteria whether stringent or loose for a changing environment. One extreme leads to missed detections, while the other leads to false recognition. The solution is to use more features with less strict thresholds, record the score of each feature and distinguish the object from backgrounds through carefully designed decision rules involving all the features and the scores. The decision-making process based on the extracted features is shown in Figure 52. We set this decision rules

based on the statistical distribution of objects and backgrounds for each feature. For example, according to our analysis, the feature “Red Box Location” and “Red Box Ratio” could separate the most portion of objects and backgrounds, thus we make it the primary feature to test in our decision process. Then for all samples that have passed this test, we figure the “Red Pixel Ratio” is the most critical feature separating the rest background samples, and thus this would be the feature to be tested in the next level. Similarly, we will test the red indicator, the bumper and blue-line contrast and other features consequently.

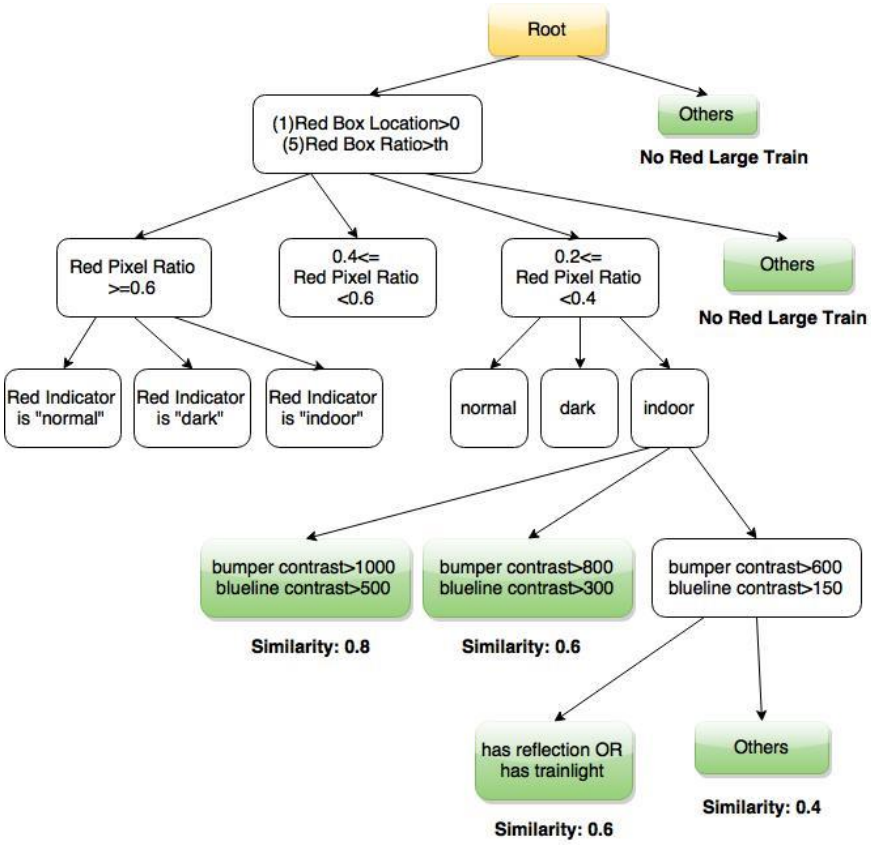


Figure 52 Decision rules of calculating the similarity scores for near red train

Chapter 5 Enhancing Performance via Machine Learning

5.1 Shadow Detection via Learning Approach

The conventional approach of the Shadow Detection Module introduced in Chapter 3 performs well for most testing frames. However, it misses detection for unseen patch that does not fit in the pre-designed rules. We can of course re-design the rules by either adding new cases or tuning thresholds, but that requires too much manual work. The weakness of the conventional approach lies in 1) not enough cases to represent all possible shadow patches, 2) manually tuned thresholds, 3) one mistake in the middle of the cascaded structure results in the total failure, and 4) quite complex features.

The classification of each stage is fast with satisfactory performance. The limitation of such rule-based classifiers is the inflexibility. A learning based method, however, could save the manual effort by either changing the training dataset or adding more penalty to miss detections during training.

5.1.1 Shadow-patch Detection using Decision Tree

5.1.2 Binary Test

Originally the features we choose to represent the shadow patch involve the histogram of oriented gradient, the color histogram, and some line contrast based feature. These are good and discriminative features, but a little too complex for a relatively simple task like classifying a shadow patch. We believe that a set of simple features with the help of learning based classifier can already achieve as good performance as the conventional method, if not better.

The feature we use is a simple binary test of the intensity difference between two pixels pair with some threshold value. Figure 53 shows an example of pixel pairs on an image patch. Assume that p_1 and p_2 are the pixel locations of the pair, the binary test function h is then defined by the following equation.

$$h = \begin{cases} 0, if\ I(p_1) < I(p_2) + \tau \\ 1, \quad otherwise \end{cases} \tag{17}$$

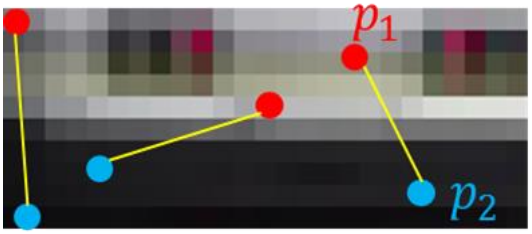


Figure 53 Example of pixel pairs on an image patch

5.1.3 Modified Decision Tree

The advantage of decision tree classifier is its fast speed, since only several binary tests are computed during the testing stage. However, it has some limitations. First, the prediction is made purely based on the posterior probability of the training data at leaf node, and the classifications at non-leaf nodes along the path have not been explicitly used. Second, at the training stage, the node stops growing as soon as the examples falling into its child node belong to the same class. However, due to the limited human power on labeling samples, not all training dataset is optimal. Some may suffer from insufficient number of examples, and some may not be able to cover all hard-to-classify examples. Overfitting could be a serious problem for such dataset. In order to increase the generalization power, we propose a modified decision tree.

Thus we propose a modified decision tree classifier where each non-leaf node is regarded as a weak classifier, and the overall prediction is the combination of the prediction from each weak classifier along the path. Let h_j represent the binary test for non-leaf node S_j at the j^{th} depth level. Let C_j ($C_j \in \{0,1\}$ for binary classification) represent the classification result from test h_j, and the overall prediction model C^* would be $C^*=Vote(C_1, C_2, ..., C_j, ...)$. We adopted the majority vote, as expressed in eqn.1, where c represents the class label, and $c \in \{0,1\}$ for binary classification, and $\delta(\cdot)=1$ if its argument is true and 0 otherwise.

$$C^* = \arg\max_c \sum_j \delta(C_j = c) \tag{18}$$

In addition to the prediction model, we also propose a confidence measurement for the classification result, by evaluating the classification accuracy of each binary test. The

classification accuracy of a binary test h_j is defined as the percentage of correct prediction evaluated on training data that fall into the parent node. For example, for a training subset $\{(x^{(k)},y^{(k)})|k=1,2,...M\}$ of M examples, where $x^{(k)}$ and $y^{(k)}$ are the image patch and class label of the k^{th} example, the classification accuracy α_j of the binary test h_j is defined in (19), where $C_j(x^{(k)})$ is the prediction result of testing patch $x^{(k)}$ by the binary test h_j.

$$\alpha_j = \frac{1}{M}\sum_{k=1}^{M}\delta(C_j(x^{(k)}) = y^{(k)}) \tag{19}$$

The overall prediction confidence α(C*) is the summation of the classification accuracy of each binary test that provides the correct prediction result in (20).

$$\alpha(C^*) = \sum_j(\alpha_j, if\,(C_j = C^*)) \tag{20}$$

The overall confidence is affected by 1) the majority prediction from all binary tests, and 2) the number of binary tests giving that prediction. The idea can be illustrated in Figure 54. The left path shows the classification of testing shadow patches with low confidence. Nodes 0 and 9 classify the patch as negative, while Nodes 4 and 20 predict them as positive. The right path shows the classification of a high-confidence patch. All nodes along the path classify the patch as positive.

Another modification is the design of the termination criteria. For a traditional decision tree, the node stops further splitting if all examples belong to the same class. However, due to the suboptimal selection of training data, for example, insufficient training samples, the tree growth might stop quite early, resulting in insufficient binary tests. This could result in an over-simplified prediction model. For such situation, we propose to continue learning the binary test for the node, until the maximum depth has been reached. Continuing learning may not provide further information gain, but can increase the number of useful binary tests, and thus greatly increase the discrimination power of the prediction model.

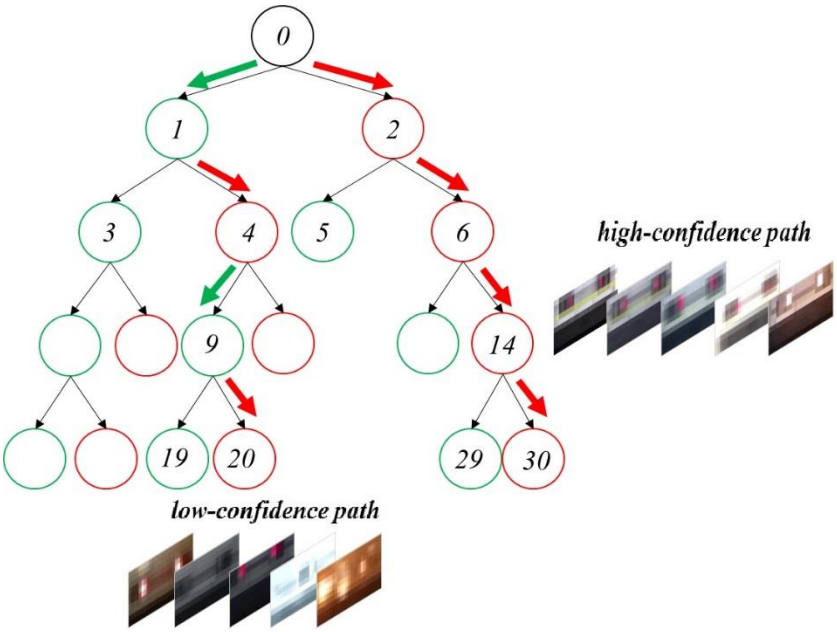


Figure 54 Classifying shadow patches using the proposed modified decision tree

5.1.4 Experimental Results

We have evaluated our algorithms on the Hong Kong light railway transportation system in both urban and suburban areas. The dataset is built from 175-hour-long videos captured by a camera mounted on railway trains running in various environmental conditions including bright sunny, cloudy, rainy outdoor conditions and ill-illuminated indoor conditions. From all frames, we extracted and annotated 263 frames that contain many conditions with ground truth bounding box of the train. The positive shadow patch and negative non-shadow patches can be then generated (Table 1).

We compare the learning-based modules with the conventional approach adopted in our early development stage, as described in 3.2.2.2, and we call it the 3-stage-cascade method for convenience. We have compared our proposed non-learning 3-stage-cascade method and learning based modified DT method with the classical SVM classifier implemented by OpenCV package. The feature used for SVM classifier is the HOG with 9 bins. The training dataset is the same for SVM classifier and for Modified DT, and the SVM hyper-parameters are automatically tuned to optimal by cross-validating the training dataset. We have also implemented the ORB-RANSAC based shadow detector using OpenCV

package. However, the detection rate is very low due to the severe changes of environmental conditions. Table 2 shows the performance comparison of the HOG+SVM, the proposed 3-stage-cascade classifier, and the proposed modified decision tree classifier. The precision, recall, and F_1-score are defined by $\frac{tp}{tp+fp}$, $\frac{tp}{tp+fn}$, $\frac{2*precision*recall}{precision+recall}$, where tp, tn, fp, and fn denote the true positive, true negative, false positive and false negative respectively. Although the classical HOG detector gives higher recall rate than our proposed non-learning method, the precision is quite low, meaning that it gives many false alarms. This makes sense since the HOG feature is only the feature used for the first stage in our 3-stage-cascaded detector, and many non-shadow patches cannot be differentiated merely by this feature. The relatively lower recall rate of the 3-stage-cascaded detector is due to the suboptimal design of threshold values due to insufficient observations. The proposed learning based algorithm further improves the performance with the highest recall rate, and also a quite satisfying precision rate. This means that the proposed method has the lowest miss rate, which is essential, while a satisfying false alarm rate, which is also acceptable since the shadow recognition is only the first stage in a vehicle detection system. The F_1-score also shows that the modified decision tree algorithm has the best performance among the compared methods.

Table 3 compares the timing of the three methods. Although the 3-stage-cascade classifier spends the longest time on classifying positive patch, it spends much less time on negative patches. This is due to the cascade architecture, and most negative patches can be rejected at the first two stages without further exploration. Note that there are far more negative patches than positive patches in each frame, so the average computation time for the 3-stage-cascade classifier is much less than the HOG+SVM method. The modified decision tree spends the least computation time among the three algorithms. In fact, it is 42 times faster than the SVM classifier with the HOG feature.

Table 1 Number of annotated frame and extracted positive and negative samples for training and testing datasets

Dataset	Number of Frames	Number of +ve samples	Number of -ve samples
Training	140	7700	10980
Test 1	60	3080	20226
Test 2	63	3135	23081

Table 2 Precision, recall, and F-score of 1) HOG and SVM classifier, 2) our proposed non-learning 3-stage-cascade classifier, and 3) proposed learning based modified decision tree classifier

Method	precision	recall	F_1 score
SVM	0.35	0.97	0.52
3-Stage-Cascade	0.88	0.61	0.72
Modified DT	0.64	0.98	0.78

Table 3 Computation time of 1) HOG and SVM classifier, 2) our proposed non-learning 3-stage-cascade classifier, and 3) proposed learning based modified decision tree classifier.

Method	Time (ms) per +ve patch	Time (ms) per -ve patch	Time (ms) per patch
HOG+SVM	1.240	1.195	1.200
3-Stage-Cascade	1.444	0.302	0.397
Modified DT	0.028	0.028	0.028

5.2 Enhancing Detection Module via Deep Learning

5.2.1 Review of system framework

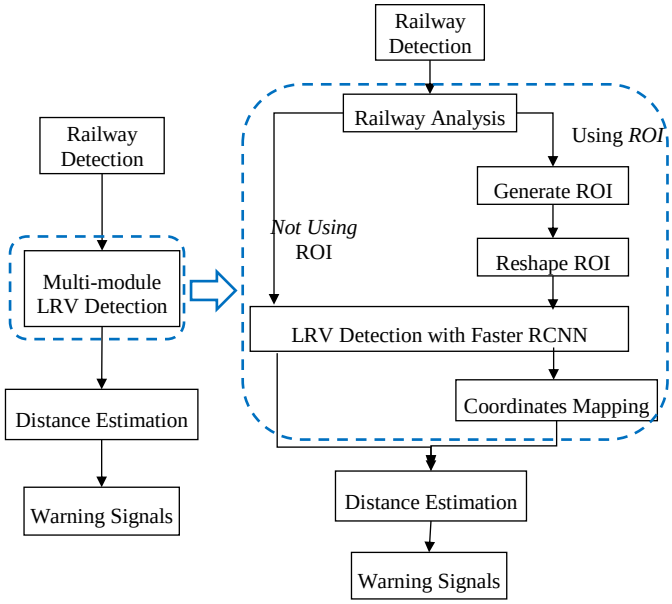


Figure 55 Multi-module based LRV detection system (left), LRV detection system with deep learning (right)

The objective of the light rail vehicle collision warning system is to provide warning signals once the system detects any frontal vehicle inside certain safety distance. A brief description of the system framework is shown in Figure 55. The system first performs railway detection to extract a pair of railways in the front to provide guidance for vehicle detection. Then, according to the detected railways, an LRV detection module with various sub-modules using multiple object detection algorithms is performed to detect if any light rail vehicle exists inside the region of interest. Once the vehicle is detected, a distance estimation module will be activated to estimate the real distance between the camera and the front vehicle, according to the geometric relationship of the camera calibration and the location of the detected vehicle.

The LRV detection module is the core to the LRV collision warning system. The challenge lies in dealing with severe variation in the vehicle appearance due to various illumination and weather conditions and the constant changing of distances between the front vehicle and the camera. In order to sustain a robust and highly reliable performance, we adopt a

multiple feature fusion approach in detection-level design, and a hierarchical modular structure approach in system-level design. The features we use include structural features such as edges and corners, and color information such as color histogram in HSV space, and also special patterns commonly possessed by vehicles, such as shadow [48]. We also design a hierarchical structure for efficiently combining sub-modules (such as Rough Search Module, Far-distance Detection, Night-time Detection, Tracking, etc.) and a buffering mechanism that allows the system shift between modules if the current detection is not convincing [49]. These designs allow our system to achieve extremely high robustness with zero missing detections through field tests. However, the system design is not the focus of this paper, and therefore will not be further explained in detail.

5.2.2 Faster RCNN with Adaptive ROI for LRV Detection

Faster RCNN is a state-of-the-art object detection algorithm. However, as mentioned before, Faster RCNN suffers from classifying small objects due to insufficient feature at ROI pooling layer. Therefore, we propose an adaptive ROI scheme for LRV detection using Faster RCNN.

The core of our proposed algorithm is a mode selection process based on railway information. For frames where the size of potential vehicle is large, ROI mode will not be used, and the entire frame will be forwarded as the input to the Faster RCNN detection module. For frames where the potential vehicle size is small, the ROI mode is selected, and a resized ROI will be forwarded as the input instead. Fig. 1 shows the framework of the system. The railway analysis module is responsible for checking the railway detection results and determine which mode to select. If ROI mode is selected, an ROI generation module followed by a reshaping module will be conducted to create the ROI and resize to a proper dimension. The Faster RCNN detection will then be conducted on this resized ROI, and the detection results will be projected to their original scale through the coordinate mapping process. If no ROI mode is selected, the Faster RCNN detection will be conducted on the entire frame.

5.2.2.1 Railway Analysis

After studying the railway detection output and the vehicle size, we found there an approximately linear relationship between the railway interval, i.e. the distance between

the ending pixels of the left and right rail, and the vehicle height. We formulate this relationship in (21), where h_{lrv} denotes vehicle height, d_{rail} denotes the railway interval, β and β_0 denote the parameters to be learned by linear regression. A similar linear relationship is also valid between the railway interval and the vehicle width, but here we choose height, because in our previous multi-module system, the detected height is more accurate and thus more reliable for learning. We took over 20000 training examples extracted from one video sequence and used the least square estimation defined in (22) to solve the linear regression, where $\boldsymbol{\beta}$ is the vector of the learned parameter β and β_0, $\mathbf{X}$ and $\mathbf{y}$ is respectively the railway intervals and vehicle heights of the training samples in matrix form.

$$h_{lrv} = \beta \cdot d_{rail} + \beta_0 \tag{21}$$

$$\boldsymbol{\beta} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} \tag{22}$$

The vehicle height can be estimated using (21) for each frame with a valid railway detection result. The detection ROI can thus be defined according to the estimated vehicle height and the railway location. The height of the ROI is the vehicle height h_{lrv} plus some padding p, and the width is calculated so that the ROI has the same aspect ratio with the original frame. The center of the ROI is aligned with the center of the railway, i.e. the center of the ending pixels of both rail.

Note that there is no need to resize every ROI but only the ones that are likely to have smaller-sized vehicles, we added a mode selection process based on the railway interval d_{rail}. Empirically we found the Faster RCNN not good at detecting objects smaller than 50 pixels, so with certain tolerance we set the height thrseshold h_{th} to be 100, and then calculate the railway interval threshold d_{th} according to (21). For the frame where $d_{rail} < d_{th}$, the *Using ROI Mode* would be selected, and an ROI as defined above will be generated and resized into the size of the original frame. This resized ROI will then be sent to the Faster RCNN detection module. For the frames where no railway has been detected or $d_{rail} \geq d_{th}$, the *Not Using ROI Mode* will be selected and the whole frame will be sent as the input of the Faster RCNN detection module.

To test the validity of our proposed Adaptive ROI scheme, we studied on a short video sequence that contains LRVs from remote distances, i.e. larger than or equal to 50 meters.

The LRV height at this interval ranges from 14 to 43 pixels, and averaged at 25.7 pixels. These are the sizes that Faster RCNN normally could not perform well, and thus we expect our adaptive ROI scheme to conduct enlargement on these cases. We counted the number of cases where the LRVs can be resized to an ideal size, and the number of cases where the LRVs can be resized to a detectable size. We defined the ideal size as 112 pixels, because this is the size where zero padding to the feature map is not required for the ROI pooling process in Faster RCNN, if structures described in [46] is adopted. The destination size for ROI pooling is 7×7, meaning that any feature map smaller than 7 will be padded with zeros, and since there are 4 pooling layers with stride 2 padding 2 during the shared convolutional layers, the size of the receptive field from the input image would be 7×16, i.e. 112 pixels. We also defined the detectable size as 53 pixels, because empirically we found it the minimum size that Faster RCNN can still detect. Table I shows the number of frames where the small LRVs are resized to at least the ideal size, and the number of frames where LRVs are resized to at least the detectable size.

Table 4 Resizing Effects on Very Small LRVs Using Adaptive ROI Scheme

Total Frames	Ideal-sized Frames	Detectable-sized Frames
4246	2309	4220

5.2.2.2 LRV Detection with Faster RCNN

The Faster RCNN structure described in [46] takes 13 convolutional layers and 4 pooling layers as shared convolutional layers based on the VGG-16 net [50]. An RPN network for region proposal generation and a Fast RCNN network for classification are built on top of the last shared layer. For the Fast RCNN network, we suggest to use the default feature size, i.e. 7×7, in the ROI-Pooling stage, since we find it extremely useful to keep the dimensions for the following fully connected layers in order to take advantage of the weights pre-trained from ImageNet dataset. For the RPN network, we adopt the same anchor sizes, i.e. 128, 256, and 512 as in the original paper, instead of reducing the sizes to adapt for smaller objects, because our targets also contain very large objects, and we believe that our proposed algorithm can already solve the small vehicle detection problem without modifying the anchor sizes.

Transfer learning is important for training small dataset with deep models. Using pre-trained model from larger dataset as initialization could accelerate the convergence of fine tuning. The Faster RCNN is trained via a 2-stage alternating optimization approach. In Stage 1, we use the ImageNet pre-trained model to initialize the RPN and Fast RCNN, and our LRV dataset for fine-tuning, including all shared layers. In Stage 2, we use the Fast RCNN model trained in Stage 1 as initialization and fine-tune all the unshared layers.

The selection of training dataset is crucial to the performance of the detection. Preliminarily, we built a small training dataset consisting of 34 LRV images of different size, color, and illumination condition. With the help of transfer learning on pre-trained models from a much larger dataset (ImageNet), our model converged quickly after 5000 iterations in the second stage of the alternating optimization process. Although the model performed well on most close-ranged, well-illuminated LRV detections, it failed at detecting more challenging situations such as turning, night-time, and remote-ranged LRVs. Later we expanded our training dataset into 110 examples including more difficult cases. However, we found the performance still unsatisfactory, especially for remote-distanced LRVs, even when we already used the A-ROI scheme. The critical problem is that the previous trainings failed at modeling the pattern on the resized ROI, which is the actual input to the network at testing stage, if the Using ROI Mode is activated. Therefore, we added more samples of the resized ROI created using the algorithms introduced in 5.2.2.1to form our final training dataset. Table 5 shows the number of successfully detected frames tested on a small video sequence of 327 frames, where an LRV at the exit of a tunnel is approached from remote to close distance.

Table 5 Number of successfully detected frames using different training datasets

Training Dataset	34 Examples	110 Examples	265 Examples
Detected Frames	23	79	323

5.2.2.3 Coordinate Mapping

The Faster RCNN detection module will output the probability of having or not having the vehicle as well as the bounding box coordinates of the vehicle if detected. However, these coordinates are the coordinates relative to the resized ROI, instead of the original frame. Therefore, a coordinate mapping mechanism is required to map back the relative coordinates regarding to ROI to absolute coordinates regarding to the whole frame.

Figure 56illustrates the relationship between the absolute coordinates (left) and the relative coordinates (right), where the height and width of the original frame is denoted as w_F and h_F, the height and width of the ROI is denoted as w_R and h_R, and the coordinate of the top-left corner of the ROI is denoted by (x_R, y_R). Assume there is an object inside the ROI with its top-left corner denoted as (x_o, y_o). At the *Reshaping ROI Step*, the ROI is resized into the frame size, and the relative coordinate of its top-left corner (x_R', y_R') would become $(0, 0)$. The coordinate of the object top-left corner (x_o', y_o') relative to the resized ROI can then be calculated as in(23).

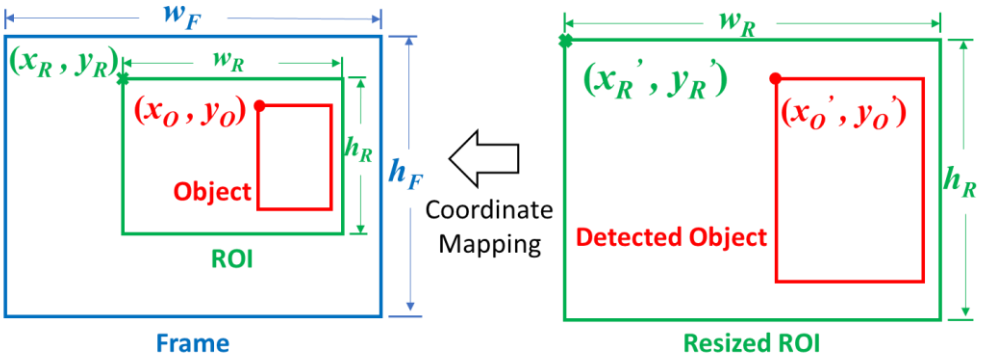


Figure 56 Coordinate mapping for resized ROI (right) back to original frame (left)

$$x_o' = \frac{x_o - x_R}{w_R} \times w_F \quad , \quad y_o' = \frac{y_o - y_R}{h_R} \times h_F \tag{23}$$

After the Faster RCNN detection, the *Coordinate Mapping Step* will map the relative coordinates (x_o', y_o') back to the absolute coordinate (x_o, y_o), which is just the reverse computation of the forward mapping, as described in (24).

$$x_o = \frac{w_R}{w_F} \times x_o' + x_R \quad , \quad y_o = \frac{h_R}{h_F} \times y_o' + y_R \tag{24}$$

Similar to the mode selection of the *Railway Analysis Module* described in5.2.2.1, the *Coordinate Mapping* is only applied to frames that selects *Using ROI Mode*. The output is the final bounding-box if the vehicle is detected, and will be passed to the Distance Estimation Module as in our conventional LRV system.

5.2.3 Experiments

We have evaluated the detection performances on video sequences collected from a monocular camera mounted on the front window of light rail vehicles running in Hong Kong. We labeled 55,000 frames out of these 175-hour-long sequences as *containing LRVs (positive)* or *only backgrounds (negative)* to form our testing dataset. In order to test the effectiveness of detection modules on LRVs from different distance levels, we further divided our dataset into three categories according to the front vehicle distance. The LRV is relatively large and its size normally ranges from 70 to 480 pixels when the distance is within 20 meters. It is critical for the system to perform extremely accurate detection and provide timely warning signals in such close range to prevent potential collision. The requirement is less strict when the distance of the front vehicle ranges from 20 to 60 meters, since the distance is larger than the safety distance and the ill-detections are likely to be saved as the LRV approaches closer. However, it is still desirable that highly accurate detection is achieved from a far distance, which will not only facilitate the closer-range detection/tracking, but could also accumulate a long-time detection record and thus increase the overall detection confidence from a system design view. And the rest are the LRVs at a distance farther than 60 meters, where the vehicle size is smaller than 20 pixels and becomes very difficult to recognize even with human eyes.

We tested the system performances for three different detection modules. The first is the non-learning approached detection module with hand-crafted feature fusion techniques as described before, specifically, the module includes the Shadow Detection Module, the Far Train Detection Module, and the Close-range Train Detection Module described in 3.2.2.2, 3.2.2.3 and 3.2.2.4 respectively. The second is the Faster RCNN detection module that directly applies the trained Faster RCNN network on the entire frame to perform detection.

The third module used our proposed Faster RCNN with Adaptive ROI algorithm (FRCNN with A-ROI). The Faster RCNN network used in both the second and the third module was trained with 265 training samples extracted from videos exclusive to our testing videos.

Table 6, 7, and 8 show the performance comparison of the three detection modules in each distance range. The performance is measured by accuracy, precision, recall and F_1-score, defined as $\frac{tp+fp}{tp+fp+tn+fn}$, $\frac{tp}{tp+fp}$, $\frac{tp}{tp+fn}$, $\frac{2*precision*recall}{precision+recall}$, where *tp*, *tn*, *fp*, *fn* denote the number of true positive, true negative, false positive and false negative. Recall rate reflects the situation of missing detections, and higher recall rate indicates fewer missing cases, which is crucial for vehicle detection systems, especially at close distance. Precision rate reflects situation of false alarms, and it is also important for collision warning system to have high precision rate since it should avoid annoying the driver with falsely detected cases as much as possible.

From the result we can see that all three detection modules have achieved very high recall and precision rate when the distance is smaller than 20 meters. Note that we only evaluated the LRV detection modules frame by frame in this experiment, and some system-level techniques such as tracking and buffering mechanisms were disabled. Thus, it is understandable that the recall rate is not 100% in this evaluation. In fact, with these mechanisms enabled, the system performance of the non-learning approach can already achieve zero-missing-detection requirement. Now if we look at the results in 20-60m distance range, it is clear that the Faster RCNN detection module has the lowest recall rate. This is due to its inability to detect very small objects, and most LRVs at this distance have sizes ranging from 20 to 70 pixels. Our proposed Faster RCNN with adaptive ROI (FRCNN A-ROI) algorithm introduces a railway-based ROI selection mechanism that greatly enlarge the object to be detected, especially for very small vehicles. This is why our proposed detection module can achieve a 48% (87.7-38.9)% increase in the recall rate compared with the original Faster RCNN detection module. This result shows the effectiveness of our proposed algorithm in detecting distant vehicles. Our proposed detection module could still detect a small portion of LRVs farther than 60 meters, while the other two conventional detection modules can hardly detect any at such far distance.

Although detection beyond 60 meters is not our major focus, still it shows the capability of our proposed detection module as well as the entire system. Figure 57 shows the histogram of recall rate on vehicle distance. It is clear to see that the recall rate for Faster RCNN without A-ROI drops severely as the distance becomes larger, while the recall rate for Faster RCNN with A-ROI remains a high level until extreme distances.

Table 6 Performance of different detection modules at distance from 0 to 20 m

Detection Modules	Distance : 0 - 20 (m)			
	Accuracy	Precision	Recall	F_1-Score
Multi-Feature Fusion	0.953	0.995	0.922	0.957
Faster RCNN	0.966	0.974	0.967	0.97
FRCNN with A-ROI	0.981	0.998	0.969	0.983

Table 7 Performance of different detection modules at distance from 21 to 60 m

Detection Modules	Distance : 21 - 60 (m)			
	Accuracy	Precision	Recall	F_1-Score
Multi-Feature Fusion	0.749	0.821	0.725	0.77
Faster RCNN	0.632	0.941	0.389	0.551
FRCNN with A-ROI	0.913	0.97	0.877	0.921

Table 8 Performance of different detection modules at distance over 60 m

Detection Modules	Distance : over 60 (m)			
	Accuracy	Precision	Recall	F_1-Score
Multi-Feature Fusion	0.554	0.016	0.008	0.011
Faster RCNN	0.686	0.183	0.018	0.033
FRCNN with A-ROI	0.761	0.757	0.281	0.41

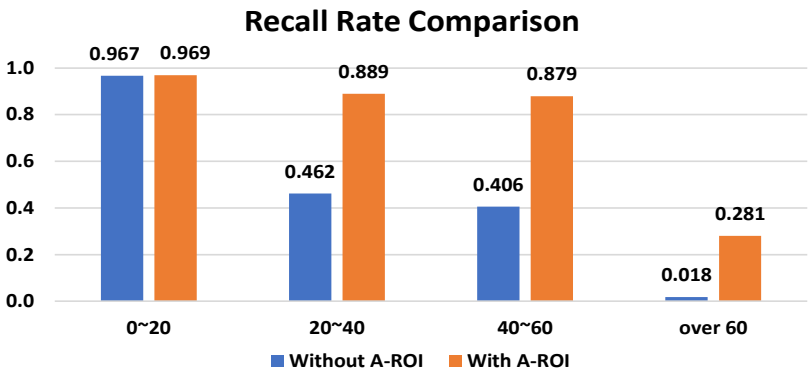


Figure 57 Recall rate comparison for Faster RCNN with and without A-ROI at different distance levels

Figure 58 shows some example detections using the proposed FRCNN A-ROI algorithm. Figures on the right shows the resized ROI estimated from railway. The resized ROIs provide larger and thus more discriminative features for Faster RCNN to process in the classification stage. Thus the system is able to detect distant vehicles even in ill-illuminated environments, in which cases the original Faster RCNN without ROI fails to detect.

For computational cost, the Faster RCNN based methods take an average of 0.065 seconds per frame, while the conventional method takes an average of 0.005 seconds, which can explain the reason of not using deep learning methods at the start of our development. It can be further explored as a future topic to reduce the computation of deep learning methods in this project.

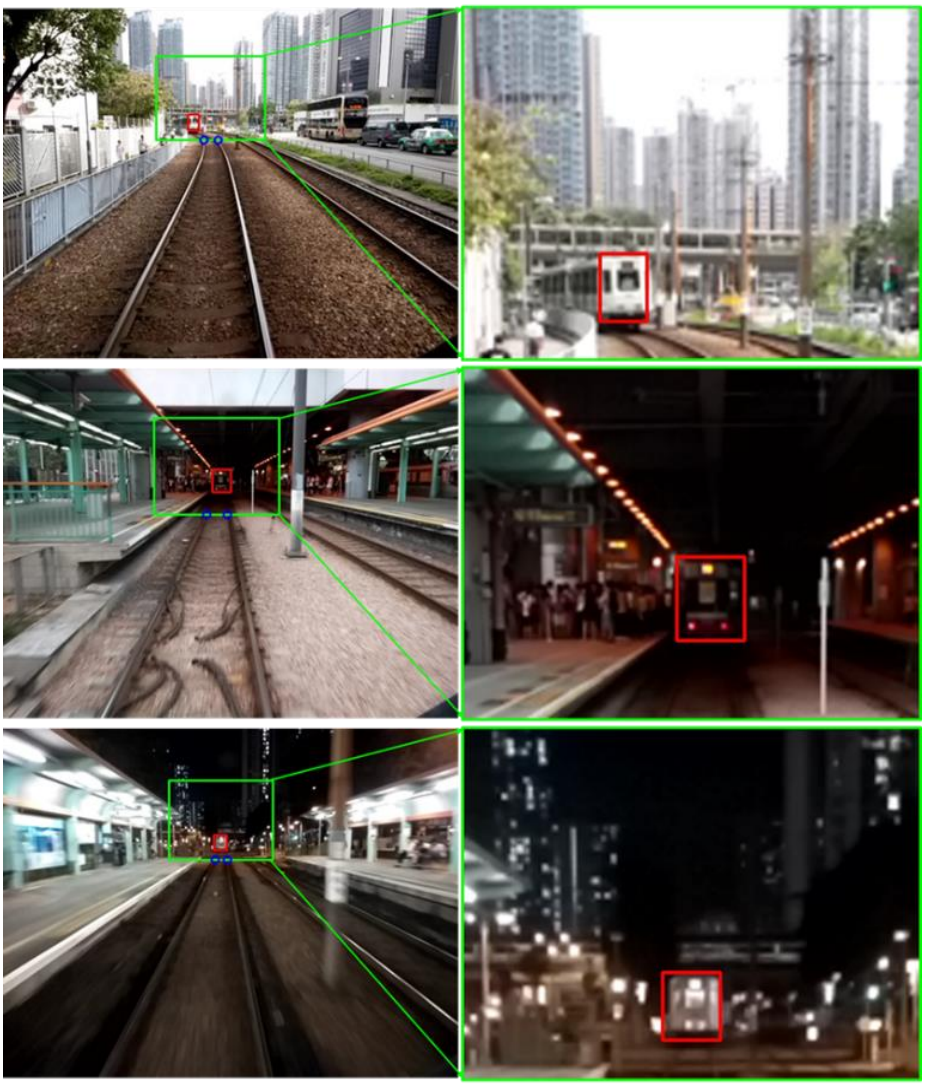


Figure 58 Examples of the Faster RCNN LRV Detection result. (Right: LRV detected on resized ROI; Left: final detection result after coordinate mapping; Green box: estimated ROI; Blue dots: detected railway ends)

Chapter 6 **Conclusion**

We have developed a highly reliable Light Rail Vehicle detection system based on hierarchical structural design. We design a hierarchical multi-module structure with each module adopting various orthogonal or semi-orthogonal features to detect vehicles under certain circumstances. We also improve the detection accuracy by designing a verification module that checks the low-confident detections by totally different sets of features. These studies specifically show how multiple orthogonal features could increase discriminability as well as maintain high robustness. The system achieves high performance with no missing detections and few false alarms in our field tests.

We propose a shadow detection step that aims at recognizing the shadow part of the train in various environments (including very tough cases) to accelerate the detection process. We propose two shadow recognition approaches for railway trains. In our first approach, we propose a prioritized feature extraction scheme that examines multiple features such as HOG and Color Histogram hierarchically to achieve high robustness as well as preserve the fast detecting speed. Experiments show satisfying results. Subsequently we propose a second approach using machine learning that automatically learns the features and decisions via a modified decision tree classifier with a novel confidence measuring scheme. Experiments show further improvements in both accuracy and execution time.

We also proposed a faster RCNN based detection module for our LRV collision warning system. We then further improve the detection performance through a novel Adaptive ROI selection scheme based on railway information. Experimental results show high reliability in the overall detection accuracy and proves the ability of detecting small vehicles even from a very far distance.

There are a few suggestions for future study. Our system is developed when the training samples are scarce. However, as obtaining data becomes more and more accessible, it is beneficial to take advantage of the big data environment. An online learning module where the system could automatically learn and adjust its detection model from new incoming data is suggested. Another suggestion could be a deeper analysis on interpreting and applying CNN features. Although we have shown the benefit of applying deep learning

models for detection in our research, yet it requires high-performance hardware support, which is often not available in inexpensive monocular cameras. Thus it is attractive if the CNN features or the filter design in deep learning models could also be used in other machine learning algorithms to enhance the performance.

Chapter 7 **References**

[1] S. Sivaraman and M. M. Trivedi, “Looking at Vehicles on the Road: A Survey of Vision-Based Vehicle Detection, Tracking, and Behavior Analysis,” *IEEE Transactions on Intelligent Transportation Systems,* vol. 14, no. 4, pp. 1773-1795, 2013.

[2] X. Mao, D. Inoue, S. Kato and M. Kagami, “Amplitude-Modulated Laser Radar for Range and Speed Measurement in Car Applications,” *IEEE Transactions on Intelligent Transportation Systems,* vol. 13, no. 1, pp. 408-413, 2012.

[3] S. Tokoro, K. Kuroda, A. Kawakubo, K. Fujita and H. Fujinami, “Electronically scanned millimeter-wave radar for pre-crash safety and adaptive cruise control system,” in *Proceedings of IEEE IV2003 Intelligent Vehicles Symposium. Proceedings (Cat. No.03TH8683),* pp. 304-309, 2003.

[4] S. Sato, M. Hashimoto, M. Takita, K. Takagi and T. Ogawa, “Multilayer lidar-based pedestrian tracking in urban environments,” in *Proceedings of 2010 IEEE Intelligent Vehicles Symposium,* pp. 849-854, 2010.

[5] S. Sivaraman and M. M. Trivedi, “Integrated Lane and Vehicle Detection, Localization, and Tracking: A Synergistic Approach,” *IEEE Transactions on Intelligent Transportation Systems,* vol. 14, no. 2, pp. 906-917, 2013.

[6] M. Mahlisch, R. Schweiger, W. Ritter and K. Dietmayer, “Sensorfusion Using Spatio-Temporal Aligned Video and Lidar for Improved Vehicle Detection,” in *Proceedings of 2006 IEEE Intelligent Vehicles Symposium,* pp. 424-429, 2006.

[7] C. Rabe, U. Franke and S. Gehrig, “Fast detection of moving objects in complex scenarios,” in *Proceedings of 2007 IEEE Intelligent Vehicles Symposium,* pp. 398-403, 2007.

[8] Sun Zehang, G. Bebis and R. Miller, “On-road vehicle detection: a review,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 28, no. 5, pp. 694-711, 2006.

[9] Christos Tzomakas and Werner von Seelen, ″Vehicle Detection in Traffic Scenes Using Shadows," Institut fur Neuroinformatik, Ruhr-Universitat, 1998, Germany.

[10] Minkyu Cheon, Wonju Lee, Changyong Yoon and Mignon Park, “Vision-Based Vehicle Detection System With Consideration of the Detecting Location,” *IEEE Transactions on Intelligent Transportation Systems,* vol. 13, no. 3, pp. 1243-1252, 2012.

[11] B. Tian, Y. Li, B. Li and D. Wen, “Rear-View Vehicle Detection and Tracking by Combining Multiple Parts for Complex Urban Surveillance,” *IEEE Transactions on Intelligent Transportation Systems,* vol. 15, no. 2, pp. 597-606, 2014.

[12] B. Aytekin and E. Altuğ, “Increasing driving safety with a multiple vehicle detection and tracking system using ongoing vehicle shadow information,” in *Proceedings of 2010 IEEE International Conference on Systems, Man and Cybernetics*, pp. 3650-3656, 2010.

[13] C. Hoffmann, “Fusing multiple 2D visual features for vehicle detection,” in *Proceedings of 2006 IEEE Intelligent Vehicles Symposium,* pp. 406-411, 2006.

[14] C. H. Hilario, J. M. Collado, J. M. Armingol and A. de la Escalera, “Pyramidal image analysis for vehicle detection,” in *Proceedings of IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, pp. 88-93, 2005.

[15] J. Arrospide, L. Salgado, M. Nieto and F. Jaureguizar, “On-board robust vehicle detection and tracking using adaptive quality evaluation,” in *Proceedings of 2008 15th IEEE International Conference on Image Processing*, pp. 2008-2011, 2008.

[16] Corvin Idler, Roland Schweiger, Dietrich Paulus, Mirko Mdhlisch and Werner Ritter, “Realtime Vision Based Multi-Target-Tracking with Particle Filters in Automotive Applications,” in *Proceedings of 2006 IEEE Intelligent Vehicles Symposium*, pp. 188-193, 2006.

[17] Ronan O'Malley, Edward Jones and Martin Glavin, “Rear-Lamp Vehicle Detection and Tracking in Low-Exposure Color Video for Night Conditions,” *IEEE Transactions on Intelligent Transportation Systems,* vol. 11, no. 2, pp. 453-462, 2010.

[18] L. W. Tsai, J. W. Hsieh and K. C. Fan, “Vehicle Detection Using Normalized Color and Edge Map,” *IEEE Transactions on Image Processing,* vol. 16, no. 3, pp. 850-864, 2007.

[19] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Proceedings of 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 886-893 vol. 1, 2005.

[20] Zhu Qiang, Yeh Mei-Chen, Cheng Kwang-Ting and S. Avidan, “Fast Human Detection Using a Cascade of Histograms of Oriented Gradients,” in *Proceedings of 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, pp. 1491-1498, 2006.

[21] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, pp. I-511-I-518 vol.1, 2001.

[22] Wei Liu, XueZhi Wen, Bobo Duan, Huai Yuan and Nan Wang, “Rear Vehicle Detection and Tracking for Lane Change Assist,” in *Proceedings of 2007 IEEE Intelligent Vehicles Symposium*, pp. 252-257, 2007.

[23] Sayanan Sivaraman and Mohan M Trivedi, “Active learning for on-road vehicle detection: A comparative study,” *Machine vision and applications*, vol. 25, no. 3, pp. 599-611, 2014.

[24] Sun Zehang, G. Bebis and R. Miller, “Monocular precrash vehicle detection: features and classifiers,” *IEEE Transactions on Image Processing*, vol. 15, no. 7, pp. 2019-2034, 2006.

[25] J. Cui, F. Liu, Z. Li and Z. Jia, “Vehicle localisation using a single camera,” in *Proceedings of 2010 IEEE Intelligent Vehicles Symposium*, pp. 871-876, 2010.

[26] G. Y. Song, K. Y. Lee and J. W. Lee, “Vehicle detection by edge-based candidate generation and appearance-based classification,” in *Proceedings of 2008 IEEE Intelligent Vehicles Symposium*, pp. 428-433, 2008.

[27] A. Haselhoff and A. Kummert, “A vehicle detection system based on Haar and Triangle features,” in *Proceedings of 2009 IEEE Intelligent Vehicles Symposium*, pp. 261-266, 2009.

[28] S. Sivaraman and M. M. Trivedi, “A General Active-Learning Framework for On-Road Vehicle Recognition and Tracking,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 2, pp. 267-276, 2010.

[29] David G Lowe, “Distinctive image features from scale-invariant keypoints,” *International journal of computer vision,* vol. 60, no. 2, pp. 91-110, 2004.

[30] X. Zhang, N. Zheng, Y. He and F. Wang, “Vehicle detection using an extended Hidden Random Field model,” in *Proceedings of 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pp. 1555-1559, 2011.

[31] Corinna Cortes and Vladimir Vapnik, “Support-vector networks,” *Machine learning,* vol. 20, no. 3, pp. 273-297, 1995.

[32] Wei Liu, XueZhi Wen, Bobo Duan, Huai Yuan and Nan Wang, “Rear vehicle detection and tracking for lane change assist,” in *Proceedings of Intelligent Vehicles Symposium, 2007 IEEE*, pp. 252-257, 2007.

[33] Zehang Sun, George Bebis and Ronald Miller, “Monocular precrash vehicle detection: features and classifiers,” *IEEE transactions on image processing,* vol. 15, no. 7, pp. 2019-2034, 2006.

[34] T. Liu, N. Zheng, L. Zhao and H. Cheng, “Learning based symmetric features selection for vehicle detection,” in *Proceedings of IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, pp. 124-129, 2005.

[35] A. Khammari, F. Nashashibi, Y. Abramson and C. Laurgeau, “Vehicle detection combining gradient analysis and AdaBoost classification,” in *Proceedings of Proceedings. 2005 IEEE Intelligent Transportation Systems, 2005.*, pp. 66-71, 2005.

[36] I. Kallenbach, R. Schweiger, G. Palm and O. Lohlein, “Multi-class Object Detection in Vision Systems Using a Hierarchy of Cascaded Classifiers,” in *Proceedings of 2006 IEEE Intelligent Vehicles Symposium*, pp. 383-387, 2006.

[37] T. T. Son and S. Mita, “Car detection using multi-feature selection for varying poses,” in *Proceedings of 2009 IEEE Intelligent Vehicles Symposium*, pp. 507-512, 2009.

[38] D. Acunzo, Y. Zhu, B. Xie and G. Baratoff, “Context-Adaptive Approach for Vehicle Detection Under Varying Lighting Conditions,” in *Proceedings of 2007 IEEE Intelligent Transportation Systems Conference*, pp. 654-660, 2007.

[39] S. Sivaraman and M. M. Trivedi, “Real-time vehicle detection using parts at intersections,” in *Proceedings of 2012 15th International IEEE Conference on Intelligent Transportation Systems*, pp. 1519-1524, 2012.

[40] J. J. Huang and W. C. Siu, “Learning Hierarchical Decision Trees for Single-Image Super-Resolution,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 5, pp. 937-950, 2017.

[41] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton, “Imagenet classification with deep convolutional neural networks,” in *Proceedings of Advances in neural information processing systems*, pp. 1097-1105, 2012.

[42] Spyros Gidaris and Nikos Komodakis, “Object detection via a multi-region & semantic segmentation-aware CNN model,” in *Proceedings of ICCV*, 2015.

[43] Ross Girshick, “Fast R-CNN,” in *Proceedings of 2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1440-1448, 2015.

[44] Ross Girshick, Jeff Donahue, Trevor Darrell and Jitendra Malik, “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation,” in *Proceedings of 2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580-587, 2014.

[45] Koen EA Van de Sande, Jasper RR Uijlings, Theo Gevers and Arnold WM Smeulders, “Segmentation as selective search for object recognition,” in *Proceedings of 2011 International Conference on Computer Vision*, pp. 1879-1886, 2011.

[46] Shaoqing Ren, Kaiming He, Ross Girshick and Jian Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149, 2017.

[47] Liliang Zhang, Liang Lin, Xiaodan Liang and Kaiming He, “Is Faster R-CNN Doing Well for Pedestrian Detection?,” in *Proceedings of CVPR*, 2016.

[48] Xue-Fei Yang and Wan-Chi Siu, “Vehicle detection under tough conditions using prioritized feature extraction with shadow recognition,” in *Proceedings of 2017 22nd International Conference on Digital Signal Processing (DSP)*, pp. 1-5, 2017.

[49] Xue-Fei Yang, Wan-Chi Siu, Wan-Lam Hui, Calvin Cheung, Hao Wu, Jun-Jie Huang, Zi-Jun Wang, Bo-Chuan Du and etc, "Trial of LRV Close-up Monitoring System by image recognition technology-Final report," *Internal document K1439 (unpublished)*, The Hong Kong Polytechnic University, HKPC Automotive and Electronics Division, MTR, pp. 85-91, 2015.

[50] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.