Real-Time Video Analytics Empowered by Machine Learning and Edge Computing for Smart Transportation Applications

Ruimin Ke

A dissertation

submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

University of Washington

2020

Reading Committee:

Yinhai Wang, Chair

Ed McCormack

Xuegang (Jeff) Ban

Program Authorized to Offer Degree:

Civil & Environmental Engineering

University of Washington

Abstract

Real-Time Video Analytics Empowered by Machine Learning and Edge Computing for Smart Transportation Applications

Ruimin Ke

Chair of the Supervisory Committee:
Yinhai Wang, Professor
Civil & Environmental Engineering

Traffic cameras have the properties of being cost-effective, information-rich, and widely deployed, which are filling up a big gap in today's traffic sensor needs. With the recent progress in traffic operations, information technology, and computer vision, traffic video analytics is driving a broad range of smart city applications with great potential to benefit future transportation and infrastructure systems. Most such applications, e.g., smart traffic surveillance and autonomous driving, require not only high intelligence but also real-time processing capability. Real-time video analytics is well-believed to be one of the most challenging yet most powerful applications for

smart cities. It is often bottlenecked by the large volume of video data, high computational cost, and limited data communication bandwidth.

This dissertation explores general guidelines and new traffic video analytical methods and systems towards high intelligence and real-time operations for roadway transportation. The designs focus on both the algorithm level and the application system level. On the one hand, lightweight methods are devised based on machine learning techniques and transportation domain knowledge for high smartness, accuracy, and efficiency in specific traffic scenarios. On the other hand, system architectures are developed by leveraging the power of edge computing so that we can split the computational workload between the centralized servers and local Internet-of-Things (IoT) devices for the purpose of system performance optimization.

The traffic analytics products and findings in this dissertation can be applied to three transportation-related scenarios with different properties regarding video data collection and processing: (1) traffic surveillance, (2) vehicle onboard sensing, and (3) unmanned aerial vehicle (UAV) sensing. Correspondingly, they apply to three key components of modern intelligent transportation systems (ITS), i.e., smart infrastructures, intelligent vehicle, and aerial surveillance for road traffic. These components possess unique characteristics that can be utilized for video analytics, yet with different challenges to address. To this end, the dissertation proposes algorithms, frameworks, and field implementation examples of how to design and evaluate traffic video analytics systems for smart transportation applications towards high intelligence and efficiency. Experiments were conducted with real-world datasets and tests in a variety of scenarios. This dissertation is among the first efforts in developing edge computing applications for transportation and in exploring UAV sensing for traffic flow.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

It is every interaction between you and the external world that makes you who you are today. It is every person I met and everything I experienced that brings me to where I am and makes this dissertation possible. I would like to express my gratitude from the bottom of my heart to everyone who makes this journey special, enjoyable, and fruitful.

I am extremely fortunate and grateful to be advised by Prof. Yinhai Wang throughout my master's and doctoral study. Without his valuable advice and constant support, I could never reach this big milestone. Prof. Wang is not only my academic advisor but also my mentor who inspires me in many other aspects. I am astounded at his ability to see the nature of many problems and effectively dedicate his time and energy to explore solutions. His attitude towards life and career always motivates me to strive for greatness and be a good person.

I am sincerely thankful to Prof. Ed McCormack, Prof. Xuegang (Jeff) Ban, and Prof. Ming-Ting Sun, for serving on my doctoral supervisory committee. I cherish your advice and feedback, which have greatly improved my research in this dissertation. I enjoy our discussions, which have inspired my thinking and let me see the insights of different problems. Your help and support in many other aspects of my graduate study have also encouraged me to be a better version of myself.

I would like to thank my colleagues and friends at the Smart Transportation Applications and Research (STAR) Lab: Zhiyong Cui, Kris Henrickson, Yifan Zhuang, Zhibin Li, John Ash, Ziyuan Pu, Wenbo Zhu, Cole Kopca, Frank Yang, Chenxi Liu, Meixin Zhu, Shuyi Yin, Chris Gottsacker, Sonia Xiao, and many others, for your tremendous help and collaboration. I also thank the Pacific Norwest Transportation Consortium, Federal Transit Administration, Sound Transit, Washington

State Department of Transportation, Norwegian Public Road Administration, Transportation Research Board, and so on for the funding support throughout my graduate study.

I am fortunate to have worked with and learned from my collaborators: Jerome Lutin, Jerry Spears, Heidi Soule, Dave Valadez, Dan Sellers, Andrew Krum, Xinqiang Chen, Jinjun Tang, Shuo Feng, Wan Li, Brian Brooke, Franz Loewenherz, Torgeir Vaa, Haena Kim, and so on. A special thanks to Dr. Jerome Lutin, who is also a mentor of mine in research and career development during my doctoral study. Thanks to professors Guohui Zhang, Cathy Liu, Xiaolei Ma, Yao-Jan Wu, Lijun Sun, Sean Qian, and Lei Zhang, who have offered helpful suggestions and supports to me as senior researchers.

The first step is often the most difficult. I had the great fortune of starting my journey in the field of intelligent transportation systems (ITS) when I was an undergraduate student at Tsinghua University. I joined the ITS lab in the Department of Automation and started to learn about ITS. Thanks to professors Danya Yao, Yi Zhang, Li Li, Jianming Hu, Zuo Zhang, Xin Pei, Meng Li, and Diange Yang at Tsinghua who guided me into this fascinating research field.

My life in Seattle would not be as colorful without the great times I spent on badminton courts with my badminton friends: Yicheng, Sushant, Vu, Naim, Li-Ang, John, Ron, Norm, Curtis, David, Richard, Chris, Bettina, Ploy, Joseph, Kevin, Wen-Sung, Jared, Ben, Harrison, Yiming, Gary, Robin, Chang, Linbo, Fangmao, Guozheng, Xiaojuan, Lawrence, Xiaosong, Kun, Xinyi, etc.

The tremendous support from my family and loved ones, even has been remote in this journey, is as warm as you are right next to me. I am grateful for the supports and efforts from Dr. Xuan Zhou that we have together maintained this great long-distance relationship. Finally, I dedicate this dissertation to my parents, Junwei Qiu and Xizheng Ke, whose love makes everything possible.

PART I: INTRODUCTION AND LITERATURE REVIEW

Chapter 1. INTRODUCTION

1.1 DATA EXPLOSION AND SMART CITY

Urbanization has been posing great opportunities and challenges in different areas, including environment, health care, economy, housing, transportation, etc. The opportunities and challenges boost the fast advances in cyber-physical technologies and bring connected mobile devices to people's daily life. Nowadays, approximately 90% of people are connected to the internet and has fast access to a variety of information (Internet/Broadband Fact Sheet 2019). The superb communication infrastructure in the urban area has been attracting more and more population to cities at an unprecedented scale and speed (Levin and Downes 2019). In order to efficiently manage the data generated every day and use them to allocate urban resources better, the Smart City concept has been brought into people's sight. This concept combines sensors, system engineering, artificial intelligence (AI), and information and communication technologies for the optimization of city services and operations.

Smart city applications have a high demand for computing services to process, analyze, and store big data. Conventionally, big data were mostly born and processed on the cloud datacenters, e.g., social media data and online shopping data. Cloud computing was widely recognized as the best computing service for big data processing and AI tasks. Nevertheless, with the urban data enlarged at explosive speed, cloud computing is no more the optimal solution in many cases because it not only consumes large bandwidth but also brings latency in information transmission. Meanwhile, in some extreme situations where there is a limited internet connection (speed or volume limitation), it will be challenging to transmit and process all the data on the cloud or run data processing in an online manner.

As a key driver for recent smart city applications, big data has been going through a radical shift from the cloud datacenters to the edge of the network, closer to where they are generated, such as smart sensors, mobile units, and the Internet of Things (IoT) devices. An estimation states that about 20 ZB data traffic can be handled by all the cloud datacenters globally by 2021 (Z. Zhou et al. 2019), while in total 850 ZB data will be generated at the network edge at that time. If following the traditional cloud computing framework to transmit all the raw data to the cloud for processing and storage, not only will the network capacity be overwhelmed, but the privacy and cybersecurity concerns will also be escalated.

The science and engineering community is aware of the need to shift the computing workload away from the centralized cloud to the logical edge of the network. Edge computing, as a new solution to address this data explosion challenge, allows data generated from edge devices to be handled closer to the local clients where it is produced. In edge computing, data is processed by an IoT device itself or by a local computer, rather than all being transmitted to the cloud datacenters. Edge computing is efficient in network-wide data processing given its (1) reduced communication bandwidth, (2) reduced computation load at the centralized cloud, and (3) reduced overall cost for sensors, computation, data transmission, and maintenance. It is also more scalable, and at the same time makes privacy protection easier by eliminating or reducing the use of raw data on the cloud.

The data explosion offers massive opportunities for smart city applications. In recent years, big data facilitate the rapid progress in machine learning, which can learn the data patterns and extract potentially useful information from the data generated by the city-wide deployment of sensors, actuators, and IoT devices. Machine learning has empowered innovations and greatly changed the environment we live in, from online services to offline activities in almost every aspect

of our world. Machine learning models have high demands for data and computation power. The training of a machine learning model, especially deep neural networks (DNNs), may take days or even weeks to complete. In most cases, inference with ML models is also less efficient than traditional statistical methods.

Earlier, machine learning was often operated on the cloud datacenters, or at least a decent computer with powerful computation and storage supports. Under the context of edge computing, machine learning is also rapidly shifting from the cloud to the decentralized network edges to enhance the intelligence of smart city applications. This is an unstoppable trend for smart cities, however, it creates new barriers we need to overcome. Given the resource and energy constraints on IoT devices, it is very challenging to perform machine learning tasks with high demands for computation resources. On the other hand, high efficiency is critical in many emerging fields, which sets up another challenge for the design and implementation of machine learning techniques in smart city applications.

1.2 Video Analytics for Intelligent Transportation Systems

Intelligent Transportation Systems (ITS) has been a concept in the engineering field for some time, with applications in parking management, congestion management, traffic signal control, electronic toll collection, traffic incident management, speed limit control, emergency vehicle notification systems, and so on. ITS mainly focuses on improving and optimizing traffic performances using existing infrastructures, instead of building extra roads and facilities. Recently, with the fast emergence of new transportation applications such as connected vehicles, deep learning in transportation, mixed transportation autonomy, and cyber-physical transportation systems, the term ITS has been given new meanings that target higher automation, better

connectivity, smarter learning ability, faster decision making, larger scale, and finer data resolution.

Among the existing transportation infrastructures, camera stands out as one of the most widely deployed, information-rich, and low-cost monitoring devices. Cameras have already been installed in large scale at intersections, roadway segments, freeway ramps, parking lots, and as dashcam in personal vehicles and public transit vehicles. Camera sees what human eyes can see. While human beings rely on eyes to perceive visual information for complicated tasks like driving, walking, and biking, some experts believe that autonomous driving perception can be achieved relying on camera sensors (Yan Wang et al. 2019), which are much cheaper than advanced sensors like Lighting Detection and Ranging (LiDAR). Although cameras have shown great performances in supporting transportation management, they still carry a huge potential for even more advanced smart transportation applications. The potential comes from the way camera sensors sample the surrounding environment and the rich information with the collected the images or videos.

Video analytics in ITS analyzes real-time or recorded video streams and generates metadata about transportation-related objects, interactions, and events that are useful for traffic status understanding, traffic management, and decision making. There are three major types of video analytics for ITS applications, each tie to a specific source of video data: (1) traffic surveillance camera, (2) ground vehicle onboard camera, and (3) Unmanned Aerial Vehicle (UAV) camera or drone camera.

Traffic surveillance cameras are installed at fixed locations for traffic surveillance of roadways, intersections, curbside spaces, or parking garages. Most surveillance cameras serve solely for monitoring purposes, while a few of them have been turned into smart sensors to automate traffic monitoring tasks. Regarding video analytics, surveillance video processing is

easier than onboard video and UAV video given its fixed location and static video background. Static background makes several key tasks in traffic surveillance solvable without complex procedures: (1) separating foreground from the background, (2) determining the regions of interest, and (3) camera calibration. The current trend in traffic surveillance video processing lies in multi-camera integration, real-time surveillance, and semantic understanding of the environment.

Onboard cameras are conventionally installed on ground vehicles for driving condition recording or liability-related issues. Onboard video analytics are generally harder than surveillance video analytics because of its moving background. Separating the video background and foreground is very difficult with traditional methods. Camera calibration is also more difficult since the extrinsic parameters constantly change. However, the recent process in machine learning and computer vision has enabled the detection of objects, roads, scene flow, depth, driver behavior, passenger motion, and so on in the onboard camera videos with high accuracy. Particularly, deep learning offers a choice of object detection in onboard cameras based on their appearances, thus, modeling the complex video background motion can be skipped in many cases or handled in different ways. Recently, onboard camera video analytics has become one of the hottest topics in the science and engineering community due to the applications of camera sensors in autonomous driving.

UAVs are gaining popularity in smart city applications as an emerging sensing platform due to their low cost, high flexibility, and wide view range. UAV sensors have not been widely deployed but have generated much research interest due to their promising potential for a variety of applications. UAV video also has a moving background, but the background motion patterns are different from onboard videos. The purpose of UAV-based traffic video analytics is similar to fixed traffic surveillance video analytics, yet the irregular motion of the background makes

traditional surveillance video analytics methods not suitable. In preliminary studies, researchers focused on either UAV videos with static backgrounds or some straightforward research tasks that could be completed without handling the UAV motion issue. More recent studies addressed this issue by designing methods like frame stitching and motion-vector clustering, which advanced the application of UAV to meet the needs and vision of ITS.

1.3 Motivation and Challenges for Real-Time Capability and High Intelligence

For engineering systems, there is always a tradeoff between efficiency and intelligence. High intelligence entails high computation power and system complexity, which is commonly the opposite of high efficiency. However, both high efficiency and high intelligence are desirable for smart transportation applications. In some application cases, processing latency is not tolerable. For example, smart traffic surveillance needs immediate information to automate real-time control and management. Without real-time capability, there is no way some tasks can be delivered, such as autonomous driving. UAV sensing for incident management also requires the real-time capability to support fast response and reduce side consequences, e.g., incident-induced traffic delay. On the other hand, high intelligence is necessary to deliver innovative functions, complete sophisticated tasks, and increase the accuracy, reliability, and level of automation.

Under the context of data explosion in smart cities, the fusion of edge computing and machine learning is natural to enhance real-time capability and intelligence. The decentralized structure of edge computing offloads the computation on the cloud datacenters, thereby laying the foundation for real-time operations at a city scale. The big data produced by edge devices would be wasted unless being processed at the edge with machine learning techniques to fully unlock their potentials. The data insights extracted by machine learning not only saves network bandwidth for

data communications but also enables smarter and faster decision making in response to the rapidly changing surroundings. High intelligence and efficiency at individual edge nodes then contribute to the overall region-wide network to aggregate the multi-source and multi-location information so that even greater intelligence can be achieved.

Traffic video, as a major source of data in smart cities, needs enormous computing power and storage resources. Video has a unique data structure compared to most other types of data. It is composed of image frames, often 20 – 30 frames per second; each image frame is a 2D array consisting of thousands of quantized numbers (i.e., pixels). This unique data structure of video contains a large data volume yet massive amount of useful information. It creates both enormous opportunities and great challenges for realizing real-time capability and high intelligence for smart transportation applications.

Machine learning has tremendously improved the intelligence obtained from video analytics, however, in many cases, real-time capability is far from being realized. Edge computing, while regarded as an inevitable trend in computing services, is considered a killer app for real-time video analytics because of the resource constraints on edge devices, the high computation cost of video data, and the requirement for low latency (Ananthanarayanan et al. 2017). Particularly, traffic scenes are usually very complex with mixed types of objects, unpredictable motions, various environments, challenging lighting conditions, shadows, occlusions, and so on. In addition, there is no uniformly recognized standards for traffic cameras, hence, traffic cameras have unknown and very different parameters such as focal lengths, resolutions, and view angles. Other disturbances, e.g., camera vibration caused by wind, may also lead to unexpected noises.

1.4 Research Questions and Objectives

In light of these motivations and challenges in the new era of traffic video analytics, key questions of interest include:

- What are the capabilities of video analytics for smart transportation applications under the context of smart cities? What tasks can be automated? What kinds of traffic parameters can be extracted from videos?
- How can machine learning and edge computing be fused to realize real-time video analytics? How can these emerging technologies be integrated with transportation domain knowledge to enhance certain applications?
- How can we take advantage of the different traffic scenes for the design of solutions?
- What are the requirements with respect to computation, hardware, software, and communication for enabling real-time traffic video analytics?
- What are the parts that requires the design of new algorithms, systems, and frameworks? What traditional algorithms can still be of great help for certain functions?
- What are the potential impacts of real-time video analytics for transportation systems, for smart cities, and for our society?

To answer these questions and address the challenges, the main research objectives of this dissertation include:

- To develop light-weight machine learning-based methods that target operating on resource-constraint devices for real-time video analytics
- To design optimized system architectures that balances the computation loads between edge and cloud, or between different algorithms

- To investigate new frameworks and algorithms for extracting useful traffic data (parameters, roads, events, etc.) from videos in a real-time manner
- To design and carry out field implementations to test product prototypes for real-time video analytics on both the vehicle side and the infrastructure side
- To extend video analytics products to enable additional smart transportation applications

1.5 Dissertation Overview and Contributions

This dissertation contains two chapters of literature review and six chapters of main contents. Chapter 2 reviews machine learning applications in ITS, including one sub-chapter discussing the related work on the use of machine learning with edge computing in transportation. Chapter 3 reviews fundamental video analytics elements, which have been individually adopted and as the building blocks for advanced smart transportation applications. These fundamental elements include foreground segmentation, object detection, object tracking, road detection, and camera calibration.

Part I of this dissertation includes this introduction chapter and two review chapters (Chapter 2 & 3). The main contents start from Chapter 4 to Chapter 9. They are summarized as three parts (Part II: Smart Infrastructure, Part III: Intelligent Vehicle, and Part IV: Aerial Sensing) by the application role in smart transportation. Specifically, the motivation for organizing the main contents in the way we divide them into Part II-IV is that: (1) From the transportation system functionality perspective, infrastructure and road users are the two crucial elements that form the ground transportation system; the ground transportation system's functionality is further extended with the emergence of aerial-based surveillance in civil utilization; (2) From the methodological perspective, video properties for surveillance video, vehicle onboard video, and aerial video are each different and unique so that requires different solutions.

Chapter 2 reviews machine learning methods in transportation applications, with a focus on traffic sensing, traffic pattern learning and prediction, and edge machine learning. Chapter 3 reviews the existing works in traffic video analytics, summarizing the research based on the fundamental functionality. Part II contains chapters 4 and 5, in which Chapter 4 focuses on the design and field implementation of a real-time smart parking surveillance system with edge AI, and Chapter 5 is the design and field implementation of a real-time edge AI system for multi-task road and traffic condition monitoring. Part III (Chapter 5 and 6) is about video analytics in intelligent vehicle applications powered by two near-crash detection algorithms proposed in the dissertation. The method in Chapter 5 is adopted for the evaluation of a state-of-the-practice collision avoidance system, and the algorithm and edge computing system proposed in Chapter 6 is backward-compatible to existing vehicles with great transferability for near-crash detection and data fusion in real-time. Part IV fills the gaps in aerial sensing of road traffic by developing deep learning-based object detectors and frameworks for online UAV video analytics. Specifically, they address the video background motion challenges in aerial sensing and automate the collection of traffic flow parameters.

Table 1-1 summarizes the contributed categories of this dissertation. The contribution can be perceived from three perspectives. First, the overall contribution includes methodological contributions to the algorithm level, the system level, real-time and online processing capability, machine learning techniques, and edge computing. The second category of contribution lies in the field of video analytics, divided into sub-categories by the definition of the five fundamental elements. Last but not least, the contributions to smart transportation applications are summarized, including traffic flow parameters estimation, motion estimation for individual

vehicles/pedestrians, detection of the interactions between road users, vehicle collision avoidance in the broad traffic safety area, parking availability study, and road information extraction.

Table 1-1 Dissertation Contribution Areas

	Part II: Smart Infrastructure	Part III: Intelligent Vehicle	Part IV: Aerial Sensing
Overall Contributions			
Algorithm Level	√	√	√
Application System Level	√	√	√
Real-Time Capability	√	√	√
Machine Learning	√	√	√
Edge Computing	√	√	-
Contributions to Video Analytics			
Foreground Segmentation	√	√	√
Object Detection	√	-	√
Object Tracking	√	√	√
Road Detection	√	-	√
Camera Calibration	√	√	√
Contributions to Smart Transportation			
Traffic Flow Parameters	√	-	√
Individual Motion Estimation	√	√	√
Road User Interaction	√	√	√
Collision Avoidance	-	√	-
Parking Availability	√	-	-
Road Information	√	-	√

Chapter 2. REVIEW OF MACHINE LEARNING IN INTELLIGENT TRANSPORTATION SYSTEMS

Machine learning arises in transportation-related applications where we see a significant amount of transportation data being collected and a need for more efficiency, safety, sustainability, and automation. There are roughly three major components in intelligent transportation systems: 1) traffic sensing and data pre-processing, 2) traffic pattern learning and prediction, and 3) decision making (control and optimization). While machine learning techniques show effectiveness in all three, this chapter is focused on the first two components due to their high relativeness to the key contributions of this dissertation. In addition, another category of related work is added, focusing on a summary of machine learning with edge computing in transportation.

2.1 Traffic Sensing and Data Pre-Processing

Sensing is essentially the detection of certain types of signals in the real-world and the conversion of them into readable data. Traffic sensors generate data that supports the analysis, prediction, and decision making of intelligent transportation systems. There are various sensors for different data collection purposes and scenarios. The most commonly seen traffic sensors in today's roadway networks and transportation infrastructures include but not limited to inductive loop detectors (Ban et al. 2009; Sharma, Bullock, and Bonneson 2007; Yinghai Wang and Nihan 2000), magnetic sensors (Cheung et al. 2005; Haoui, Kavaler, and Varaiya 2008), cameras (Buch, Velastin, and Orwell 2011; Datondji et al. 2016), infrared sensors (Odat, Shamma, and Claudel 2018), LiDAR sensors (J. Zhao et al. 2019), acoustic sensors (Sen, Siriah, and Raman 2011), Bluetooth (Malinovskiy et al., 2012), wi-fi (Dunlap et al. 2016), mobile phones (Herrera et al. 2010), and probe vehicle sensors (Ban, Hao, and Sun 2011; Kamyab et al. 2020; McCormack and

Hallenbeck 2006). These sensors measure the feature quantity of some objects or scenarios in transportation systems, such as road users, traffic flow parameters, congestion, crashes, queue length at intersections, automobile emission.

There are two types of machine learning applications in traffic sensing: 1) the conversion of raw signals into formatted readable data and 2) data pre-processing for quality control. For most sensor signals, while there are a lot of valuable information that can be mined from them, the conversion process is straightforward and can be completed based on some simple rules. For example, loop detectors measure the change in the inductance when vehicles pass over them for traffic volume and occupancy detection; Bluetooth sensors capture the radio communication signal with a device unique identifier, i.e., the media access control address (MAC), so that they can estimate the number of devices (usually associated with the number of road users) or travel time; acoustic sensors generate acoustic wave to detect the existence of objects at a certain location, without the ability to tell the object type. For some sensors, like camera and LiDAR, the conversion of the raw signals (i.e., the digital images and the 3D point cloud) to useful data can be quite complicated, and thereby machine learning has been applied widely into the conversion of LiDAR and camera signals.

LiDAR has been predominately used in autonomous vehicles compared to its use in transportation infrastructure systems. LiDAR signal is 3D point cloud and it can be used for 3D object detection, 3D object tracking, lane detection, obstacle detection, traffic sign detection, and 3D mapping in autonomous vehicle's perception systems (Van Brummelen et al. 2018). For example, Qi et al. proposed PointNets, a deep learning framework for 3D object detection from RGB-D data that learned directly from the raw point clouds to extract 3D bounding boxes of vehicles (Qi et al. 2017). Allodi et al. proposed using machine learning for combined

LiDAR/stereo vision data that did tracking and obstacle detection at the same time (Allodi et al. 2016). Jung et al. designed an expectation-maximization based method for real-time 3D road lane detection using raw LiDAR signals from a probe vehicle (Jung and Bae 2018). Guan developed a traffic sign classifier based on a supervised Gaussian-Bernoulli deep Boltzmann machine model, which used LiDAR point cloud and images as input (Guan et al. 2018). There are also some representative works providing critical insights into the application of LiDAR as an infrastructure-based sensor. Zhao et al. proposed a clustering method for detection and tracking of pedestrians and vehicles using roadside LiDAR (J. Zhao et al. 2019). The findings are helpful for both researchers and transportation engineers.

Camera collects images or videos, and these raw data are essentially some 2D matrices with quantized pixel numbers that are samples of the real-world visual signals. Machine learning techniques are applied to convert these complex 2D matrices into traffic-related data. One fundamental application is object detection. Researchers in the engineering and computer science fields have spent a lot of effort in designing smart and fast object detectors recently using traditional machine learning (Buch, Velastin, and Orwell 2011) and deep learning techniques (Redmon et al. 2016; Ren et al. 2015). Object detection localizes and classifies cars, trucks, pedestrians, bicyclists, etc. in traffic camera images, and enables different data collection tasks. There are also datasets being collected and published specifically for object detection and classification in traffic surveillance images, and has been generated much interest (Z. Luo et al. 2018). AI City Challenge is a leading workshop and competition in the field of traffic surveillance video data processing (Chang et al. 2020). It has been guiding the direction of applying machine learning in some of the most popular and challenging tasks in this field, such as traffic volume counting, vehicle re-identification, multiple-vehicle tracking, and traffic anomaly detection. Since

camera sensor is a critical component of autonomous vehicles, machine learning is heavily deployed in autonomous vehicles' perception systems for understanding the environment (Van Brummelen et al. 2018). Traffic events, congestion levels, road users, road regions, infrastructure information, road user interactions, etc. are all meaningful data that can be extracted from the raw images using machine learning. Datasets have also been published and widely recognized to facilitate the design of cutting-edge machine learning methods for converting camera data into readable traffic-related data (Avola et al. 2020; Geiger, Lenz, and Urtasun 2012; S. Li and Yeung 2017; H. Xu et al. 2017). Chapter 3 introduces a more comprehensive review of video analytical methods and systems in transportation.

Machine learning techniques are also useful for smart data pre-processing. It can help address data quality issues that are hard to be addressed previously. Noisy data and missing data are two major problems in traffic data pre-processing and cleaning. While data denoising can be done with satisfactory performance using traditional methods such as wavelet filter, moving average model, and Butterworth filter (Xinqiang Chen et al. 2020), missing data imputation is much harder since it is adding information properly. Another commonly applied traffic data denoising task is trajectory data map matching. The most popular models for this task that denoises the map matching errors are often based on the Hidden Markov Model (Goh et al. 2012; Mohamed, Aly, and Youssef 2017; Taguchi, Koide, and Yoshimura 2019). There have been quite some efforts in deep learning based missing data imputation lately. These state-of-the-art methods often focuses on learning spatial-temporal features using deep learning models so that able to inference the missing values using the existing ones (Asadi and Regan 2019; Xinyu Chen, Yang, and Sun 2020; Yuanyuan Chen, Lv, and Wang 2020; Duan et al. 2016; K. Zhang et al. 2020; Yifan Zhuang, Ke, and Wang 2018). Given the spatial-temporal property of traffic data, Convolutional Neural

Network (CNN) is a nature choice due to its ability to learn image-like patches. Zhuang et al. designed a CNN-based method for loop traffic flow data imputation, and demonstrated its improved performance over the state-of-the-art (Yifan Zhuang, Ke, and Wang 2018). Generative adversarial network (GAN) is another deep learning method that are appropriate for traffic data imputation given its recent advances in image-like data generation. Chen et al. proposed a GAN algorithm to generate time-dependent traffic flow data. They made two modifications to the standard GAN on using the real data and introducing a representation loss (Yuanyuan Chen, Lv, and Wang 2020). GAN is also experimented for travel time imputation using probe vehicle trajectory data. Zhang et al. developed a travel times imputation GAN (TTI-GAN) considering the network-wide spatial-temporal correlations (K. Zhang et al. 2020).

2.2 Traffic Pattern Learning and Prediction

With the data organized, the next step for an intelligent transportation system is to learn the traffic patterns, understand traffic status, and do traffic prediction. There are two critical steps in most machine learning tasks for traffic pattern learning: 1) feature selection and 2) model design. Essentially, feature selection forms the original data space, and model design converts the original space to a new space that is learnable for classification, regression, clustering, or other tasks. On the one hand, traffic sensing is so important that without original data property collected, it is almost impossible to make up a new space for pattern learning from poor original data space. One the other hand, once the traffic sensing is done with good design and quality, it is then necessary to focus on designing models to extract the useful information for your tasks. Machine learning has been widely applied for a variety of traffic pattern learning tasks, such as driver and passenger classification using smart phone data (Torres, Ohashi, and Pessin 2019), K-means clustering for truck bottleneck identification using GPS data (W. Zhao et al. 2013), estimation of the number of

bus passengers using deep learning (Y. W. Hsu, Chen, and Perng 2020), and faulty detection in vehicular cyber-physical systems (Sargolzaei et al. 2017).

A traditional group of studies using machine learning is transportation mode recognition. Machine learning models are developed to recognize the mode of travelers, such as working, biking, running, and driving. This can be achieved by identifying travel features like speed, distance, and acceleration. Jahangiri and Rakha applied multiple traditional machine learning techniques for mode recognition using mobile phone data, and found Random Forest (RF) and Support Vector Machine (SVM) to have the best performances (Jahangiri and Rakha 2015). Ashqar et al. enhanced the mode recognition accuracy by designing a two-layer hierarchical classifier and extracting new frequency domain features (Ashqar et al. 2020). Another work introduced an online sequential extreme learning machine (ELM) which focuses on transfer learning techniques for mode recognition. It was trained with both labeled and unlabeled data for better training efficiency and classification accuracy. Recently, deep learning models were also developed for mode recognition (Zhenyu Chen et al. 2013). Jeyakumar et al. developed a convolutional bidirectional Long Short-Term Memory (LSTM) model for transportation mode recognition. Feature extraction includes time domain and frequency domain features from the raw data (Jeyakumar et al. 2018).

Another representative group is using machine learning for traffic accident detection. It is beneficial for transportation management agencies and travelers to have real-time information of traffic accidents regarding where it occurs and what the situation is. Otherwise, it may cause severe congestion and other issues besides the accident itself. This group of work often extract features from traffic flow data, weather data, and so on to identify the traffic pattern change or differences around the accident location. Parsa et al. implemented eXtreme Gradient Booting (XGBoost) to

detect the occurrence of accidents using real-time data including traffic flow, road network, demographic, land use, and weather information. Shapley Additive exPlanation (SHAP) is employed for interpretation of the results for the analysis of importance of individual features (Parsa et al. 2020). They also led another study that showed the superiority of probabilistic neural network for accident detection on freeways using imbalanced data. It revealed the speed difference between the upstream and downstream of the accident was very significant (Parsa et al. 2019). In addition to traffic flow data, social media data is also shown to be effective for traffic accident detection. Zhang et al. employed Deep Belief Network (DBN) and LSTM in the detection of traffic accident using Twitter data in Northern Virginia and New York City. They found that nearly 66% of the accident related tweets can be located by the accident log, and over 80% can be linked to abnormal traffic data nearby (Zhenhua Zhang et al. 2018). Another sub-category is to detect accident in real-time from vehicle's perspective. For example, Dogru and Subasi studied the possibility of applying RF, SVM, and neural network for accident detection based on individual vehicle's speed and location under the context of Vehicular Ad-hoc Network (VANET) (Dogru and Subasi 2018).

Traffic pattern learning is fundamental to traffic prediction. In traffic prediction, the models extract features and learn the current pattern of traffic in order to predict some measurements. Traffic prediction is crucial for the intelligence. It is one of the areas where artificial intelligence, especially deep learning techniques, have been heavily applied. Traffic prediction has covered many tasks in transportation systems. Examples are traffic flow prediction (Z. Cui et al. 2020; Zhiyong Cui et al. 2019; Guo et al. 2019; Ruimin Ke, Li, et al. 2020; Y. Lv et al. 2015; X. Ma et al. 2015; Y. Zhang et al. 2019), transit demand prediction (Noursalehi, Koutsopoulos, and Zhao 2018; J. Zhang et al. 2020), taxi or ride hailing demand prediction (Geng et al. 2019; J. Ke, Zheng,

et al. 2017; Yao et al. 2018), bike sharing-related prediction (L. Lin, He, and Peeta 2018; C. Xu, Ji, and Liu 2018), parking occupancy prediction (Yang et al. 2019), pedestrian behavior prediction (Ridel et al. 2018), and lane change behavior prediction (Tang et al. 2018). LSTM, CNN, GAN, and graph neural network are some of the most widely used deep learning methods for traffic prediction. The current trend of traffic prediction is larger scale, higher resolution, higher prediction accuracy, and real-time speed. For instance, Ma et al. investigated the feasibility of applying LSTM for single-spot traffic flow data prediction (X. Ma et al. 2015). Their work is a milestone in this field and has laid the foundation for sophisticated models that can capture network-wide features for large-scale traffic speed prediction (Z. Cui et al. 2020; Zhiyong Cui et al. 2019; Y. Zhang et al. 2019).

2.3 Machine Learning with Edge Computing

Data has been increasingly generated in transportation systems, thus, it would be more efficient to process the data closer to where they are generated. Edge computing concept has pushed the horizon of computing paradigm, and brings massive opportunities to smart city and smart transportation. The benefit of edge computing lies in the improvement in computation efficiency, network bandwidth usage, response time, cyber security, and privacy (Shi et al. 2016). However, the resource constraints on edge devices are the key bottleneck for the implementation of high intelligence. Zhou et al. conducted a comprehensive survey on edge AI and considered edge computing is paving the last mile of AI (Z. Zhou et al. 2019). In terms of AI model optimization at the edge, the compression of deep neural networks using pruning and quantization techniques are significant (S. Han, Mao, and Dally 2015; Rastegari et al. 2016). In transportation applications, there have been not many but a few pioneering studies that explore the way of designing both system architectures and algorithms for certain transportation scenarios using edge

computing. It is widely aware that edge computing with machine learning is a trend for smart transportation applications. Ferdowsi et al. introduced a new ITS architecture that relies on edge computing and deep learning to enhance computation, latency, and reliability (Ferdowsi, Challita, and Saad 2019). They investigated the potential of using edge deep learning to solve multiple ITS challenges including data heterogeneity, path planning, autonomous vehicle and platoon control, and cyber security.

Crowdsensing with Internet of Vehicles (IoV) is one category of research using edge computing for ITS. Vehicles are individual nodes in the traffic network with local data collection and processing units. As a whole, they form the IoV network that can do crowdsensing. One example is that monitoring urban street parking spaces with in-vehicle edge video analytics (Grassi et al. 2017). In this work, smart phone serves as the data producer; the edge unit detects cars, signs, GPS and upload the ego-vehicle location and road identifier to the cloud for data aggregation. Another study uses crowdsensing scheme and edge machine learning for road surface condition classification. Multi-classifier is applied at the edge to recognize road surface type and anomaly situation (El-wakeel et al. 2018). Liu et al. proposed SafeRNet, a safe transportation routing computation framework that utilized Bayesian network to analyze crowdsensing traffic data to infer safe routes and deliver them to users in real time (Q. Liu, Kumar, and Mago 2020). Some other studies focus on managing and optimizing resources of the system to ensure efficient message delivery, computation, caching, and so on for IoV (Dai et al. 2019; He et al. 2017; X. Wang et al. 2018; X. Xu et al. 2019; Yuan et al. 2018). Dai et al. exploited reinforcement learning (RL) to formulate a new architecture that dynamically optimize edge computing and caching resources (Dai et al. 2019). Yuan et al. proposed a two-level edge computing architecture for

efficient content delivery of large-volume, time-varying, location-dependent, and delay-constrained automated driving service.

Another group of research on this topic focuses on developing machine learning methods for certain ITS tasks with edge computing, instead of for resources management in crowdsensing. Real-time video analytics, as the killer app for edge computing, has generated challenges and thereby huge interests for research (Ananthanarayanan et al. 2017; Barthélemy et al. 2019). Microsoft Research has explored a new architecture with deep learning and edge computing techniques for intersection traffic monitoring and potential conflict detection (Ananthanarayanan et al. 2017). Detecting parking space occupancy by light-weight CNN models on edge devices has also been investigated by different group of researchers (Amato et al. 2017; Ling et al. 2017; J. Zhou, Dai, and Wang 2019). Another light-weight CNN that comprised of factorization convolution layers and compression layers was developed for edge computing and multiple object detection on Nvidia Jetson device for transportation cyber-physical systems (J. Zhou, Dai, and Wang 2019). Cyber attack can also be detected in transportation cyber-physical systems using machine learning. Chen et al. proposed a deep belief network structure to achieve attack detection in a transportation mobile edge computing environment (Yuanfang Chen et al. 2019). UAV can also serve as edge unit for attack detection for smart vehicles (Garg et al. 2018). Another interesting application of edge machine learning is detecting road surface quality issues onboard a vehicle (Kulkarni et al. 2014; Zheng et al. 2019). Traditional machine learning methods such as random forest appeared to perform well with high accuracy and real-time operation for this task.

Chapter 3. REVIEW OF FUNDAMENTAL ELEMENTS IN TRAFFIC VIDEO ANALYTICS

This chapter reviews the fundamental elements of traffic video analytics. The sub-chapters provide concise summaries of the intuition, functionality, and state of the arts of these elements. There are five fundamental elements: foreground segmentation, object detection, object tracking, road detection, and camera calibration. They are also the building blocks for advanced traffic video analytics applications.

3.1 FOREGROUND SEGMENTATION

Foreground segmentation is a crucial component and often the first stage for traffic video analytics. As the name suggests, it is the segmentation of video foreground and background. Single image-based foreground segmentation uses intensity or gradient information, but normally needs some manual initialization of the foreground region, e.g., graph cut, intelligent scissor. In surveillance videos, the most common assumption to differentiate foreground and background is that background does not move so that background pixel's color or intensity at the same location does not change over time. Based on this assumption, temporal information in multiple frames can be used to model pixel values and do a binary classification. Representative methods were proposed including frame differencing (Abdulrahim, Salam, and others 2016; Rahim et al. 2010), averaging (Gupte et al. 2002), single Gaussian (P. Kumar, Ranganath, and Weimin 2003; Morris and Trivedi 2006), Kalman filter (Messelodi et al. 2005), Wavelets (Gao et al. 2008), fuzzy theory (Lu et al. 2014), and Gaussian mixture model (GMM) (M. Chen et al. 2017; N. Kumar and Sureshkumar 2015; Saran and Sreelekha 2015). GMM performs well and has some variants such as a modified GMM for dealing with shadow removal (Martel-Brisson and Zaccarin 2007). In this

work, GMM was applied to identify and remove the shadow of foreground objects, with the assumption that the shadow state was more stable than the foreground state yet less stable than the background state. Image segmentation methods like using graph cut or image textures can be applied for foreground segmentation in moving videos (Kalinke, Tzomakas, and von Seelen 1998; Sturgess et al. 2009).

3.2 Object Detection

Object detection locates and classifies objects. Under the context of smart transportation, the detected objects are road users or transportation system components such as traffic lights and traffic signs. Sun et al. provided a thorough literature review on traditional vehicle detection using onboard cameras (Sun, Bebis, and Miller 2006). Blob detection is often coupled with the aforementioned foreground segmentation methods to locate the separated foreground into individual objects (Bhaskar and Yong 2014). In additional to using intensity features, gradient features (e.g., interest points, edges) and motion features are useful as well for vehicle detection (Jazayeri et al. 2011; Tsai, Hsieh, and Fan 2007). Hand-crafted appearance features are another group of widely used features for object classification and part of object detection, e.g., histogram of gradient (HOG) (X. Li and Guo 2013; Yun Wei et al. 2019), Haar-like features (Yun Wei et al. 2019; S. Zhang, Bauckhage, and Cremers 2014), and local binary patterns (LBP) (Neumann et al. 2017). These features are combined with machine learning classifiers, e.g., support vector machine (F. Han et al. 2006) and boosting (Khammari et al. 2005), and localization methods, e.g., sliding window (Noh, Shim, and Jeon 2015), to detect the objects of interest. Recently, deep learning-based object detection has been popular due to their outstanding capability to learn deep features to localize and classify the objects more accurately. Object detectors relied on deep neural network are either two-stage detectors or one-stage detections. Two stage detectors have a region proposal

stage and a final classification stage, such as Faster R-CNN (Ren et al. 2015). One-stage detectors try to simplify the two-stage procedure to increase the efficiency, e.g., Single Shot multibox Detection (SSD) (W. Liu et al. n.d.), You Only Look Once (YOLO) (Redmon et al. 2016). It is worth mentioning that object detection supports a new direction of foreground segmentation studies. These studies rely on object detection or feature detection to determine the foreground. They do not assume a stationary video background, so that can be applied to onboard videos and UAV videos (Ruimin Ke et al. 2017; Yifan Zhuang, Ke, and Wang 2020).

3.3 Object Tracking

Object detection locates and classifies the road users in a single video frame. A nature subsequent step is to estimate the paths of the detected road users. Tracking method connects the same road users detected in consecutive frames together so that we can get their identifiers and paths. Tracking is in most cases an indispensable step because it helps move forward to the goal of traffic data extraction and understanding from video streams. Example cues for tracking are Intersection over Union (IoU) and motion trends for predicting object's paths (Bochinski, Senst, and Sikora 2018; Bouttefroy et al. 2008). There are offline tracking methods (Gu, Zheng, and Tomasi 2011; Yichen Wei et al. 2007) and online tracking methods (Bewley et al. 2016; Tian, Lauer, and Chen 2019; Q. Wang et al. 2019; Y. Wu, Lim, and Yang 2013), where offline tracking works on recorded video with the assumption that all the video frames are known ahead of time, and online tracking works for live stream with solely the need for frames prior to the current time. For this dissertation targeting real-time video analytics, all the tracking, no matter proposed or existing methods, are all in the group of online tracking. Recently, deep features are incorporated into tracking tasks, which improves the tracking performance regarding re-identification, motion blur, illumination change, occlusion, etc. (Bhat et al. 2018; Hou, Wang, and Chau 2019; H.-M.

Hsu et al. 2019). In additional to tracking object contours or bounding boxes, it is sometimes effective to just track features, such as interest points. It may skip the road user detection part to directly extract useful traffic information (Ruimin Ke et al. 2015), or inversely detect objects based on the feature tracking results (Ruimin Ke et al. 2017).

3.4 ROAD DETECTION

The three key components of ground transportation systems are human, vehicle, and road. Road detection for road information extraction is equivalently important to road user detection in many cases. Road extraction enables the study of the interactions between road users and roads so that richer traffic information can be collected. Road information may also assist road user detection by improving the detection accuracy and efficiency. Hillel et al. introduced a informative survey paper about recent approaches and algorithms in road and lane detection using onboard camera sensors (Hillel et al. 2014). Detection results and codes of some state-of-the-art road detection for advanced driver assistant systems (ADAS) can be found at the Karlsruhe Institute of Technology and Toyota Technological Institute (KITTI) benchmark suite (Fritsch, Kuehnl, and Geiger 2013). Road information in surveillance videos can help locate the road mask to assist traffic information detection. Road boundary detection (Helala, Pu, and Qureshi 2012; Q.-J. Kong et al. 2013), lane detection (Lai and Yung 2000; Yue Wang, Teoh, and Shen 2004), road surface condition classification (Omer and Fu 2010; Shibata et al. 2014; Takeuchi et al. 2012) are critical sub-areas that have been explored in video analytics. Road detection has also been investigated in UAV video analytics. Zhou et al. proposed an efficient road detection and tracking method from low-altitude UAV (H. Zhou et al. 2014). When the UAV altitude is high or even using satellite imagery, GPS data may be needed to help detection the road regions (C. Cao and Sun 2014).

3.5 CAMERA CALIBRATION

Camera calibration solves camera parameters and the mapping between the 3D real-world coordinate and the 2D image coordinate. It is so important in smart transportation applications that it makes recovering physical metrics possible from 2D video frames. In surveillance cameras, the full set of camera parameters can be calibrated assuming they are at fixed locations with fixed angles, including the intrinsic parameters (e.g., focal length) and extrinsic parameter (e.g., rotation and translation). Camera intrinsic parameters do not change so the calibration of them can be done offsite, while extrinsic parameters have to be done onsite after the installation. The most common traffic scene used for calibration is lane marking, which are generally parallel to one another in the 3D world but appear to converge in a 2D image. Parallel lane markings are a set of vanishing lines in the video camera images, which offers a train of thought for calibrating surveillance cameras (Zhaoxue. Chen and Shi 2004; Y. Li et al. 2007). Parallel vehicle tracks can also be used for camera calibration (Kanhere, Birchfield, and Sarasua 2008; Schoepflin and Dailey 2003), and this technique can be applied to scenarios where lane markings are not clearly visible. Work has also been completed using pedestrians or vehicles for surveillance video camera calibration. Researchers have tried to take advantage of motion information in video data when using pedestrians, but most existing work focusing on pedestrians requires an accurate pedestrian detection and even precise foot-head localization (F. Lv, Zhao, and Nevatia 2006; Micusik and Pajdla 2010). Zhang et al. proposed an algorithm using vanishing points obtained from both vehicles and standing pedestrians (Zhaoxiang Zhang et al. 2012). Camera calibration for onboard cameras and UAV videos can be done only for intrinsic parameters since the cameras move. However, in many cases, it is possible to use the intrinsic camera parameters or leveraging other camera properties to get the information needed. In UAV videos, the mapping from the video

frame to the real world can be simplified to a perspective projection model in camera calibration if the UAV height is significantly larger than the object sizes (Shastry and Schowengerdt 2005).

PART II: SMART INFRASTRUCTURE APPLICATIONS

Chapter 4. REAL-TIME PARKING SURVEILLANCE WITH EDGE AI

4.1 Overview and Contribution

Smart parking has been introduced to solve parking sensing and management problems in cities. A recent report shows that people spend 17 hours on average on searching for parking spaces a year, while this number for New York drivers is 107 hours (Al-Turjman and Malekloo 2019). To improve the parking space searching efficiency, we will require smart parking surveillance systems for automatic and online parking occupancy detection. However, it faces the same challenge as other surveillance tasks regarding the computing workload and transmission volume in the video data processing. While there are many video processing studies for smart parking surveillance (Al-Turjman and Malekloo 2019; Alam et al. 2018; Amato et al. 2016, 2017; Baroffio et al. 2015; Bulan et al. 2013; Cho et al. 2018; T. Lin, Rivano, and Le Mouël 2017; Ling et al. 2017; Nieto et al. 2018; Nurullayev and Lee 2019; Rianto et al. 2018; Vitek and Melničuk 2018; Q. Wu et al. 2007), exploring edge computing solutions for parking surveillance is still at an early stage. Pioneering works have investigated implementing machine learning and AI algorithms on IoT devices (Amato et al. 2016; Vitek and Melničuk 2018). While they provide insightful findings to the community, their objectives are not to develop a system for real-world practice.

This chapter proposes an edge computing surveillance system to detect parking space occupancy with smartness, efficiency, and reliability. These three metrics are defined towards the performance goals of our system: smartness is the automatic detection and pattern recognition in a parking garage scene; efficiency is about processing in a real-time and online manner; reliability means reliable and consistent detection performance in various environmental conditions. The system's processing pipeline and components are carefully designed considering data transmission volume, efficient online processing, flexibility, detection accuracy, and system robustness. This

study implements a background-based detection method and a SSD finetuned on a new traffic surveillance benchmark dataset on the edge devices. On the cloud data server, we improve a state-of-the-art multiple object tracking method and develop an occupancy judgement method that can handle extreme lighting conditions and occlusions. The system is first developed and set up in a lab environment, and then it is deployed in a real-world parking garage for three months. The real-world test demonstrates the system's exceptional performance in various challenging scenarios, and its potential to support a few critical future applications in smart cities.

The contributions of this chapter are six folds:

- This study proposes a new system architecture with IoT and AI technologies for real-time smart parking surveillance, which splits the computation load to local IoT devices and servers targeting optimal system performance
- The data transmission volume is designed to be small to handle the limited network bandwidth issue in real-time video analytics.
- A new pipeline is proposed to perform detection in extreme lighting conditions and occlusion conditions with a combination of background subtraction and SSD detection.
- An SSD-Mobilenet detector is implemented using Tensorflow Lite on the IoT devices with transfer learning on the MIO-TCD traffic surveillance dataset.
- A tracking algorithm is designed to operate on the server side for vehicle tracking in parking garages.
- The thorough experimental results and findings from a variety of real-world scenarios can be a valuable reference for future research.

4.2 Related Work

From the sensing functionality perspective, recent work in the area of parking occupancy detection can be divided into three categories: wireless sensor network (WSN) solution (Al-Turjman and Malekloo 2019; Grodi, Rawat, and Rios-Gutierrez 2016; Jeon, Ju, and Yoon 2018; Lee, Yoon, and Ghosh 2008; T. Lin, Rivano, and Le Mouël 2017; Lou et al. 2019; Park et al. 2008; Sifuentes, Casas, and Pallas-Areny 2011; Zusheng Zhang et al. 2013; Zusheng Zhang, Tao, and Yuan 2014; H. Zhu and Yu 2015), moving sensor solution (F Bock and Di Martino 2017; F Bock, Di Martino, and Origlia 2019; Fabian Bock, Di Martino, and Sester 2017; Grassi et al. 2017; Houben et al. 2013; Q. Luo et al. 2017; Mathur et al. 2010; Mitsopoulou and Kalogeraki 2018; Peng et al. 2018; Sarkar, Totaro, and Elgazzar 2019; Satonaka et al. 2006), and vision-based solution (Al-Turjman and Malekloo 2019; Alam et al. 2018; Amato et al. 2016, 2017; Baroffio et al. 2015; Bulan et al. 2013; Cho et al. 2018; T. Lin, Rivano, and Le Mouël 2017; Ling et al. 2017; Nieto et al. 2018; Nurullayev and Lee 2019; Rianto et al. 2018; Vitek and Melničuk 2018; Q. Wu et al. 2007).

WSN solution puts one sensor node to each parking space, then multiple sensor nodes are required for the detection of multiple parking spaces. A WSN sensor should be small, sturdy, low power, and cost-effective. Over the past years, WSN sensors with different sensing abilities have been developed and deployed. The most widely used ones are magnetic, ultrasonic, infrared, and loop sensors. For example, Sifuentes et al. design a simple yet effective magnetic-based parking vehicle detection method, which incorporates a wake-up function using optical sensors (Sifuentes, Casas, and Pallas-Areny 2011). Their system reliability is improved over standalone magnetic sensors. Park et al. develop an ultrasonic sensor solution for parking occupancy detection (Park et al. 2008). They design a multiple echo function for more accurate parking space detection than the

single echo function in a real parking environment. The detection algorithms for WSN are commonly very efficient; in most cases, a thresholding method or a straightforward pipeline taking the sensor signals as input would work. However, simple algorithms lead to high false detections in certain scenarios: magnetic sensors are sensitive to large metals nearby, such as a truck in neighboring parking spaces; ultrasonic and infrared sensors can be influenced by the environment noises, such as weather and lighting conditions. Another unique feature of WSN is the large number of sensor nodes, which has high robustness to sensor failure. That is to say, even if a few sensors stop working, the system can still convey quite accurate parking information. However, this feature also leads to a high cost and scalability issue. The installation and maintenance of hundreds of sensors are inefficient, labor-intensive, or even impracticable, especially for in-ground sensors like loops.

We summarize the second category as using moving sensors for parking occupancy detection (F Bock and Di Martino 2017; F Bock, Di Martino, and Origlia 2019; Fabian Bock, Di Martino, and Sester 2017; Grassi et al. 2017; Houben et al. 2013; Q. Luo et al. 2017; Mathur et al. 2010; Mitsopoulou and Kalogeraki 2018; Peng et al. 2018; Sarkar, Totaro, and Elgazzar 2019; Satonaka et al. 2006). This group of work usually uses sensors on phone apps or probe vehicles to monitor urban parking availability via crowdsensing strategies. They can support various smart parking applications in urban areas and be an alternative to static parking sensors. For example, Bock et al. conduct multiple innovative studies on using GPS sensors on the crowd of taxis to sense on-street parking space availability (F Bock and Di Martino 2017; F Bock, Di Martino, and Origlia 2019; Fabian Bock, Di Martino, and Sester 2017). They start the research by answering a question of how many probe vehicles are needed for on-street parking information collection, then prove the availability and investigate more detailed aspects such as misdetection amounts and quality of

sensors. Some other studies explore and test the feasibility of onboard ultrasonic sensors and camera sensors as the moving sensors for crowdsensing (Grassi et al. 2017; Houben et al. 2013; Satonaka et al. 2006). While recent research has demonstrated the enormous potential of crowdsensing for parking occupancy detection in the future, their applicability is still limited to specific scenarios at present. First, the cost can be very high since it requires high penetration rates of sensors (probe vehicles) to obtain sufficient parking information; second, this strategy is suitable for on-street parking detection in urban areas but not for large parking lots or rural areas where there are few moving sensors. In additional to crowdsensing, researchers have also examined single moving sensors for parking occupancy detection, such as drones (Peng et al. 2018; Sarkar, Totaro, and Elgazzar 2019). With the advantage of the flexibility and wide view range, drones are considered an emerging parking sensor with high cost-effectiveness.

The vision-based solution has received increasing attention for parking occupancy detection lately with the advance in computer vision and data transmission technologies (Al-Turjman and Malekloo 2019; Alam et al. 2018; Amato et al. 2016, 2017; Baroffio et al. 2015; Bulan et al. 2013; Cho et al. 2018; T. Lin, Rivano, and Le Mouël 2017; Ling et al. 2017; Nieto et al. 2018; Nurullayev and Lee 2019; Rianto et al. 2018; Vitek and Melničuk 2018; Q. Wu et al. 2007). Compared to WSN, where one sensor covers a single space or moving sensors where one moving unit has one sensor, one camera sensor covers multiple spaces; thus it decreases the cost per parking space. It is also more manageable and efficient since the installation of camera systems is non-intrusive and demands no closedown of parking lots. In addition, camera is information richer than other parking sensors, which has a greater potential to support more advanced parking management. Pioneering studies model the occupancy detection as a binary classification problem on predefined regions using relatively simple features and traditional classification methods (Alam et al. 2018; Baroffio

et al. 2015; Bulan et al. 2013; Rianto et al. 2018; Q. Wu et al. 2007). Baroffio et al. propose a method utilizing hue histogram and linear support vector machine (SVM) (Baroffio et al. 2015). Their method achieves real-time processing and high accuracy on the validation data. Bulan et al. design a pipeline based on background subtraction and SVM, which has a great performance and is robust to occlusion (Bulan et al. 2013). While these traditional methods tend to have an unstable detection performance in relatively complex scenarios, they lay a great foundation for more advanced methodologies. Recently, with the emerging trend in deep learning, researchers have examined the availability of deep learning models for vision-based parking occupancy detection. For example, Nurullayev et al. propose a dilated convolutional neural network (CNN) architecture. With the specific architecture design, it is more robust and suitable for parking occupancy detection (Nurullayev and Lee 2019).

However, vision-based solutions often generate a large volume of data that may increase the cost and unreliability of data transmission. To solve this problem, vision-based systems have been implemented to edge devices instead of transmitting the original videos to the data processing center. Vitek and Melnicuk implement a histogram of gradient (HOG) based classifier on IoT devices, though the HOG feature is still handcrafted which can lead to significant errors in real-world parking scenes (Vitek and Melničuk 2018). Some recent studies combine deep learning and IoT device to realize edge artificial intelligence to improve detection accuracy and reduce data transmission volume. Amato et al. implement CNN classifiers to determine the occupancy status of pre-defined parking spaces. Their work is an essential milestone in the area of parking occupancy detection. Though their CNNs are already quite efficient compared to most standard CNNs such as VGG (Simonyan and Zisserman 2014), they still have a relatively slow classification speed even on a single image (Amato et al. 2016, 2017). Also, for this type of

classification-based parking detection system, people need to manually label each parking space at local IoT devices after the installation, and in practical applications, it can be labor-intensive, not flexible, and not scalable.

4.3 Proposed Solution and Design

The overview of the system design is shown as a flow diagram in Figure 4-1. The system is composed of camera nodes, IoT devices, cellular data transmission modules, and a centralized server. In this study, the IoT devices are Raspberry Pi 3B, yet other IoT devices like Arduino and Jetson Nano could be the alternatives. The overall design considers the balance between computational load and data transmission volume, as well as the reliability and scalability of the system.

Two efficient computer-vision-based object detection algorithms are implemented at the edge as two threads. They utilize the limited computation power of the IoT device to convert the raw video frames to detections in an online manner, thus largely reduce the data transmission volume and ensures efficient updates. Also, one video frame is transmitted to the data server every a few minutes for parking space labeling, results verification, and demonstration purposes.

On the server side, we propose a real-time object tracking algorithm based on SORT (Bewley et al. 2016), as well as occupancy judgement algorithms considering occlusion and extreme lighting conditions. The modified SORT algorithm is implemented on the server side rather than the edge side because this design reduces the computation load at the edge while this implementation does not increase the transmission volume. Background-based occupancy detection results and SSD-based occupancy detection results are combined based on the occupancy judgement algorithms for improved robustness and accuracy.

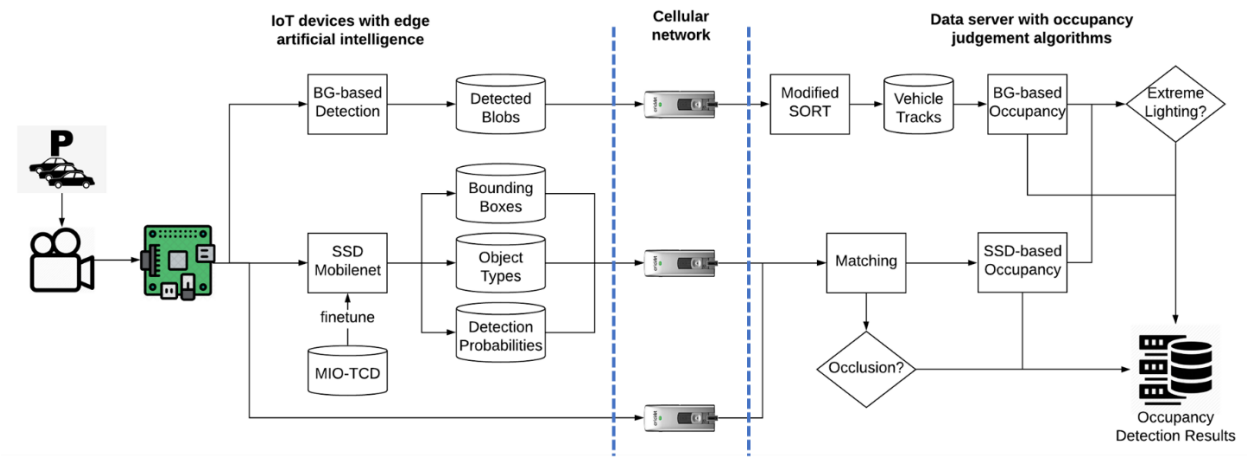


Figure 4-1 Overview of the system solution and design

4.4 CHOICE AND DESIGN OF THE MAIN PIPELINE

There are two major groups of pipelines in camera-based parking occupancy detection methods. In summary, in the first group, binary occupancy classifiers are developed to determine the status (occupied or vacant) of every parking space region in the camera view. The second group applies vehicle detection to localize vehicles in the whole camera view and then determines the status of parking spaces based on the matches of detection results and parking space locations. Both pipelines need a manual labeling process to mark the region of parking spaces that we are interested in. Note that automatic labeling has been attracting some research interests, but still far away from being practicable.

This labeling process has little difference between classification and detection regarding flexibility or workload in traditional server-based parking detection systems, because in either case, the labeling process is done on the server side with raw videos/images directly sent back to the server. However, in an edge computing parking system, we argue that the detection-based pipeline (the second group) is a better choice than the classification-based pipeline.

4.4.1 *The Classification-Based Pipeline and the Concern for Scalability*

First of all, please keep in mind that there are two options for the parking spaces labeling, i.e., locally on the IoT devices or on the server. For the classification-based pipeline, if the classification is done on the server, image patches of parking spaces would need to be transmitted back to the server, which significantly increases the data transmission volume and is not what we want. Hence, the classification task needs to be done on IoT devices, which means the classifier on IoT devices has to know where the parking spaces are. Thus, instead of labeling the parking spaces sitting by a server monitor, we would have to visit all IoT devices at different places, set up a monitor, look at the camera view after installation, and do the labeling. Moreover, once there is a change of the camera view (e.g., angle change or zooming in/out), someone needs to visit that IoT device again. This is not flexible or scalable. Remote connection to the IoT device could be a solution. However, in most cases, the IoT device connects to the internet using wifi or cellular network, which is not secure or friendly to remote access.

4.4.2 *The Detection-Based Pipeline and the Design*

For the detection-based pipeline, the detection has to be done at the edge. Otherwise, the system would turn into a traditional server-based system with raw videos being transmitted back to the server. As aforementioned, there is a matching stage following vehicle detection in the detection-based pipelines. In this study, we propose to move the detection to the edge side while keeping the matching stage on the server side. In this way, the system just transmits the detection results such as bounding boxes to the server for matching, rather than raw videos for detection and matching. With this design, we essentially keep the labeling process on the server side, which is flexible and scalable. To label the parking spaces, we make every edge device send one frame back

to the server. This is a once-and-for-all process, and even if there is a change in the camera view, the relabeling is much less labor-intensive than the classification-based pipeline.

4.5 VEHICLE DETECTION AT THE EDGE

There are two detection methods implemented at the edge of our system: single shot multibox detector (SSD) and background (BG) modeling detector. They work in separate threads at the edge and then their detection results are combined in occlusion or extreme lighting conditions on the server for enhanced performance.

4.5.1 *Enhanced SSD with MIO-TCD for Edge Artificial Intelligence*

SSD with a Mobilenet backbone network is the primary detector. There are different backbones for SSD, while Mobilenet has the lightest structure which makes the detection faster than other backbones. This is appropriate for an IoT device with limited computational power. We recommend using TensorFlow Lite for the SSD implementation since it is designed for deep learning on mobile and IoT devices. A normal state-of-the-art object detector like YOLO-V3 (Redmon and Farhadi 2018) with the TensorFlow platform still runs slowly with a speed lower than 0.05 frames-per-second (FPS) on Raspberry Pi 3B, and has a slightly lower detection accuracy as well. However, SSD-Mobilenet with TensorFlow Lite runs over 1 FPS on the same device according to our test. The detection results including bounding boxes, object type, and detection probabilities (how likely the result is true) are transmitted back to the server. Compared to sending videos, it reduces the data volume by thousands of times (the exact number depends on the number of detections in the video).

TensorFlow models can be converted to TensorFlow Lite models. We recommend training a TensorFlow model and then convert it to the TensorFlow Lite model. In order to improve the

detection performance to make it more appropriate for practical applications, we enhance a pre-trained SSD on the Pascal VOC dataset (Everingham et al. 2010) with a new traffic surveillance dataset called MIO-TCD (Z. Luo et al. 2018), which contains 110,000 surveillance camera frames for traffic object detection training. This dataset includes a variety of challenging scenarios for traffic detection such as nighttime, truncated vehicle, low resolution, shadow, etc. To our knowledge, this is the first time MIO-TCD been adopted for parking detection, and we find it works well.

Some key parameters for the training are listed as follows: the learning rate is 0.00001, the weight decay is 0.0005, the optimizer is Adam, the batch size is 32, and the training-validation split ratio is 10:1. All layers are trainable. The training and validation loss curves, as well as some sample images at certain training steps, are displayed in Figure 4-2.

The enhanced SSD-Mobilenet model demonstrates great performances on traffic detection, especially in challenging surveillance image data. Figure 4-3 shows three examples comparing detection results between SSD trained on Pascal VOC and Pascal VOC + MIO-TCD. In the first column, the pre-trained SSD detects all big targets but misses two small targets in the back; in the second column, the pre-trained SSD misses two vehicles partially blocked by a tree; in the third column where there is snow in the nighttime, the pre-trained SSD misses most of the vehicles. Overall, the enhanced SSD produces much better detection results with few missed detections and no false detections.

4.5.2 *Background-Based Detection at the Edge*

Despite the enhanced performance of the SSD, the detection results are still not universally satisfying if your objective is to apply it to various real-world scenarios due to two reasons: (1) though much improved in speed, the SSD running 1 FPS still does not meet real-time detection at

the edge, which limits the use of video temporal information; (2) deep learning model's performance depends much on the training data, but the training data can never cover all real-world scenarios, so the detector itself could still perform poorly in extreme cases. Standalone SSD-based detection may be a good option for lab demonstration, but not for field practice universally.

With this observation and consideration, we propose to add BG-based detection to the edge. BG-based detection is a widely used traditional method for traffic video surveillance that is sensitive to video noises and has no classification ability (Sen-Ching and Kamath 2004; Yong Xu et al. 2016). But it has two advantages that can help compensate SSD: (1) it is very efficient and operates in real-time locally at the edge; (2) it has a relatively more stable detection performance in extreme scenarios where SSD does not work, though not as good in normal cases. The BG-based detection is followed with a regular blob detection step, then the bounding boxes of the detected blobs are transmitted back to the server.

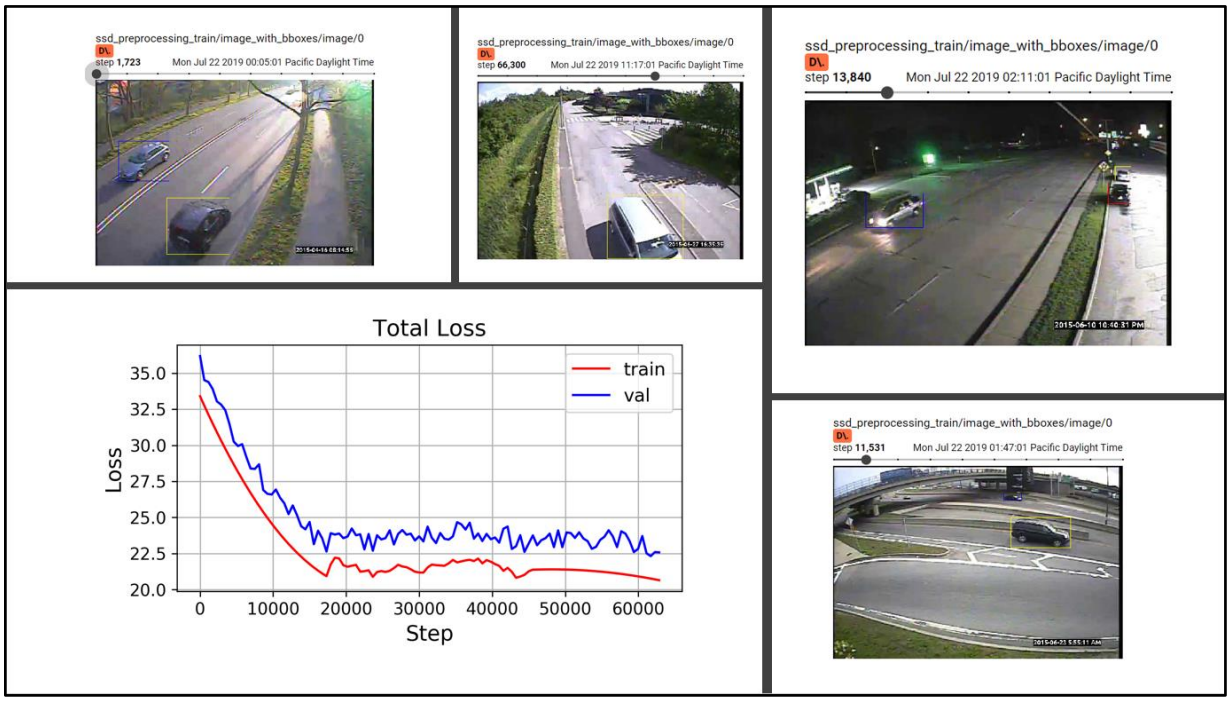


Figure 4-2 The training and validation loss curves and sample images from MIO-TCD at certain training steps.

Figure 4-3 The enhanced SSD-Mobilenet Detector implemented at the edge of our system has a significantly improved detection performance, especially on challenging parking scenarios in surveillance image data.

4.6 DATA TRANSMISSION

The data transmission module in the system is composed of a 4G LTE Huawei USB Modem E397u-53, a T-Mobile data-only SIM card with 6GB monthly, and the software part. The T-Mobile data card is plugged into the 4G modem, and the modem connects with the Raspberry Pi via the USB interface. The connection of the device to the cellular network is activated via the Network Manager API in the software. The Network Manager allows automatic network connection upon start-up and automatic re-connection to the Internet whenever the connection fails. It is a reliable and helpful network connection tool that we recommend for IoT applications.

The reason we use cellular network connection for the data transmission is that the place where we do the field test does not have available wifi or ethernet. This will also be the case for many real-world IoT applications since the cellular network covers most urban areas and quite some rural areas. Other communications like Zigbee and LoRa are getting popular in IoT

applications; however, they are good for short-distance communication rather than remote communication to the server. Cellular network communication is expensive with limited data amount, which, from another perspective, encourages data processing and reduction on edge. With the edge computing modules in the proposed parking system, it transmits BG-based detection results and SSD-based detection results to the server as strings. Also, the system transfers a video frame every ten minutes to the server for demonstration, validation, and space labeling. For an average camera, assuming one frame is 100Kb and the frame rate is 10 FPS (which is usually higher), and the detection results are 40Kb per minute, our system reduces the data transmission amount from around 86Gb per day per device to around 70Mb per day per device.

4.7 Occupancy Judgement Pipeline and Algorithms

With the detection results from the edge, we develop a parking occupancy judgement method on the server. This method first calculates the SSD-based occupancy based on a proposed matching algorithm and BG-based occupancy based on multiple object tracking, then combine them together considering extreme lighting conditions and occlusion conditions.

4.7.1 *SSD-Based Occupancy Detection*

The SSD-based detection results are matched with labeled parking spaces using a proposed matching algorithm. First, we design a metric for calculating the matching score of any space i and detection j. The score V_{ij} is shown below in Eq. (4-1),

$$V_{ij} = IoU(S_i, B_j) \times \sqrt{p_j} \qquad (4-1)$$

where IoU is the function to calculate the intersection-over-union between two rectangles, S_i and B_j are the labeled parking space i and the bounding box of detection j, and p_j is the detection probability of detection j. Note that only detections with the category being a vehicle (e.g., car, van, bus, truck) will be kept in the detection list. Since the probability is between 0 and 1, we multiply the IoU by the square root of the detection probability rather than the original probability in order to give more weight to the term $IoU(S_i, B_j)$, which should be the primary indicator of parking occupancy status than the probability.

Considering that parking occupancy status does not change very often, the status in the immediate previous time step is another indicator of the current status. Hence, a double thresholding method is adopted to filter out invalid V_{ij} with two thresholds Th_{max} and Th_{min} ($Th_{max} > Th_{min}$). If space i is occupied in the previous time step, the threshold for V_{ij} will be Th_{min}; otherwise, if space i is vacant in the previous time step, the threshold for V_{ij} will be Th_{max}.

There are two cases that need further consideration: (1) one detection corresponding to multiple spaces and (2) one space corresponding to multiple detections. We deal with the first case first. Since one detection can only match one space at most, in the first case, the space with the largest matching score will be identified as occupied and others vacant. These should address part, if not all, of case 2. Then, if there are still case 2 for any space, its status is occupied.

4.7.2 *Modified SORT and BG-Based Occupancy Detection*

The detections from background modeling at the edge are inputs to the BG-based occupancy detection algorithm on the server. The video's temporal information is used in this module in the way of object tracking. Object tracking eliminates false detections and noises in the BG detection step and generates tracks of objects. Since our system only has the bounding boxes' location

information transmitted back, the object tracking algorithm is supposed to use no more information than the boxes' locations. Tracking algorithms that require LiDAR, radar, or other image information (histogram, color, deep feature, etc.) would not work for our system (Ruimin Ke et al. 2018; Muresan and Nedevschi 2018; Wojke, Bewley, and Paulus 2017).

A state-of-the-art tracking algorithm, called SORT (Bewley et al. 2016), achieves excellent performance on efficiency and accuracy using only bounding box location information. The proposed tracking algorithm is a modified version of the algorithm. The original SORT does not have a re-identification process, which will lose track of an object if not detected for a few frames. In the BG-based detection method, only moving objects are detected. Thus, in parking lots, a vehicle is often lost with an ID switch when it stops to change direction (see Figure 4-4). This is also the motivation for Deep SORT, which adds a re-identification metric using deep association (Wojke, Bewley, and Paulus 2017). In our system, the Deep SORT is not possible to incorporate because it requires deep features. Hence, we add a simple yet efficient decision rule to SORT: when a new ID is assigned to an object, the algorithm searches if the new object's bounding box has enough overlap (IoU) with any old object within the past m seconds. An old object is defined as an object that was tracked in the past. If yes, the two objects are associated.

With objects' tracks and the labeled parking spaces, parking occupancy can be detected: if a track starts from inside a parking space and ends outside the space, and the tracked time of the object is over a threshold (t_track seconds), the space's status is vacant; if a track starts from outside any parking spaces and ends inside a space, and the tracked time is over a threshold (t_track seconds), this space's status is occupied.

4.7.3 *Final Detection Considering Occlusion and Extreme Lighting Condition*

The final detection considering occlusion and extreme lighting condition further improve the system accuracy in extreme cases. In the proposed system, the SSD-based method is the primary detector. The BG-based detector serves as the compensation for SSD in corner cases like occlusion and extreme lighting conditions. In normal condition, the proposed SSD-based method performs near-perfectly; however, in extreme lighting conditions such as strong fog, direct sunshine, and strong shadow, SSD or any pattern-based detector, especially when there is no following object tracking process, could have poor performance and sometimes even miss most targets. On the other hand, the BG method is relatively more stable in extreme conditions, though not as good as the enhanced SSD in normal conditions.

If extreme lighting condition warning is triggered, the two sets of results will be combined. For those spaces detected as occupied in SSD detection, their final statuses are occupied given the low false-positive rate of SSD; for spaces recognized as vacant by SSD, the system under warning will believe the BG detection results. We determine if the lighting condition is bad enough to activate the combined detection using a metric as follows,

$$r_t = \frac{ssd_t}{bg_t} + \frac{ssd_t}{ssd_{t-1}} \qquad (4-2)$$

where bg_t is the number of occupied spaces at current time *t* from the BG method, ssd_t is the number of occupied spaces at current time *t* from the SSD method, and ssd_{t-1} is the number of occupied spaces at the time *t-1* from the SSD method. This metric measures the difference ratio for the two detection methods and the short-time change in the SSD-based method. Extreme lighting conditions change, such as direct sunshine, usually happens in a short time and have an

immediate influence on SSD. One time step here is set to five minutes based on the consideration that five minutes is short enough to ensure most space statuses are the same and long enough for a sudden lighting condition change to impact the SSD detector. The SSD-based method will be re-activated when $r_1 = \frac{ssd_t}{bg_t}$ returns from very small to close to 1.

Occlusion is often caused by a large vehicle's appearance and camera angle. In this study, we consider the case that one vehicle blocking two spaces, while other types of occlusion are even rarer. The system checks routinely if a bounding box of a vehicle covers two spaces. Here "cover" means two adjacent parking spaces are both at least $occ\%$ inside a vehicle's bounding box. In the occlusion case, the system first determines which space this vehicle is in by comparing the center of the two spaces, and the one closer to the camera (closer to the image bottom) is the space the vehicle in. For the other space, the system will use the BG-based results when it is occluded, because the object tracking will still give a clue which spaces a vehicle starts from or ends in.

Figure 4-4 BG-based detection and the original SORT tracking results. In a parking lot scene, the car at the bottom-left is lost with an ID switch due to its stop to change direction. Our modified SORT algorithm on the server solves this problem, thus reduce the error in parking occupancy detection.

4.8 EXPERIMENTAL RESULTS AND ANALYSIS

4.8.1 *Preliminary Test and Parameter Settings*

This study was sponsored by Sound Transit, which is a public transit agency serving the Seattle metropolitan area in the U.S. state of Washington. The preliminary test was conducted at the Smart Transportation Applications and Research Laboratory (STAR Lab) of the University of Washington. Before the field test, over five hours of parking lot surveillance video clips and over two thousand parking lot images were collected from the internet and the Angle Lake parking garage for preliminary test and system parameter setting. Note that the Angle Lake parking garage is the test site of this study, which is a busy parking garage located near the Sea-Tac International Airport with 1,160 available parking spaces. The parameter setting is critical to the operation of the system. Table I summaries the key parameters of the system and their setting for the field test in the study based on the preliminary lab test.

There are seven parameters that need to be set. The given parameter values in Table 4-1 can be a reference for the general parking context. For some specific cases, these parameters may need to be adjusted for optimal system performance. $\mathrm{Th}_{\max}$ is always set larger than $\mathrm{Th}_{\min}$ according to their definition. A general rule for setting these two parameters is that if neighboring parking spaces have larger overlaps from the camera angle, the two parameters may need to be set larger to avoid false matching. We do not suggest modifying IoU_track or t_track in most cases. They are parameters for the tracking algorithm, which are not sensitive to the context. But they can also be adjusted based on the drivers' behaviors in a region or the user preference. The r_t and r_1 can be adjusted based on the number of parking spaces covered in the camera view and the weather conditions in a region. If a camera view includes quite a few parking spaces, say, more than 6, we do not recommend changing r_t and r_1 by much. But if a camera covers only 2 or 3 spaces, extra

efforts would be expected in the parameter setting because you may not be able to tell whether a missed detection is due to sudden lighting changes or other factors. The difference in weather conditions of different regions could also influence the setting of these two parameters. The last parameter $occ\%$ can be adjusted based on camera angles; the more the overlap of neighboring parking spaces, the larger $occ\%$ should be.

Table 4-1 Description and settings of the key system parameters

Parameter	Parameter Description	Parameter Value
$\mathrm{Th}_{\max}$	The larger threshold for the matching score V_{ij} in the SSD-based occupancy detection	0.25
$\mathrm{Th}_{\min}$	The smaller threshold for the matching score V_{ij} in the SSD-based occupancy detection	0.1
IoU_track	The intersection-over-union threshold to determine if a new object is associated with an old object in the modified SORT algorithm	0.6
t_track	The time threshold in the unit of seconds to determine if a new object is associated with an old object in the modified SORT algorithm	8
r_t	The threshold to trigger the bad environment lighting warning to the system	0.8
r_1	The threshold to re-activate the normal detection pipeline in the system	0.7

occ%	The portion of a space inside a vehicle bounding box to determine if the space is covered by the vehicle in camera view in the occlusion judgement	90%

4.8.2 *System Installation and Data Collection*

Two IoT devices were installed, one on the sixth floor and another on the third floor of the Angle Lake parking garage. Figure 4-5 shows the installation of the IoT device on the sixth floor, the data server set up at the STAR Lab, and camera views from the two cameras. The sixth floor was an outdoor parking scene and the third floor was indoor. In the field test, our cameras monitored sixteen parking spaces, which were No.1013 – 1022 on the sixth floor and No.503 – 508 on the third floor. The purpose of choosing the sixth floor was to test the system performance outdoor, particularly how it performed in different weather, temperature, and time of day. The third floor was selected to test the indoor performance. The low ceiling height of this floor and the installation angle of the camera created challenges such as occlusion, which was meaningful for testing the system. The system was operating for three months from September 16, 2018, to December 15, 2018, at the Angle Lake parking garage. In total, only less than 20Gb data was transmitted back onto the STAR Lab server from the two IoT devices in three months.

Figure 4-5 System installation at the Angle Lake parking garage (top-left) and the server set up at STAR Lab (top-right); the bottom row displays the camera views of the devices we installed on the sixth floor (bottom-left) and the third floor (bottom-right).

4.8.3 *Results and System Evaluation*

Every ten minutes, a video frame was transmitted to the server for validation and demonstration. Overall, the system achieves 95.6% detection accuracy during the three months. Figure 4-6 presents the sample detection results and Table 4-2 shows the summarized statistics of the experiment. In this table, we divided the detection conditions into multiple categories based on the weather, time (day or night), day (weekday or weekends), and floor (indoor or outdoor). The categorization was done by manually classifying the scene pictures captured by the cameras as well as checking the historical weather records. And the detection performance for each category was summarized using accuracy as the metric ($accuracy = \frac{correct\ detection}{total\ number\ of\ spaces} \times 100\%$).

4.8.3.1 Cloudy and Rainy Scenarios

The weather was classified into four categories, which are sunny, rainy, cloudy, and foggy conditions. For the third floor (indoor), there were no significant accuracy differences across the four weather conditions, while the sixth floor (outdoor) was influenced more by the weather. Among the four weather conditions, the system performed the best in cloudy conditions, reaching 97.5% accuracy on weekdays and 99.2% on weekends, due to the relatively consistent lighting conditions over the video field of views (Figure 4-6(a)). The second highest was in rainy conditions with an accuracy of 93.7% on weekdays and 96.2% on weekends, where the lighting conditions were similar to cloudy days. However, raindrops on the lens might sometimes block parking spaces though a camera shelter was employed to protect the camera (see Figure 4-6(b)). This was rare but the main cause of its lower accuracy than cloudy days. In rainy and cloudy days, background-based occupancy detection was seldom activated.

4.8.3.2 Sunny and Foggy Scenarios

The detection accuracies in sunny conditions and foggy conditions were both lower than cloudy and rainy days. We carefully examined the ground-truth images and found out the reasons. In sunny conditions, there were sometimes strong shadows of the vehicles, and reflections towards the camera on the sixth floor. In our case, shadows and reflections could significantly change the visual appearances of the vehicles, thereby confusing the feature extraction processes, especially when the occupancy was high: since vehicles were very close to each other and the ten parking spaces were covered by just one camera from around 25 – 30 feet away, the shadow of the fence on the sixth floor and the reflection sometimes could influence multiple vehicles; also, the shadow of one vehicle could not only change the visual appearance of itself but also the vehicles next to it. Foggy conditions had the lowest detection accuracy for the outdoor parking with 85.7% on

weekdays and 91.6% on weekends. Our observation indicated that the thicker the fog was, the lower the detection accuracy was. In sunny and foggy conditions, background-based occupancy detection was activated more than rainy and cloudy conditions. They help compensate the SSD detection and significantly improve the accuracy following the proposed extreme condition detection pipeline. Though the overall accuracies of sunny and foggy days were still lower than the average, it was already increased a lot over standalone SSD-based occupancy detection. For example, in Figure 4-6(c) and (d), it can be seen in the extreme lighting conditions like direct sunshine and strong fog, even the enhanced SSD's performance significantly decreased with higher missed rates and lower detection probabilities.

4.8.3.3 Nighttime, Weekend, and Occlusion

It was interesting to note that the overall detection accuracy at night was higher than that during the day, and the detection performed better on weekends than on weekdays (see Figure 4-6(e)(f)(g)). These results were mainly caused by the property of the SSD detector. Our detector was finetuned to have a very low false-positive rate in order to achieve high precision. In other words, in case there was a false detection, it is more likely that an occupied space was recognized as a vacant space, rather than a vacant space being recognized as occupied. According to this fact, it was interpretable that the accuracy at night and weekend were overall higher than in the day and weekday, because the traffic volume and the number of occupied spaces were lower at night and over the weekend. Moreover, the enhanced SSD got good detection results in dark due to the large number of image training samples taken at night in the MIO-TCD dataset.

It was observed that the detection performance was more consistent on the third floor with a lower variance than the sixth floor in different weather conditions. Also, the main causes of errors in detection for the two floors were actually different. It was found that most errors on the sixth

floor were caused by extreme lighting conditions (shadow, reflection, and fog), while on the third floor the errors were caused more by occlusion. For spaces 503, 504, and 505, the detection accuracy was almost 100% for all scenarios. However, due to the installation angle of the camera and the low ceiling height, spaces 506, 507, 508 could be partially or fully blocked by the vehicles parking next to them. Figure 4-6(h) showed an example of a van parking in space 507 completely blocking space 508, but the occlusion case was handled by our proposed pipeline. Note that occlusion was dealt with by our system with (1) SSD on partially blocked vehicles, or (2) BG-based detection and tracking if an occlusion warning was triggered.

Table 4-2 System detection accuracy statistics

	Sunny	Rainy	Cloudy	Foggy	Day	Night	Average
Average	91.4%	93.5%	95.5%	89.9%	92.7%	98.4%	**95.6%**
On third Floor (Weekday)	92.3%	91.8%	92.6%	92.0%	92.2%	99.1%	95.7%
On third Floor (Weekend)	94.3%	94.5%	93.9%	93.1%	94.0%	99.0%	96.5%
On sixth Floor (Weekday)	88.5%	93.7%	97.5%	85.7%	91.7%	97.3%	94.5%
On sixth Floor (Weekend)	93.8%	96.2%	99.2%	91.6%	95.4%	98.9%	97.2%

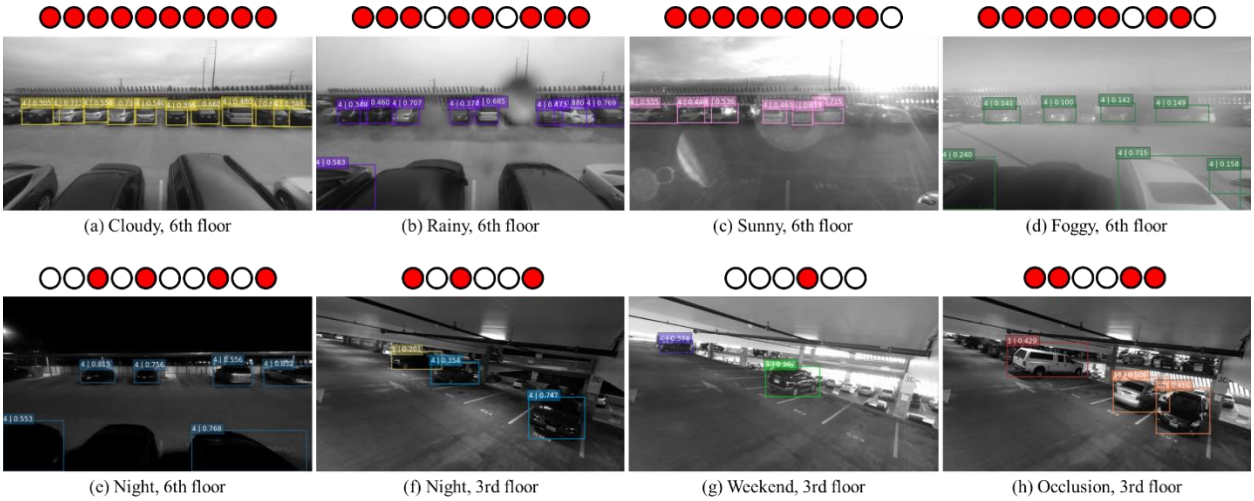


Figure 4-6 The figure displays final detection results (on the top of each image) and enhanced SSD-based detection results (in images) in representative scenarios. Extreme lighting condition or occlusion warning was triggered in scenario (c) (d) and (h), in which BG-based detection was activated.

4.8.4 *Comparison with State of the Arts*

This sub-chapter summarized the comparison between this study and the state of the arts in automatic parking surveillance in Table 4-3. First of all, the system inputs are different among different systems. Taking images as the input could be straightforward but will lose the temporal information. Systems that have IoT devices leverage the power for efficient transmission and onboard processing so that they enable real-time occupancy updates. Compared to previous studies which also use IoT devices, this study deploys the computations on both the edge and the server, which helps efficiently handle the workload; also, we implement a detection-based pipeline thereby does not need labeling on every IoT device. Please note that though previous systems including IoT devices do not have computation tasks on the server, they do need servers as part of the system for data storage. Regarding primary algorithms, this study and (Nieto et al. 2018) are among the first efforts to use the deep-learning-based object detectors for parking (Faster R-CNN and SSD). However, SSD is the latest one-stage object detector, which is faster than the two-stage

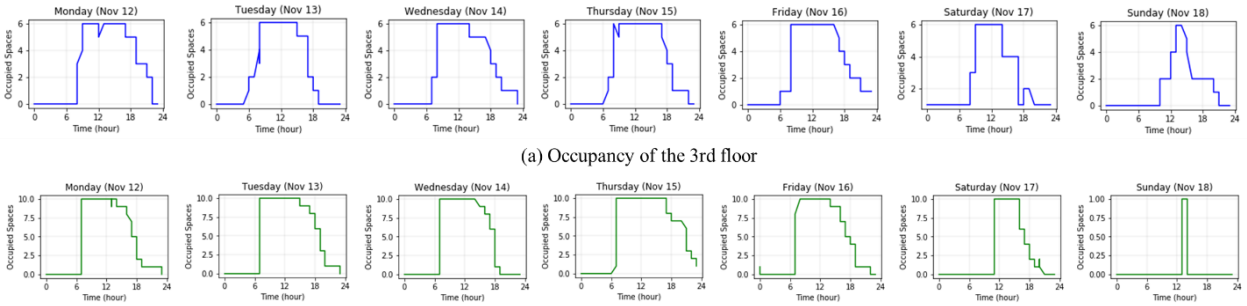
Faster R-CNN. Thus, it can better support edge computing. Our study also has the largest number of frame samples for training (127,125 frames). The work (Cho et al. 2018) has 390,000 image patches for training, where one patch is one parking space they cropped from the original frame. They do not mention how many frames they use. This study and (Bulan et al. 2013) are the two using real-world data for validation. While others using a few frames or images. Our validation covers a relatively long time (three months) and more scenarios. In terms of system efficiency, (Cho et al. 2018; Ling et al. 2017; Nieto et al. 2018) do not mention their processing speeds or efficiency measures in their papers since that is not their main focus. (Bulan et al. 2013) achieves 5 frames per second processing on their desktop with no artificial intelligence methods, but is still an impressive performance in 2013. (Amato et al. 2017) mentions their CNN can process 50 spaces in an image per 15 seconds (about 3 spaces per second). The proposed system achieves about 1 frame per second, which is faster than most existing systems (some not shown in the table). Updating a parking lot's occupancy status every one second is sufficient in most cases. The state of the arts all achieve great accuracy (over 90%). Due to the different inputs and designs of these systems and the lack of a widely accepted public parking video dataset (image datasets does not work for many systems), the system accuracy of each system cannot be directly compared at this time.

Table 4-3 Comparison between the proposed parking surveillance system and the state of the arts

Research work	**Bulan et al. 2013**	**Ling et al. 2017**	**Amato et al. 2017**	**Cho et al. 2018**	**Nieto et al. 2018**	This study
System input	Video	Video	Image	Image	Multiple videos	**Video**
Computation platform	Desktop	IoT devices	IoT devices	NA	Desktop	**IoT devices and server**
Process mode	Post analysis	Onboard processing	Onboard processing	Post analysis	Post analysis	**Onboard processing**
Pipeline logic	Detection	Classification	Classification	Classification	Detection	**Detection**
Primary algorithms	SVM, HOG, BG	Haar, F-test	CNN	Random forest	Faster R-CNN, fusion	**SSD, BG, SORT, fusion**
# of training frames	1,800	469	4,323	390,000 (patches)	23,741	**127,125**
Validation data	Several days real-world validation	90 detections	CNRPark + EXT image dataset	24,000 image patches	1,000 frames	**Three months real-world validation**
Testing scenarios	Outdoor, sunny, cloudy, rainy, daytime, occlusion	Outdoor, daytime	Outdoor, sunny, cloudy, rainy, daytime	Indoor	Outdoor, clear, rainy, daytime, nighttime	**Outdoor, indoor, occlusion, sunny, cloudy, rainy, foggy, daytime, nighttime**
System efficiency	5 frames per second	NA	3 spaces per second	NA	NA	**1 frame per second**
System accuracy	93.9%	91%	> 90%	98.6%	> 90%	**95.6%**

4.9 System and Data Applicability

A sample occupancy dataset collected by the system is shown in Figure 4-7. It was in the week Nov 12 – Nov 18, 2018. The plots give an intuition on the parking occupancy patterns in the garage. The proposed system and the real-time parking occupancy data generated by the system can be valuable resources to support a variety of intelligent transportation applications, such as smart parking management, advanced infrastructure systems, and connected and automated vehicles.



(a) Occupancy of the 3rd floor

(a) Occupancy of the 6th floor

Figure 4-7 The parking occupancy patterns of the week Nov 12 – Nov 18, 2018.

One of the key components in intelligent transportation systems is smart parking management. Smart parking management targets improving the efficiency in parking resource allocation, parking information dissemination, and parking space searching with high accuracy, robustness, and low cost. Smart parking management also has the ability or potential to help mitigate traffic congestion and other societal problems. Parking prediction, dynamic parking pricing, real-time parking guidance, etc., most smart parking management strategies and functions need parking occupancy data as the input. Thus, the proposed system, which is designed to work in a wide range

of scenarios with low cost and high performance, provides foundations to smart parking applications as well as modern transportation management.

Future advanced infrastructure systems will solve problems related to buildings, bridges, pipelines, roadways etc. by combining conventional physical assets with emerging cyber technologies in computer science, system engineering, and other fields. Edge computing will be a critical component of infrastructure management of tomorrow, especially with the emergence of 5G communication. A cost-effective, real-time, reliable, and scalable edge computing system for parking occupancy detection will offer new solutions and opportunities to smart city developments by making infrastructures like buildings and roadways smarter, more efficient, and more sustainable.

Parking occupancy data will be an essential component in connected and automated vehicle applications. First of all, automated vehicles will need to find parking themselves, which will be completed faster with their own systems communicating with nearby parking facilities. Additionally, parking facilities will serve as crucial nodes in a traffic roadway network to support V2I functions. IoT devices monitoring parking spaces will indirectly obtain the traffic conditions nearby from parking occupancy, thus help network-wide decision making in the era of connected and automated vehicles.

Chapter 5. MULTI-TASK SYSTEM FOR REAL-TIME TRAFFIC AND ROAD MONITORING WITH EDGE AI

5.1 Overview and Contribution

Parking surveillance and roadway traffic surveillance are two major tasks in traffic surveillance. In Chapter 4, a parking surveillance system is introduced. In this chapter, we focus on roadway traffic surveillance by introducing the design, development, and implementation of a real-time multi-task roadway and traffic surveillance system with AI methods.

The three key problems this system tries to address are: 1) Robust roadway mask detection and direction classification in challenging scenarios (e.g., low lighting conditions); 2) roadway surface condition (rainy, dry, snowy) classification on the edge using low-cost features from videos and environment sensors; 3) traffic surveillance, including traffic volume counting and classification, operated in real-time on the edge with AI classifier.

This system operates on the recently released edge device Raspberry Pi 4. Instead of splitting computational loads onto the edge and the cloud, this system operates mostly on the edge. The real-time operation is achieved with a proposed multi-thread system architecture with several new methods for different threads. This system achieves multi-task traffic surveillance including road detection, road surface condition classification, traffic volume counting, and traffic classification.

In addition to the key system functions, we introduce an on-site camera calibration method using standard shipping containers. We prove that the camera calibration model is solvable using least square method with the shape of standard shipping containers through mathematical derivations. This calibration method tends to support the 2D to 3D reconstruction in roadway surveillance tasks, such as estimating vehicle speed and location in surveillance videos.

Specifically, the contributions of this chapter are listed below.

- A multi-thread system architecture designed to run on the edge device in real-time for multi-task roadway traffic surveillance.
- An efficient and robust motion-vector clustering method for road mask extraction and road direction classification.
- An AI trigger method aiming to support real-time edge computing that integrates traditional object detection algorithm and AI classifier for traffic flow detection, vehicle counting, and vehicle classification.
- A road surface condition classification method using video image features and environment features and machine learning models.
- An onsite camera calibration method using standard shipping containers, which is proved to be solvable and demonstrates effectiveness for 2D to 3D traffic information reconstruction.

5.2 System Architecture

The system architecture is shown in Figure 5-1. Unlike in chapter 4 to split the computation loads to the edge and cloud server, this architecture is designed to operate mostly on the edge device. There are three threads operating independently in this system: the main thread, the road mask extraction thread, and the traffic flow detection thread. The reason for designing three threads instead of one is that the road mask detection task and the traffic flow detection task have high computation complexity. This architecture design, coupled with the road mask extraction method and the AI trigger method, can address this issue, thus enabling real-time processing on the edge device.

In the main thread, background modeling and SORT tracking is implemented to generate foreground image and blobs for detecting and tracking the moving objects. At the same time, the

background image (without moving objects) is used for roadway surface condition classification (dry, rainy, snowy). The road mask received from the road mask extraction thread serves as a filter to keep solely the road pixels in the background image. Thus, moving traffic on the road and non-road regions in the camera view are filtered out in the road surface condition classification task for improved accuracy. Four features, including two image features (intensity and black channel) and two environmental features (temperature and humidity) are taken by a random forest model for the road condition classification.

The road mask extraction thread also takes the raw video stream as input. The intuition behind this new road mask extraction method is that human eyes can tell where the road regions are at night with the help of vehicle trajectories. Optical flow tracker extracts moving pixels at night in traffic videos with good accuracy. In this work, the optical flow tracker is applied to extract moving pixels to roughly get the vehicle trajectories, and then to cluster the pixels into road regions for different traffic directions with location and motion features and DBSCAN clustering. DBSCAN clustering is chosen since it determines the number of clusters and filter out outliers.

The third thread is for traffic flow counting and classification. We design an AI trigger method to select, store, and process just a small portion of the original video frames, which are the video frames of interest. Specifically, when moving objects are defined and tracked in pre-defined regions on the road, the frame will be stored in a queue waiting to be processed. The Mobilenet finetuned on the MIO-TCD traffic surveillance dataset is implemented to locate and classify road users in the pre-defined road region. With the tracked and classified objects, traffic volume data for each type of road users are obtained.

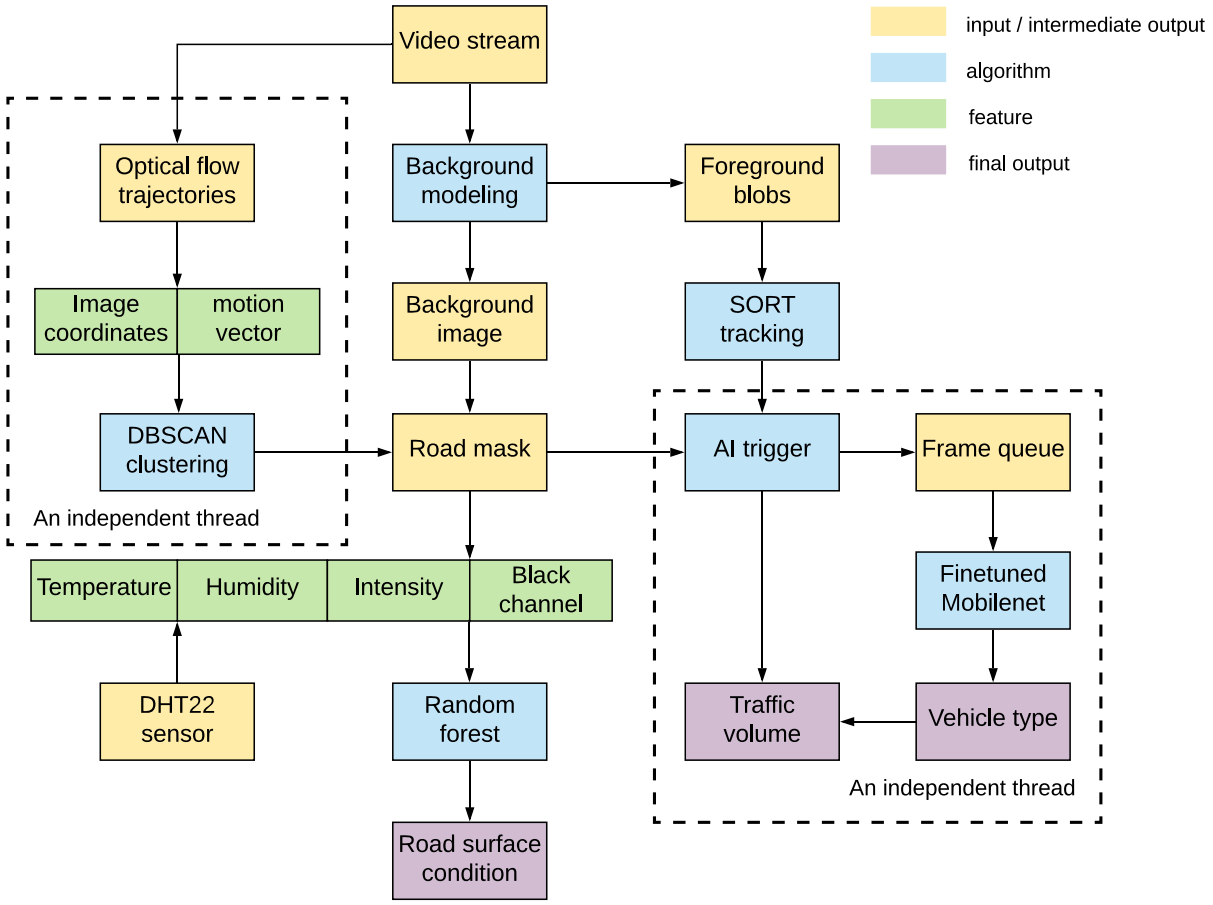


Figure 5-1 System architecture and the proposed methods on edge device.

5.3 Motion-Vector Clustering Method for Road Mask Extraction

One of the threads of the proposed edge computing system is for road mask extraction. It serves two purposes: 1) to help locate the road regions so that the image features (intensity and black channel) for the road surface condition classification are more representative; 2) to help identify the number of traffic directions and roughly locate the regions for each direction to assist traffic volume detection. The method targets efficiency and robustness in road mask extraction. We adopt clustering of motion vectors extracted from optical flow on the moving traffic, instead

of using appearance features of road itself. The intuition behind this method is an approximation of human eyes identifying road regions at night by watching moving traffic.

Two examples are displayed in Figure 5-2, one in the day and another at night. Optical flow is extracted in the first step of the method and filtered out by setting a lower limit in the moving distance between two consecutive frames. This step is to extract the interest points in the video frames and keep those from the moving traffic, since the background pixels don't move. We mark the pixels which have been covered by the filtered optical flow vectors in any frame with a feature vector $[x, y, d, v]$, where the four features denote pixel location *x*, pixel location *y*, moving direction *d*, and moving speed *v*. The marked pixels accumulate as time goes on, since more traffic will appear in the camera view. After some time, the marked pixels with the defined feature vectors will cover major areas where the traffic pass through. Note that there are often some false-positives (non-road regions being marked) and some false-negatives (road regions not marked).

The next step is to use DBSCAN clustering to extract the road regions and classify the traffic directions. Every pixel with the feature vectors is regarded as one point in the feature coordinate. We choose DBSCAN clustering over other popular clustering method such as k-means because of its ability to determine the number of clusters. Roadway has different structures. As shown in Figure 5-2, the two camera views cover different numbers of road branches, and it is not feasible to pre-define the number of road branches or traffic directions. Moreover, DBSCAN is able to filter out outliers which does not belong to any of the clusters, thereby auto-removing false-positives. However, the clustering cannot fill the "holes" on the road, which are the false-negatives (where the traffic trajectories did not cover). The closing operation in image morphology, as a typical method for filling the holes in an image, is applied to address this issue.

The example results are presented in the last row of Figure 5-2. In the first result, the extracted road mask has two different regions identified, which are the bi-directional traffic moving areas in this camera view. The second extracted mask contains four areas, where two of them are the mainlines and another two are the ramps. Also, it can be seen that the second result is at night, where the environment is dark, and the roadway regions are very similar to non-road regions from the appearance. In this case, the proposed method still has excellent performance in the extraction and identification.

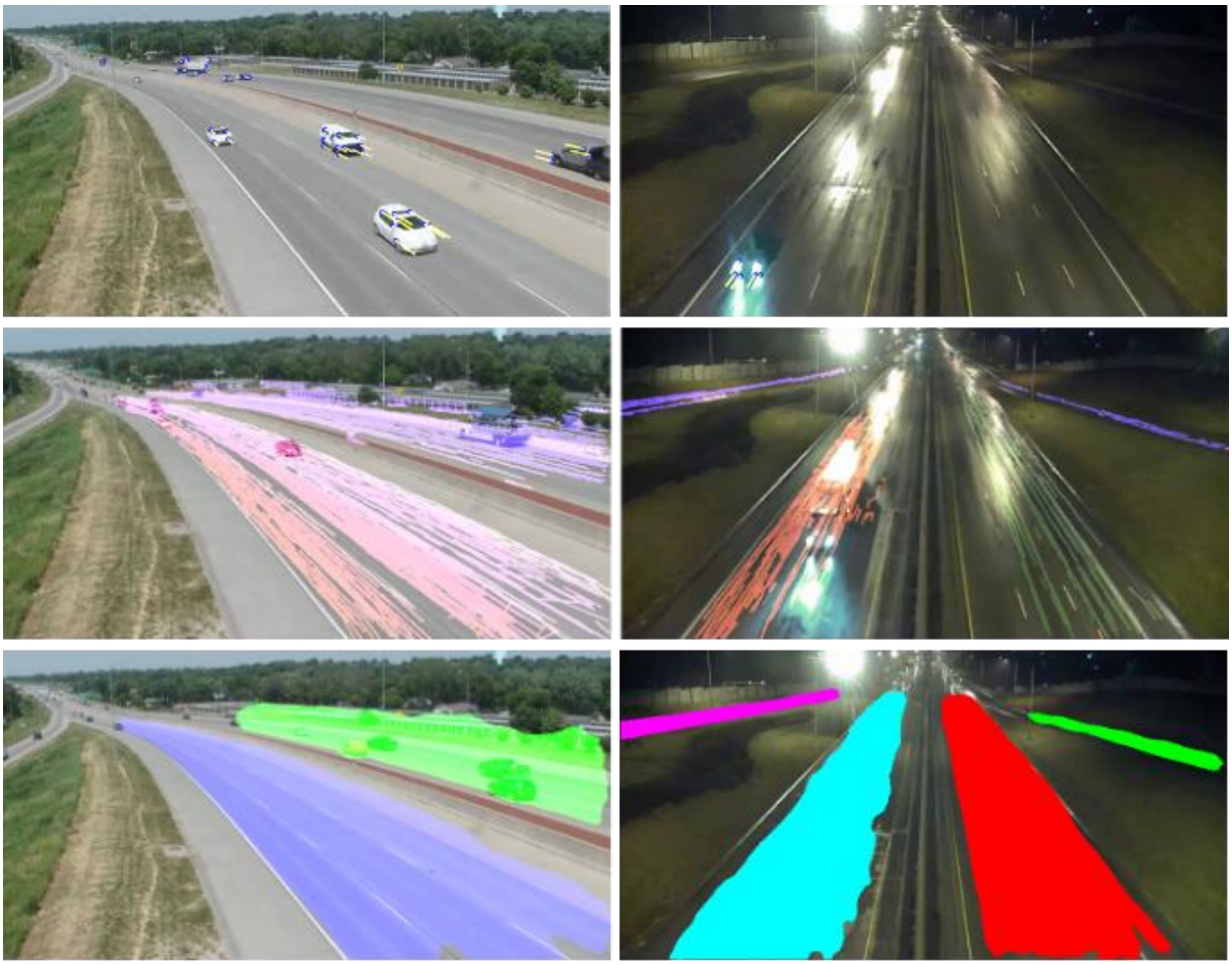


Figure 5-2 Motion-vector clustering method to extract road mask and identify traffic directions.

5.4 Road Surface Condition Classification based on Image and Environmental Features

5.4.1 *Background*

Road surface condition classification is of critical importance for traffic monitoring because it influences road capacity, travel time reliability, and traffic safety. Traffic flow speed can drop by up to 40% and capacity can drop by up to 27% on freeways in rainy, snowy, or foggy conditions. Regarding traffic safety, over 20% traffic crashes from 2007 to 2016 happened in adverse weather or on slick road surface. Hence, being able to detect hazardous conditions of road surface and issue immediate warnings are crucial for traffic management and safety. One of the objectives of the system is to classify road surface conditions into rainy, dry, and snowy using image and environmental features in real-time on edge devices. This function is realized in the main thread as shown in Figure 5-1.

5.4.2 *Methodology*

We selected four features, where two are from the image and two are from the environment, for the classification. The two image features are the median intensity value and median dark channel value. The intensity value is selected because it can differentiate snowy conditions and non-snowy conditions, since the intensity values are high in snowy conditions where the color are white in many places. The dark channel value is defined as the minimum value among R, G, B channels for a given pixel. For most objects in the nature, the dark channel value is pretty small, otherwise, the object will be in white. This is also the case for most road surfaces. However, for rainy or snowy conditions where there are a lot of reflections or white pixels, the dark channel can be relatively higher. Therefore, it is selected as a feature from the image. Two environmental

features, temperature and humidity, collected from the DHT22 sensor, are taken as features as well. Temperature is a strong indicator of snowy conditions and humidity is a strong indicator of rainy conditions. The feature we selected are representative and simple, which form a feature space as a basis for successful classification. The feature vector contains just four values, [I,K,T,H], where I is median intensity, K is median dark channel value, T is temperature, and H is humidity. Figure 5-3 shows an example of intensity histogram of the road regions in the view of a surveillance camera in Washington State. Although it is at nighttime, the intensity features of the road with snow and without snow are still very different and can be used for classification in our method. Temperature and humidity sensors will further increase the accuracy and reliability of the classification.

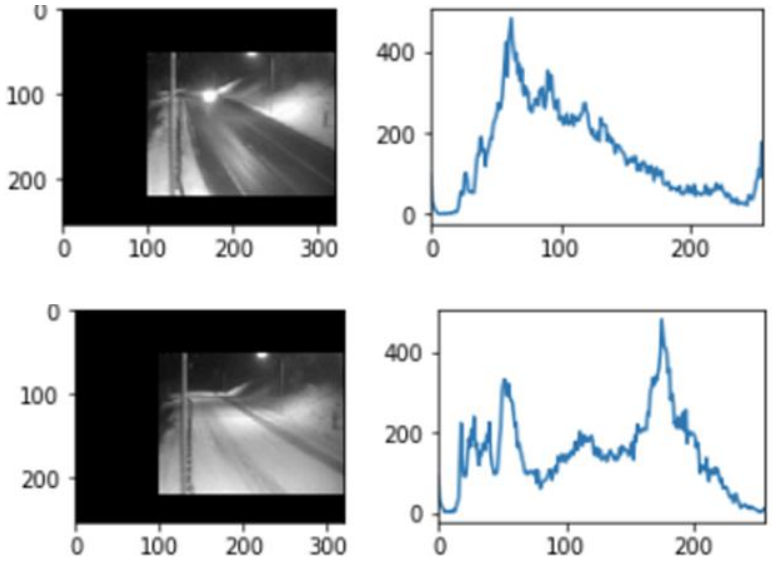


Figure 5-3 Intensity histograms of a road without snow (top) and with snow (bottom).

To generate the two image features, it takes three steps. First, the background image is generated from the background modeling method, where the moving traffic is removed and only the background road surface and other non-moving objects are kept. Then, the extracted road mask

is applied to the background image to keep only the major portion of the road surface pixels. These two steps are necessary in order to keep the input features more accurate by getting rid of non-road pixels (traffic on the road surface and other objects outside the road surface). Last, the median values of the intensity and the dark channel will be extracted as the two input features.

Regarding the classification model, we target several representative machine learning models, including K-nearest neighbors (KNN), Naïve Bayesian (NB), Support Vector Machine (SVM), Random Forest (RF), and Artificial Neural Network (ANN). We consider these models are representative and diversified. Given our classification task and generated features, these models are complex enough to do a good job and simple enough to operate in real-time on edge devices. We will compare their classification performance and select one among them as the final classifier.

5.4.3 *Data Collection*

Data collection is done in Washington State with the public API from the Washington State Department of Transportation (WSDOT). There are hundreds of surveillance cameras and weather stations available in the API. We wrote a crawler program to download the camera images and weather data. The surveillance camera distribution and weather station distribution can be seen in Figure 5-4 and Figure 5-5. Among these cameras and stations, we selected 28 locations with both the camera and weather data available. For every three minutes, the crawler store and combine the two data sources. The 28 locations are shown in Figure 5-6. Some sample image data are shown in Figure 5-7. The data collection period includes February 2020, March 2020, April 2020, October 2020, and November 2020. The summer time is excluded because there are no snowy roads during that time in Washington State. We have collected over 10,000 images and associated temperature and humidity data for training and evaluation of the classifiers.

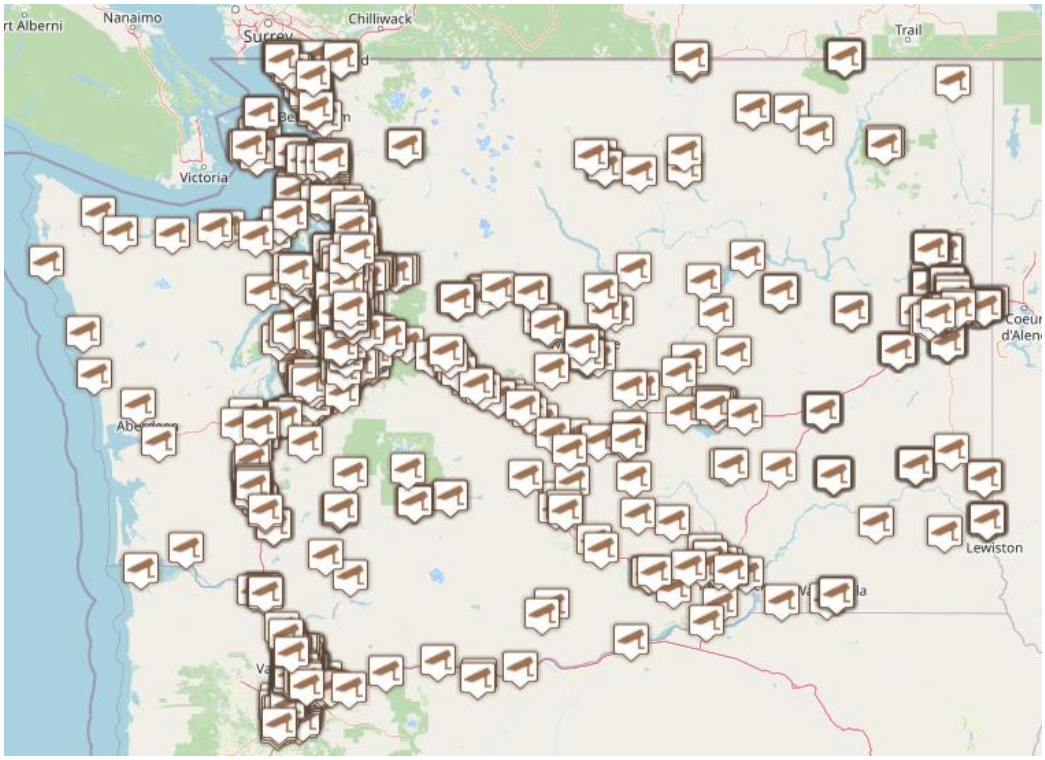


Figure 5-4 WSDOT surveillance camera distribution.

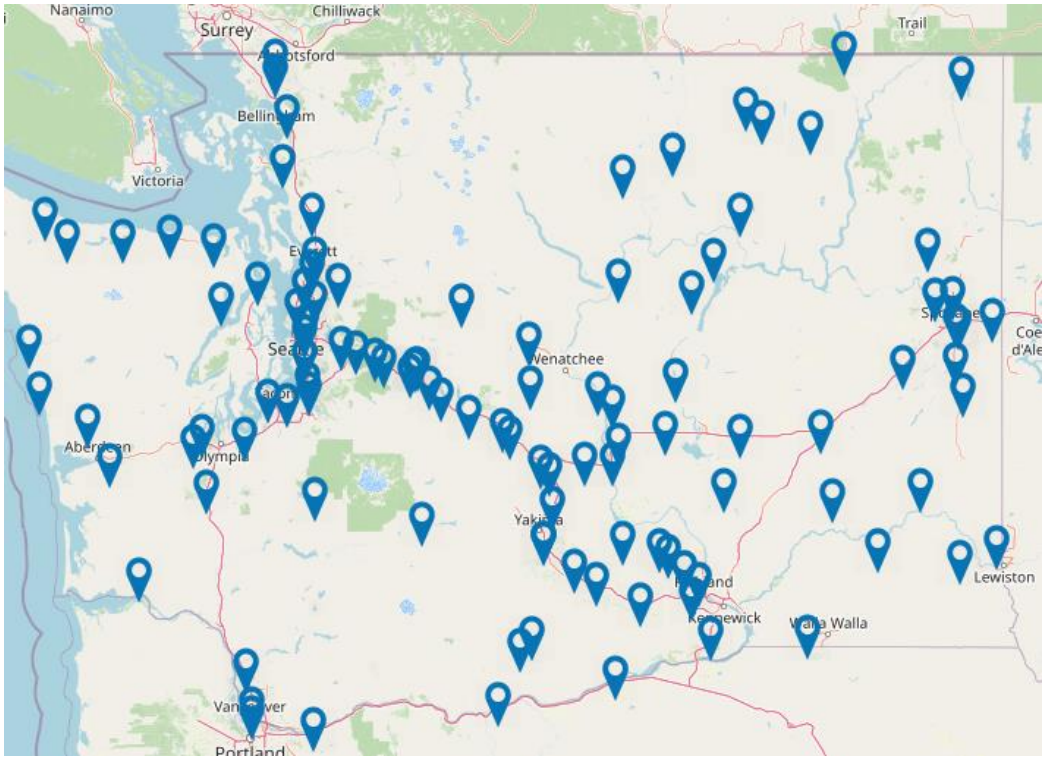


Figure 5-5 Washington weather station distribution.

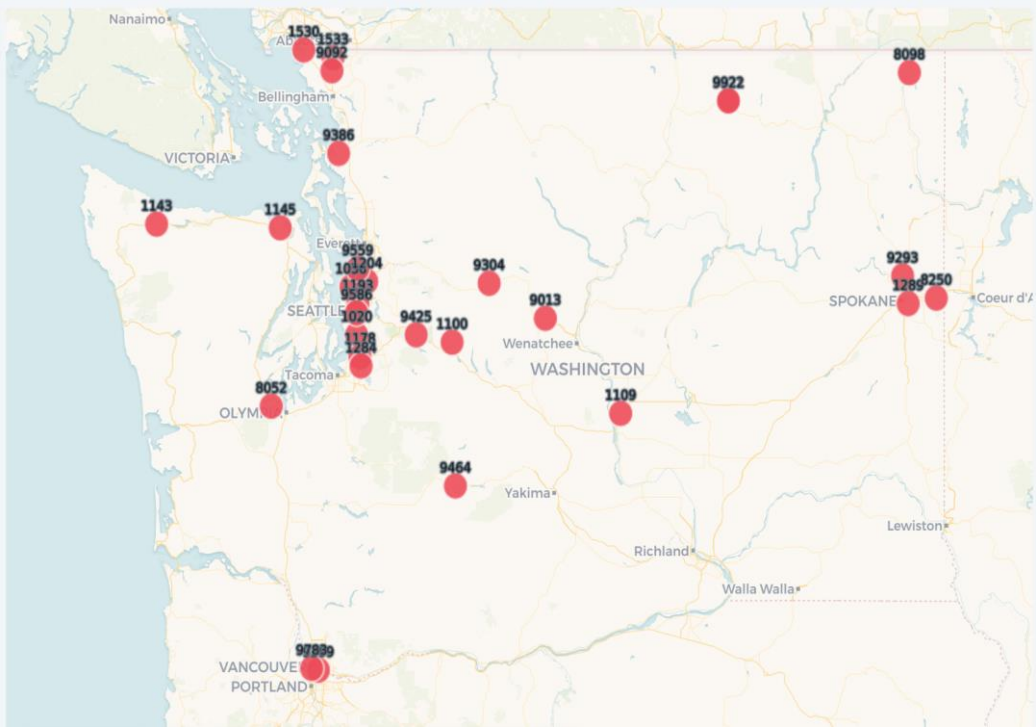


Figure 5-6 The selected locations in Washington State for training and testing the road surface condition classification method.

Figure 5-7 Surveillance camera image samples for rainy, snowy, and dry road surfaces.

5.5 AI Trigger Method for Traffic Flow Detection and Classification on Edge Device

Traffic detection and classification using surveillance cameras have been popular topics in intelligent transportation systems. However, new challenges need to be addressed if running on edge devices with the latest AI methods and for integrated multi-task processing. Limited

computation power and network access make real-time operation not easy to be achieved with traditional designs. We address this challenge from both the system level and the algorithm level.

The intuition underneath this design is that in most video frames there is no traffic in certain regions and we can selectively ignore many of the frames, thus to improve processing efficiency while keep the necessary traffic information from the video. Traditional background modeling can detect moving objects in real-time on Raspberry Pi 4, therefore, it can keep the temporal information from the surveillance video. Keeping temporal information is important because it makes object tracking possible to avoid counting the same road user multiple time. The deep neural network classifiers have good performances in object classification but are less efficient. Our design has the traditional detection and tracking in the main thread to utilize the temporal information and the AI classifier in another thread for further processing of the selected frames with traffic in the region of interest.

The AI trigger method is shown in Table 5-1. First of all, we need to define a region as the region of interest, which triggers further processing using the AI classifier when there are moving objects. This can be done by either manually labelling one area for each traffic direction or using the extracted road mask in the previous step as the region of interest. Then, the background modeling and SORT tracking runs in real-time in the main thread to process every video frame. In case there is any object detected and tracked in the region of interest, it will activate the procedure that extracts and stores the intermediate information into a global variable queue Q.

Each element in the Q has three major elements: the current time T, the current frame F, and a list L that contains objects' information in the current frame. Note that the current time T is needed as a time label because the frame may be processed later, depending on how many elements are left in Q. The list can be any length depending on the number of moving objects in the region

of interest. There are three sub-components for each object in L: the bounding box bb, the tracking id tid, and the direction id of the road area d. Note that the bb and tid are obtained from the background modeling and SORT tracking.

Table 5-1 The AI trigger method

Input: Tracked objects *objs* in the current frame, regions of interest roi_d, global variable queue *Q*
Output: traffic volume counts $count_{k,d}$ for each category *k* and direction *d*

(in the main thread)
for *obj* in *objs*:
  for *roi* in roi_d:
    if *obj*'s center is in *roi*:
      - Store 1) *obj*'s bounding box *bb*
        2) *obj*'s tracking id *tid*
        3) *obj*'s in the direction with id *d*
        into a list *obj_list* as a three-element tuple

      - Mark *AI_trigger* as True

if *AI_trigger* is True:
  - Append current time *T* as the 1st element into a list *L*
  - Append the current frame *F* as the 2nd element (a 2D array) into *L*
  - Append *obj_list* as the 3rd element into *L*
- Push *L* into queue *Q*

(in the traffic volume detection thread)
if *Q* is not empty:
  - Pop the first element in *Q* as *q*
  for each bounding box *bb*:
    - Crop the image at *bb* in frame *F* as *crop_img*
    - Run the AI classifier on the cropped image *crop_img*
    if *crop_img* is classified as road user type *k* & the *tid* is new:
      - $count_{k,d} = count_{k,d} + 1$
  - Update traffic volume information $count_{k,d}$ at time *T*

In another thread for the traffic volume detection and classification, it constantly checks if the queue Q is empty. If Q is not empty, it will pop the first element from Q, then it will run the Mobilenet classifier for road user classification and use other information we store in Q for traffic volume counting, for each type of road user and each traffic direction. Note that the moving object

may not be any road users we are interested, e.g., an animal, or just some false detections due to sudden light change of camera vibration. The AI classifier is able to filter out those detections in addition to classifying road users into different types.

For the finetuning of the Mobilenet classifier, we again use MIO-TCD for the transfer learning, which was mentioned in the last chapter. However, instead of using the detection dataset, we use the classification dataset for the learning. This dataset contains 648,959 images divided into 11 categories, including articulated truck, bicycle, bus, car, motorcycle, non-motorized vehicle, pedestrian, pickup truck, single unit truck, work van, and background. These images are all from surveillance video footage and are labeled. It is a great dataset for transfer learning for traffic classification tasks in surveillance videos.

5.6 ONSITE CAMERA CALIBRATION USING STANDARD SHIPPING CONTAINER

This section proposes an onsite camera calibration method for roadway surveillance camera, which can be broadly applied to assist traffic sensing. This method can extend the proposed multi-task monitoring system's capability (e.g., traffic speed detection) by supporting onsite calibration of camera parameters using standard shipping containers.

The proposed camera calibration method uses standard shipping container to calibrate roadway surveillance cameras. Standard shipping container is a rectangle of dimension (height × width × length) that must fall in one of four standard sizes: 8.5'× 8.0' × 20', 8.5'× 8.0' × 40', 9.5'× 8.0' × 40', or 9.5'× 8.0' × 45'. Let us denote its dimension height × width × length = h × w × l, as shown in Figure 5-8. The distance between the underside of a shipping container and ground surface is assumed to be d, which is often known for specific truck types. In addition, the origin O is assumed to be under P_1, with OP_1 = d. The X_w axis is parallel to P_1P_4, Y_w is parallel to P_1P_5, Z_w is parallel to P_1P_2. Thus, the 3D coordinates of P_1, P_2, …P_7 are shown in Figure 5-8.

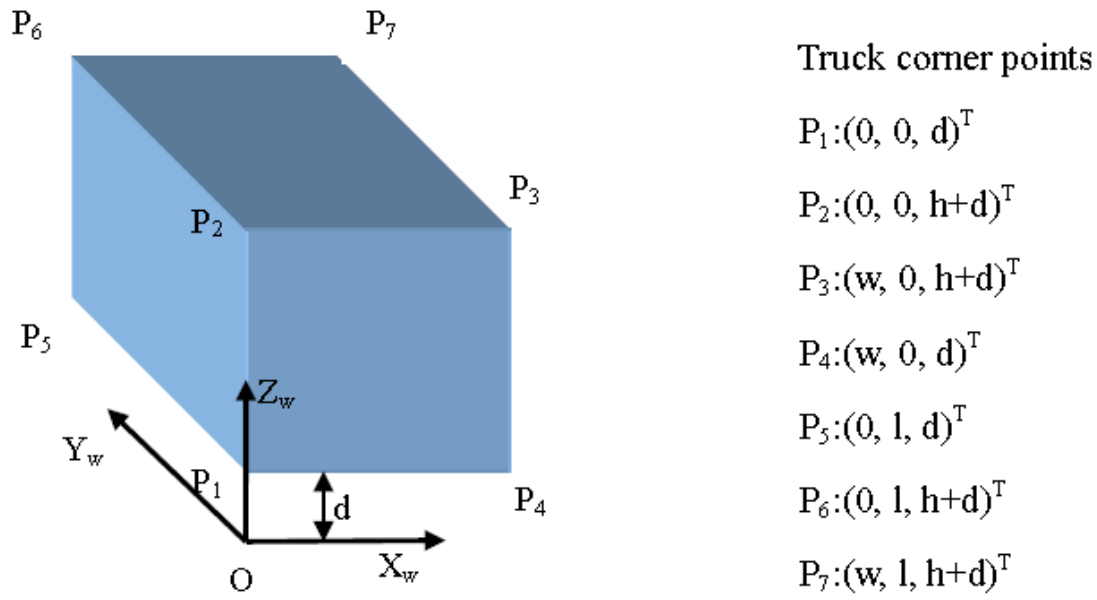


Figure 5-8 A model of standard moving object (international standard shipping container).

5.6.1 *Model Derivation*

There are basically three steps for the model derivation.

(1) From real-world coordinates to camera coordinates:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} \\ r_{yx} & r_{yy} & r_{yz} \\ r_{zx} & r_{zy} & r_{zz} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} \qquad (5-1)$$

$$X_c = r_{xx}X_w + r_{xy}Y_w + r_{xz}Z_w + T_x \qquad (5-1\text{a})$$

$$Y_c = r_{yx}X_w + r_{yy}Y_w + r_{yz}Z_w + T_y \qquad (5-1b)$$

$$Z_c = r_{zx}X_w + r_{zy}Y_w + r_{zz}Z_w + T_z \qquad (5-1c)$$

(2) From camera coordinates to image coordinates:

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \frac{f}{Z_c} \begin{bmatrix} X_c \\ Y_c \end{bmatrix} \qquad (5-2)$$

$$X = f \cdot \frac{r_{xx}X_w + r_{xy}Y_w + r_{xz}Z_w + T_x}{r_{zx}X_w + r_{zy}Y_w + r_{zz}Z_w + T_z} \qquad (5-2a)$$

$$Y = f \cdot \frac{r_{yx}X_w + r_{yy}Y_w + r_{yz}Z_w + T_y}{r_{zx}X_w + r_{zy}Y_w + r_{zz}Z_w + T_z} \qquad (5-2b)$$

(3) From image coordinates to pixel numbers:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} s_x \cdot X + u_o \\ s_y \cdot Y + v_o \end{bmatrix} \qquad (5-3)$$

$$s_x X = u - u_0 \qquad (5-3a)$$

$$Y = v - v_0 \qquad (5-3b)$$

For each corner point P_i, with X_{wi}, Y_{wi}, Z_{wi}, u_i, v_i, u_0, v_0 known, we can get $s_x X_i$ and Y_i from Eq. (5-3a) and (5-3b). Then, relationships between $s_x X_i, Y_i$ and X_{wi}, Y_{wi}, Z_{wi} can be established as follows:

$$s_x X_i = s_x f \cdot \frac{r_{xx}X_{wi} + r_{xy}Y_{wi} + r_{xz}Z_{wi} + T_x}{r_{zx}X_{wi} + r_{zy}Y_{wi} + r_{zz}Z_{wi} + T_z} \qquad (5-4a)$$

$$Y_i = f \cdot \frac{r_{yx}X_{wi} + r_{yy}Y_{wi} + r_{yz}Z_{wi} + T_y}{r_{zx}X_{wi} + r_{zy}Y_{wi} + r_{zz}Z_{wi} + T_z} \qquad (5-4b)$$

Dividing Eq. (5-4a) with (5-4b) yields:

$$\frac{s_x X_i}{Y_i} = s_x \frac{r_{xx}X_{wi} + r_{xy}Y_{wi} + r_{xz}Z_{wi} + T_x}{r_{yx}X_{wi} + r_{yy}Y_{wi} + r_{yz}Z_{wi} + T_y} \qquad (5-5)$$

Assuming $X_i' = s_x X_i,\ Y_i' = Y_i$, Eq. (5-5) can be expressed as:

$$\frac{X_i'}{Y_i'} = s_x \frac{r_{xx}X_{wi} + r_{xy}Y_{wi} + r_{xz}Z_{wi} + T_x}{r_{yx}X_{wi} + r_{yy}Y_{wi} + r_{yz}Z_{wi} + T_y} \qquad (5-6)$$

The origin of the object coordinate planes is on the ground. T_y is set to be nonzero by putting the origin of the object world coordinate away from the Y_c axis of the camera coordinate system. Thus, rearranging Eq. (5-6) yields:

$$Y_i' X_{wi} \frac{s_x r_{xx}}{T_y} + Y_i' Y_{wi} \frac{s_x r_{xy}}{T_y} + Y_i' Z_{wi} \frac{s_x r_{xz}}{T_y} + Y_i' \frac{s_x T_x}{T_y} - X_i' X_{wi} \frac{r_{yx}}{T_y} - X_i' Y_{wi} \frac{r_{yy}}{T_y} - X_i' Z_{wi} \frac{r_{yz}}{T_y} = X_i' \qquad (5-7)$$

The seven unknowns in Eq. (5-7) are $\frac{s_x r_{xx}}{T_y}$, $\frac{s_x r_{xy}}{T_y}$, $\frac{s_x r_{xz}}{T_y}$, $\frac{s_x T_x}{T_y}$, $\frac{r_{yx}}{T_y}$, $\frac{r_{yy}}{T_y}$, and $\frac{r_{yz}}{T_y}$.

The establishment of seven equations from the seven non-collinear points gives us a platform on which to solve these unknowns.

According to Eq. (5-7), for each calibration point i with (X_{wi}, Y_{wi}, Z_{wi}) as the 3D world coordinate and (X_i', Y_i') as the image coordinate, we can set up the following linear equation:

$$\left[Y_i' X_{wi}, Y_i' Y_{wi}, Y_i' Z_{wi}, Y_i', -X_i' X_{wi}, -X_i' Y_{wi}, -X_i' Z_{wi}\right] \begin{bmatrix} \frac{s_x r_{xx}}{T_y} \\ \frac{s_x r_{xy}}{T_y} \\ \frac{s_x r_{xz}}{T_y} \\ \frac{s_x T_x}{T_y} \\ \frac{r_{yx}}{T_y} \\ \frac{r_{yy}}{T_y} \\ \frac{r_{yz}}{T_y} \end{bmatrix} = X_i' \qquad (5-8)$$

Then, when we apply this equation to a shipping container, a linear equation set can be set up as follows:

$$\begin{bmatrix} 0,0, dY_1', Y_1', 0,0, -dX_1' \\ 0,0, (h+d)Y_2', Y_2', 0,0, -(h+d)X_2' \\ wY_3', 0, (h+d)Y_3', Y_3', -wX_3', 0, -(h+d)X_3' \\ wY_4', 0, dY_4', Y_4', -wX_4', 0, -dX_4' \\ 0, lY_5', dY_5', Y_5', 0, -lX_5', -dX_5' \\ 0, lY_6', (h+d)Y_6', Y_6', 0, -lX_6', -(h+d)X_6' \\ wY_7', lY_7', (h+d)Y_7', Y_7', -wX_7', -lX_7', -(h+d)X_7' \end{bmatrix} \begin{bmatrix} \frac{s_x r_{xx}}{T_y} \\ \frac{s_x r_{xy}}{T_y} \\ \frac{s_x r_{xz}}{T_y} \\ \frac{s_x T_x}{T_y} \\ \frac{r_{yx}}{T_y} \\ \frac{r_{yy}}{T_y} \\ \frac{r_{yz}}{T_y} \end{bmatrix} = \begin{bmatrix} X_1' \\ X_2' \\ X_3' \\ X_4' \\ X_5' \\ X_6' \\ X_7' \end{bmatrix} \qquad (5-9)$$

To solve Eq. (5-9), we have to prove that the coefficient matrix in this equation has full rank.

5.6.2 *Proof for the Linear Independence of the Rows of the Coefficient Matrix in Eq. (5-9)*

Firstly, we should emphasize that among the seven corner points, no three points are collinear on the image plane. During an object's motion through the view range of a video camera, there are a large number of snapshots can be collected in different frames. In order to get an available snapshot for calibration, it is reasonable to assume that on the image plane, any two points among the seven corner points are not collinear with the origin, and no point is on X axis or Y axis.

Let K be the coefficient matrix in Eq. (5-9), then K can be written as:

$$K = \begin{bmatrix} 0 & 0 & dY_1' & Y_1' & 0 & 0 & -dX_1' \\ 0 & 0 & (h+d)Y_2' & Y_2' & 0 & 0 & -(h+d)X_2' \\ wY_3' & 0 & (h+d)Y_3' & Y_3' & -wX_3' & 0 & -(h+d)X_3' \\ wY_4' & 0 & dY_4' & Y_4' & -wX_4' & 0 & -dX_4' \\ 0 & lY_5' & dY_5' & Y_5' & 0 & -lX_5' & -dX_5' \\ 0 & lY_6' & (h+d)Y_6' & Y_6' & 0 & -lX_6' & -(h+d)X_6' \\ wY_7' & lY_7' & (h+d)Y_7' & Y_7' & -wX_7' & -lX_7' & -(h+d)X_7' \end{bmatrix}$$

Let M_i be the i^{th} row of K, it is to be shown that the necessary and sufficient condition for $\sum_{i=1}^{7} a_i M_i = 0$ is that $a_i = 0$ $(i = 1,2,\dots,7)$. The sufficiency is obvious in this case, so we move on to the necessity.

From columns 2 and 6, $a_5 lY_5' + a_6 lY_6' + a_7 lY_7' = 0$ and $a_5 lX_5' + a_6 lX_6' + a_7 lX_7' = 0$. Note that$l, w, h, d$are all positive, we get:

$$a_5(X_5', Y_5') + a_6(X_6', Y_6') + a_7(X_7', Y_7') = 0 \qquad (5-10)$$

If $a_5 \neq 0, a_6 \neq 0$and $a_7 \neq 0$, then$(X_5', Y_5') = -\frac{a_6}{a_5}(X_6', Y_6') - \frac{a_7}{a_5}(X_7', Y_7')$, which is impossible since P_5 , P_6 , P_7 are non-collinear points. Therefore, at least one of a_5, a_6, or a_7 is zero. Due to symmetry, it suffices to take $a_5 = 0$. Now, if $a_6 \neq 0$ and $a_7 \neq 0$, it is also impossible since we assumed no two points have a linear relationship with the origin of the image coordinate. Hence, due to symmetry again, we take $a_6 = 0$, then it is obvious $a_7 = 0$.

Similarly, from column 1 and column 5, we have:

$$a_3\left(X_3^{'}, Y_3^{'}\right) + a_4\left(X_4^{'}, Y_4^{'}\right) + a_7\left(X_7^{'}, Y_7^{'}\right) = 0 \qquad (5-11)$$

then $a_3 = 0, a_4 = 0$.

Finally, consider the first two rows. From columns 3 and 4, we have:

$$\begin{cases} a_1 d Y_1^{'} + a_2(h+d)Y_2^{'} = 0 \\ a_1 Y_1^{'} + a_2 Y_2^{'} = 0 \end{cases} \qquad (5-12)$$

Eliminating a_1 yields $a_2 h Y_2^{'} = 0$, which means$a_2 = 0$. And based on $a_1 Y_1' + a_2 Y_2' = 0$, we get $a_1 = 0$. Hence, the necessary and sufficient condition for $\sum_{i=1}^{7} a_i M_i = 0$ is that $a_i = 0$ for $i = 1,2,\ldots,7$, which implies the linear independence of the rows of the coefficient matrix in Eq. (5-9).

5.6.3 *Solving the surveillance camera parameters*

As the matrix has full rank, the seven unknowns, i.e. $\frac{s_x r_{xx}}{T_y}, \frac{s_x r_{xy}}{T_y}, \frac{s_x r_{xz}}{T_y}, \frac{s_x T_x}{T_y}, \frac{r_{yx}}{T_y}, \frac{r_{yy}}{T_y}$, and $\frac{r_{yz}}{T_y}$ can then be solved.

We denote that $(m_1, m_2, m_3, m_4, m_5, m_6, m_7) = (\frac{s_x r_{xx}}{T_y}, \frac{s_x r_{xy}}{T_y}, \frac{s_x r_{xz}}{T_y}, \frac{s_x T_x}{T_y}, \frac{r_{yx}}{T_y}, \frac{r_{yy}}{T_y}, \frac{r_{yz}}{T_y})$. Note that the rotation matrix which contains r_{xx} , r_{xy} , …, r_{zz} is orthonormal, there are three properties we will use later:

1) $r_{ix}^2 + r_{iy}^2 + r_{iz}^2 = 1, r_{xi}^2 + r_{yi}^2 + r_{zi}^2 = 1, (i = x, y, z)$

2) $(r_{ix}, r_{iy}, r_{iz}) \cdot (r_{jx}, r_{jy}, r_{jz})^T = 0, (r_{xi}, r_{yi}, r_{zi}) \cdot (r_{xj}, r_{yj}, r_{zj})^T = 0, (i, j = x, y, z; i \neq j)$

3) $det \begin{vmatrix} r_{xx}, r_{xy}, r_{xz} \\ r_{yx}, r_{yy}, r_{yz} \\ r_{zx}, r_{zy}, r_{zz} \end{vmatrix} = 1$

Then, camera parameters $(r_{xx}, r_{xy}, \ldots, r_{zz}, T_x, T_y, T_z, s_x, f)$ can be computed in following steps.

Given that $m_5^2 + m_6^2 + m_7^2 = \frac{1}{T_y^2}$, and $m_1^2 + m_2^2 + m_3^2 = \frac{s_x^2}{T_y^2}$, we get $s_x = \sqrt{(m_1^2 + m_2^2 + m_3^2)T_y^2}$.

In order to determine the sign of T_y , we pick an object point whose computer image coordinate (u_i, v_i) is away from the image center (u_0, v_0). We assume the sign of T_y is positive, and then compute parameters $r_{xx}, r_{xy}, r_{xz}, r_{yx}, r_{yy}, r_{yz}, T_x, X_{ci}, Y_{ci}$ in the following way. $r_{xx} = m_1 T_y / s_x$, $r_{xy} = m_2 T_y / s_x$, $r_{xz} = m_3 T_y / s_x$, $r_{yx} = m_5 T_y$, $r_{yy} = m_6 T_y$, $r_{yz} = m_7 T_y$, $T_x = m_4 T_y / s_x$, $X_{ci} = r_{xx} X_{wi} + r_{xy} Y_{wi} + r_{xz} Z_{wi} + T_x$, $Y_{ci} = r_{yx} X_{wi} + r_{yy} Y_{wi} + r_{yz} Z_{wi} + T_y$

Note that $f > 0$ and $Z_c > 0$ in $\begin{bmatrix} X \\ Y \end{bmatrix} = \frac{f}{Z_c} \begin{bmatrix} X_c \\ Y_c \end{bmatrix}$. If $X_{ci} X_i > 0$ and $Y_{ci} Y_i > 0$, the sign of T_y is positive. Otherwise, the sign of T_y is negative. When the sign of T_y is known, another 7 unknowns are solved. So far, only r_{zx} , r_{zy} , r_{zz} , f , and T_z remain to be solved.

With property 1), we can derive the value of $|r_{zx}|$, $|r_{zy}|$, and $|r_{zz}|$. For example, $|r_{zx}| = \sqrt{1 - r_{yx}^2 - r_{xx}^2}$.

With property 2), every two column vectors are orthonormal, so $r_{zx} r_{zy} = -r_{xx} r_{xy} - r_{yx} r_{yy}$. Similarly, we can derive $r_{zx} r_{zz}$ and $r_{zy} r_{zz}$.

Thus, $|r_{zx} r_{zy} r_{zz}| = \sqrt{(r_{zx} r_{zy}) \cdot (r_{zy} r_{zz}) \cdot (r_{zx} r_{zz})}$.

To determine the sign of r_{zx}, r_{zy}, r_{zz}, just assume $r_{zx} r_{zy} r_{zz} > 0$, then we get $r_{zx} = r_{zx}^{'}, r_{zy} = r_{zy}^{'}, r_{zz} = r_{zz}^{'}$. With property 3), if the determinant of $\begin{bmatrix} r_{xx}, r_{xy}, r_{xz} \\ r_{yx}, r_{yy}, r_{yz} \\ r_{zx}^{'}, r_{zy}^{'}, r_{zz}^{'} \end{bmatrix}$ is positive, $r_{zi} = r_{zi}^{'} (i = x, y, z)$, otherwise $r_{zi} = -r_{zi}^{'} (i = x, y, z)$.

Finally, we come to the point of solving f and T_z. Pick the seven container corner points P_i (i=1,2,…,7) in the world coordinate and read its image coordinate (X_i, Y_i) out from corresponding

video snapshot. Since s_x has already been solved, set up seven equations in the following pattern using these seven points, $X_i = f\frac{r_{xx}X_{wi}+r_{xy}Y_{wi}+r_{xz}Z_{wi}+T_x}{r_{zx}X_{wi}+r_{zy}Y_{wi}+r_{zz}Z_{wi}+T_z}$. Transpose all the equations to get seven inconsistent equations, and then solve them using Least Squares Method to get approximate f and T_z.

5.6.4 *Image Reconstruction*

To retrieve traffic data, three-dimensional information in world coordinates is usually necessary. However, with two-dimensional information obtained from an image, only two-dimensional information in world coordinates can be calculated. Only if data in another dimension were acquired could the world coordinates be reconstructed completely.

That said, most traffic data have nothing to do with vehicle height. For instance, we can estimate the average speed of a vehicle by calculating different positions of the vehicle's projections from a camera to the ground at different moments. Therefore, the points on ground are most likely to be reconstructed since their height information is known to be zero and consistent in the model. Other points with known heights are also available and can be reconstructed when camera parameters are known. In other words, Z_w, which is the height of a point in the real world, can show information of a third dimension. Reconstruction can proceed by first rearranging Eq. (5-1) and thus yields:

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} \\ r_{yx} & r_{yy} & r_{yz} \\ r_{zx} & r_{zy} & r_{zz} \end{bmatrix}^{-1} \left(\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} - \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} \right) \qquad (5-13)$$

Considering X,Y, and f are known in $\begin{bmatrix} X \\ Y \end{bmatrix} = \frac{f}{Z_c}\begin{bmatrix} X_c \\ Y_c \end{bmatrix}$, it is justifiable to assume $X_c = k_1 Z_c$ and $Y_c = k_2 Z_c$, where $k_1 = \frac{u-u_0}{s_x f}$ and $k_2 = \frac{v-v_0}{f}$. Just plug them into Eq. (5-13), we derive Eq. (5-14) as follows:

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} \\ r_{yx} & r_{yy} & r_{yz} \\ r_{zx} & r_{zy} & r_{zz} \end{bmatrix}^{-1} \left(\begin{bmatrix} k_1 Z_c - T_x \\ k_2 Z_c - T_y \\ Z_c - T_z \end{bmatrix} \right) \qquad (5-14)$$

Since Z_w ,T_z and Matrix R are known, Z_c can be solved. World coordinate (X_w, Y_w, Z_w) is obtained.

5.7 System Implementation

The system is implemented on Raspberry Pi 4 using Python 3 as the programming language. The DBSCAN clustering is implemented using the Sklearn library and all the video analytics are developed using OpenCV 3. We have tested different OpenCV versions and both OpenCV 3 and 4 work. However, OpenCV 2 is not recommended since some functions are not available, such as extracting video background in the background modeling method. The AI classifier, i.e., the Mobilenet, is pre-trained on ImageNet and finetuned on the MIO-TCD classification dataset with 648,959 images. The deep learning framework to implement the transfer learning is TensorFlow, and then the inference graph is optimized using TensorFlow Lite and deployed to the edge device. The traffic videos are from public online videos and the AI City Challenge (Chang et al. 2020). A shell is designed to hold the Raspberry Pi device and other hardware components, such as camera, environment sensor, and communication modules. We are not describing the details in hardware selection and design since it is not the main focus. Figure 5-9 shows the edge computing product from two different angles.

Figure 5-9 The exterior of the proposed multi-task edge computing system in this chapter.

5.8 EXPERIMENTAL RESULTS

5.8.1 *Traffic Monitoring*

The training of the Mobilenet classifier is standard. There are more than 10 classes of road users in the original MIO-TCD classification dataset. We combine the class labels into 4 road user categories (car, truck, bus, and cyclist) and the background. Truck includes single-unit truck and articulated truck; cyclists include bicyclist and motorcyclist; and car includes normal car, work van, and pickup truck. Figure 5-10 shows some image samples for the classification. Table 5-2 is the training results and accuracy summarized in a confusion matrix. Background has a one hundred percent of classification accuracy. Bus and cyclist have over 95% classification accuracy; car's classification accuracy is over 90%, and truck classification is relatively lower at 87%, which may due to the common features and the labeling of pickup trucks into the car category.

As shown in the design system architecture, this Mobilnet classifier operates in a separate thread with a queue storing those "triggered frames" for further classification and validation. As shown in Figure 5-11, when moving blobs are detected and tracked in the main thread, they are first filtered using the road mask to determine if they are in the region of interest. If yes, the current

frame will be pushed to the queue and the AI thread will be triggered to further validate if the moving objects are road users or background (moving background blobs may be caused by camera shaking or sudden lighting changes); and then count the road users of each type. Since we track the road users, a new object ID will add one count to a specific road user type. The system architecture that separates the AI classifier into a separate queue allows capturing the video temporal information in real-time, and automatically finalize the classification and volume counting a few seconds later in large traffic volume conditions (i.e., when the queue is not empty). Table 5-3 presents the volume detection and counting results for three different traffic surveillance videos. The accuracy is over 95% regarding traffic volumes for each traffic direction.

Figure 5-10 Example image sample in the MIO-TCD image classification dataset.

Table 5-2 Transfer learning results of Mobilenet on the MIO-TCD classification dataset

	Car	Truck	Bus	Cyclist	Background
Car	91%	5%	2%	2%	0
Truck	6%	87%	4%	3%	0
Bus	1%	2%	96%	1%	0
Cyclist	2%	2%	1%	95%	0
Background	0	0	0	0	100%

Figure 5-11 AI trigger method that uses traditional detection and tracking as the trigger, and the finetuned Mobilenet as the final classifier for traffic volume detection.

Table 5-3 Traffic volume counting results and accuracy for three surveillance videos

	Video #1	Video #2	Video #3
Direction #1	25 / 89%	320 / 97%	451 / 94%
Direction #2	20 / 95%	348 / 92%	480 / 96%
Direction #3	7 / 88%	10 / 83%	-
Direction #4	6 / 67%	-	-

5.8.2 *Road Surface Condition Classification*

The road surface condition classification is validated using over 100,000 images collected from surveillance cameras in Washington. As aforementioned, the four features used for classification of the road surface condition is shown temperature, relative humidity, grayscale intensity, and dark channel value. The preliminary feature value summary is shown in Table 5-4. There are four categories of road surface conditions, i.e., dry, wet, partial snowy, and full snowy.

The dry and wet condition in general has higher temperature and lower grayscale intensity than the snowy conditions. The dark channel value and relative humidity value of wet condition is higher than the dry condition. Full snowy condition has a higher intensity, dark channel value, and humidity than the partial snowy condition. The classification results using five popular machine learning methods are shown in Table 5-5 to Table 5-9. The random forest method performs the best classification accuracy than the other methods, and are selected as the final classifier operated on the Raspberry Pi 4 edge computing device. Note that the parameter searching spaces for ANN is relatively small in order to keep it light-weighted. There could be better ANNs with more complicated structures.

Table 5-4 Preliminary feature value summary

Label	Temperature (°F)	Relative Humidity (%)	Grayscale Intensity	Dark Channel
1 (Dry)	45.5	64.3	96.6	85.5
2 (Wet)	42.0	90.1	94.5	87.3
3 (Partial Snowy)	29.7	75.7	118.3	106.9
4 (Full Snowy)	25.4	81.2	133.4	130.5

Table 5-5 Confusion matrix for KNN

	Dry	Wet	Partial Snowy	Full Snowy
Dry	91%	5%	4%	0%
Wet	11%	83%	5%	1%
Partial Snowy	3%	3%	90%	4%
Full Snowy	0%	0%	3%	97%

Table 5-6 Confusion matrix for SVM

	Dry	Wet	Partial Snowy	Full Snowy
Dry	98%	1%	1%	0%
Wet	57%	38%	5%	0%
Partial Snowy	49%	2%	43%	6%
Full Snowy	6%	2%	6%	86%

Table 5-7 Confusion matrix for NB

	Dry	Wet	Partial Snowy	Full Snowy
Dry	87%	5%	7%	1%
Wet	19%	78%	3%	0%
Partial Snowy	7%	4%	79%	10%
Full Snowy	3%	0%	5%	92%

Table 5-8 Confusion matrix for RF

	Dry	Wet	Partial Snowy	Full Snowy
Dry	97%	2%	1%	0%
Wet	8%	85%	7%	0%
Partial Snowy	1%	3%	91%	5%
Full Snowy	0%	0%	1%	99%

Table 5-9 Confusion matrix for ANN

	Dry	Wet	Partial Snowy	Full Snowy
Dry	93%	5%	2%	0%
Wet	21%	76%	3%	0%
Partial Snowy	8%	1%	89%	2%
Full Snowy	2%	0%	4%	94%

5.8.3 *Camera Calibration*

Figure 5-12 shows an example for calibrating our model from a roadway surveillance camera. A rough estimate for d is 3.5' here, while it has only a little influence on Matrix (T_x, T_y, T_z) and reconstruction. As shown in Figure 5-13, we manually collect the seven corner points' pixel coordinates and run the model in Matlab. Table 5-10 lists the recovered R, T, f, and s_x.

Figure 5-12 A standard shipping container in two consecutive video frames.

Figure 5-13 Identify the seven corner points and obtain their pixel coordinates for calibration.

Table 5-10 Surveillance camera calibration results

Parameters	CAMERA CALIBRATION RESULTS
Rotation matrix (R)	0.9166 -0.3991 0.0246 -0.1564 -0.4649 -0.8714 0.3432 0.8540 -0.4629
Translation matrix (T)	8.0751 13.8752 57.1012
Scale factor (s_x)	0.9475
Focal length (f)	371.7290

We haven't completely made use of the property 3) of matrix R in solving camera parameters. For property 1) and 2) of matrix R, it is clear that not all the rows or columns were taken to compute the unknowns. With the properties that have been used before, the determinant of matrix R still cannot be around the true value if it is not solved properly. Hence, the results shown in this work are convincing since the determinant of matrix R turns out to be 1.0283, which is very close to 1.

Four lane markings are selected and the pixel coordinates of their endpoints are obtained as in Figure 5-14. Running the reconstruction model to calculate the world coordinates of these eight endpoints; the results are shown in Table 5-11.

According to Table 5-11, the values of the X_w coordinates of each pair of endpoints on the same lane marking are the same within an acceptable error range (point 1 and 2, 3 and 4, 5 and 6, 7 and 8 are four pairs of endpoints on the same lane markings), which is identical theoretically because the X_w axis is approximately parallel to lane markings.

It should also be noted that the marking lines on the same section of a road should have the same length value. We can get these four marking lines' lengths approximately using the start and end point of each lane marking by simple arithmetic. Thus, the lengths are 12.8', 11.7', 12.5', 11.9', which are nearly the same within an acceptable error range.

Finally, we consider point 1, 2, 5, and 6. First of all, the Y_w values of 1 and 5 are almost the same, as are the Y_w values of 2 and 6, which is quite reasonable. Then, we estimate the lane width at 11' by calculating the average value of the distance between point 1, 5 and point 2, 6. The widths of vehicle lanes typically vary from 9 feet to 15 feet, while 10-12 feet is the most common range. 11' is a reasonably good estimate.

The same shipping container in a different frame can also be a verification for the results. Since the height of all seven corner points of the shipping container are known and do not change, we calibrate the seven points as shown in Figure 5-15. Table 5-12 presents the reconstruction results.

In the ideal situation, Point 1, 2, 3 and 4 have the same Y_w coordinates, as do Point 5, 6 and 7; Point 1, 2, 5 and 6 have the same X_w coordinates, as do Point 3, 4 and 7. As we can see, Table 5-12 presents quite reasonable reconstruction results according to this principle.

In addition, as we know the true distance between every two container corner points, we can choose any two corner points and compute their reconstructed distance to see if the result is reasonable compared to the true value. For example, from the results in Table 5-12, the reconstructed distance between Point 1 and 5 are estimated to be 20.1', which is very close to its true value 20'.

Further, if the model is established correctly, the shipping container should have the same size in different frames. The reconstructed length and width of the shipping container in this snapshot are estimated as follows:

$$\text{Length} = \frac{38.6+36.2+37.7}{3} - \frac{16.1+17.1+17.1+16.5}{4}$$

$$\text{Width} = \frac{9.8+9.2+9.0}{3} - \frac{0.0+0.8+0.2+0.8}{4}$$

The reconstructed length and width are 20.8’ and 8.8’, while the true values should be 20’ and 8’. The relative errors here are respectively 4.0% and 9.1%.

Figure 5-14 Verification using the pixel coordinates of the lane markings endpoints.

Figure 5-15 Verification using the pixel coordinates of the truck container in a different frame.

Table 5-11 Reconstruction results for lane markings

Point	u	v	Z_c	X_w	Y_w	Z_w
1	76	78	103.5	-10.3	58.5	0
2	84	97	92.5	-10.7	45.7	0
3	108	145	73.0	-10.6	22.9	0
4	125	181	63.0	-10.8	11.2	0
5	111	71	109.0	0.9	60.4	0
6	122	88	98.1	0.5	47.9	0
7	223	68	113.9	36.0	52.0	0
8	243	83	103.5	35.3	40.1	0

Table 5-12 Reconstruction results for container corner points

Point	u	v	Z_c	X_w	Y_w	Z_w
1	169	138	69.2	0.0	16.1	3.5
2	172	94	66.4	0.8	17.1	12.0
3	211	88	69.3	9.2	17.1	12.0
4	208	129	72.6	9.0	16.5	3.5
5	135	94	86.5	0.2	36.2	3.5
6	135	57	84.0	0.8	37.7	12.0
7	168	52	88.0	9.8	38.6	12.0

PART III: INTELLIGENT VEHICLE VIDEO ANALYTICS

Chapter 6. EFFICIENT VEHICLE-PEDESTRIAN NEAR-CRASH DETECTION METHOD USING ONBOARD VIDEO

6.1 Overview and Contribution

In this sub-chapter, we propose a cost-effective framework to automatically extract vehicle-pedestrian near-crashes from onboard monocular cameras. This framework is composed of four main stages: 1) pedestrian detection, 2) tracking and motion estimation, 3) vehicle-pedestrian relative position and speed calculation, and 4) near-crash detection. Following the general guidelines, the proposed vehicle-pedestrian detection system locates the pedestrian directly, instead of dealing with the complex video background in the onboard video. With a camera model, the framework does the relative speed and space calculation in the 3D space instead of the 2D image coordinate. Two near-crash indicators are used for near-crash detection. The processing framework for this study is shown in Figure 6-1. Our study addresses several challenging issues in near-crash detection including the moving video background issue, depth estimation, and real-world motion information extraction only using monocular video. The experimental results show that the proposed system is comparable to a commercial system with multiple camera sensors in terms of accuracy. Further analysis such as near-crash distribution estimation can be conducted with the proposed system. The last sub-chapter showcases an application of this method in the evaluation of the Mobileye Shield+ collision avoidance system. The contributions are two folds:

- A new method for efficient vehicle-pedestrian near-crash detection using onboard monocular camera.
- An evaluation procedure as part of a pilot study to assess commercial collision avoidance systems.

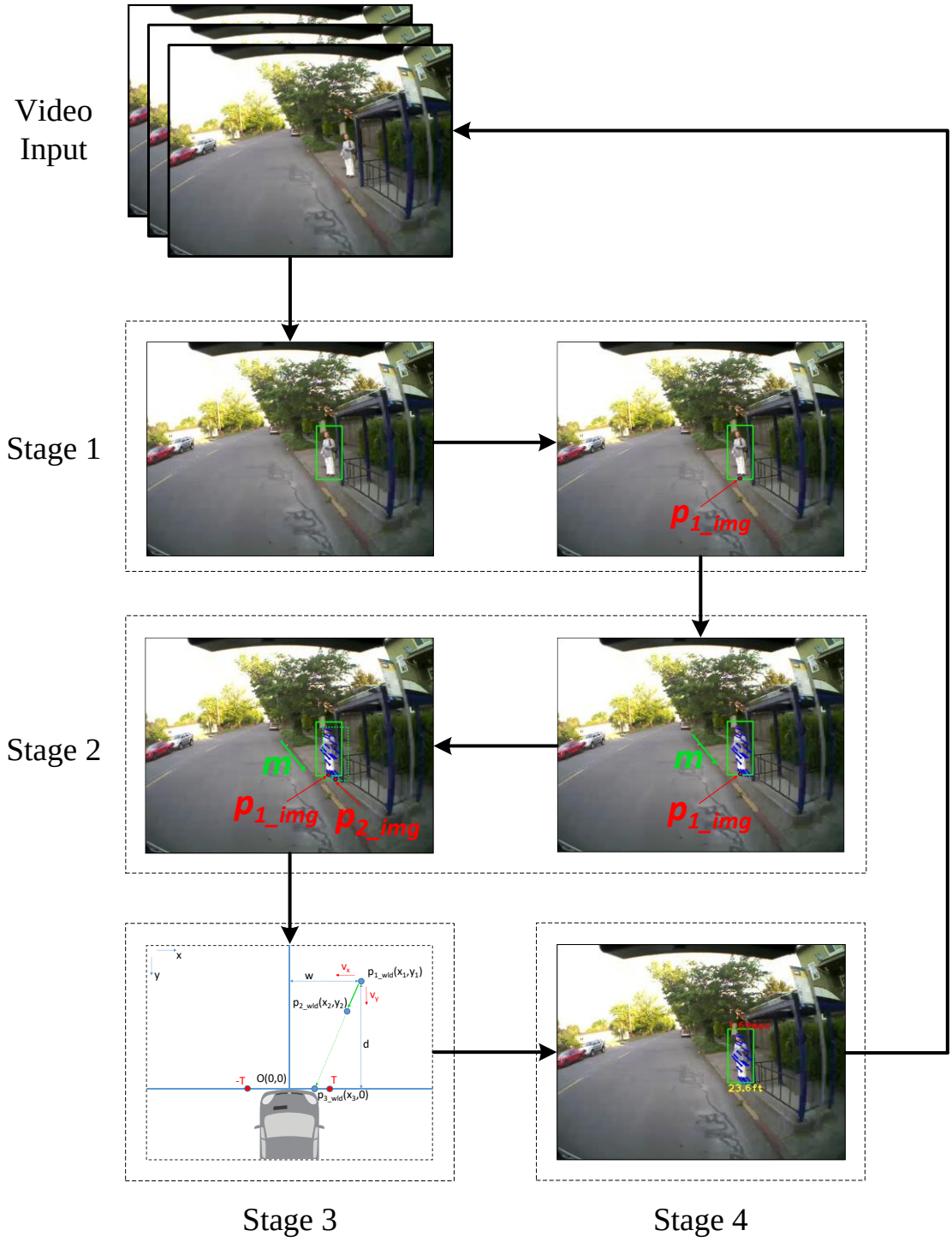


Figure 6-1 The proposed framework for vehicle-pedestrian near-crash detection through onboard front-facing camera.

6.2 PEDESTRIAN DETECTION

Pedestrian detection often plays a key role in multi-modal transportation engineering. Efficient and accurate pedestrian detection approaches would benefit traffic surveillance from many perspectives. Pedestrian detection is mainly based on the unique features of pedestrians. Generally, there are three types of single features used in pedestrian detection: gradient-based features, shape-based features, and motion-based features. Motion-based features are not suitable for pedestrian detection in onboard videos as a single feature due to the complicated motion of traffic scene which is composed of moving background, and road users with random movements. Gradient-based and shape-based features are more suitable in our case. Our framework has an advantage that it is designed for a wide range of pedestrian detectors as long as they are based on pedestrian pattern instead of motion information. In this sub-chapter, HOG is implemented as the pedestrian detector and the candidate pedestrian bounding boxes are identified using the sliding window approach. Note that the detector can be any pattern-based pedestrian detector. The input of the pedestrian detection is a video frame and the outputs are bounding boxes of the pedestrians. In order for the following description, we denote p_{1_img} the point where the detected pedestrian's feet on. In other words, p_{1_img} is the midpoint of the pedestrian candidate window's bottom edge.

6.3 TRACKING AND MOTION ESTIMATION

In traffic video analysis, KLT tracker is very effective and has been widely used in motion analysis not only in surveillance videos with fixed background but also in aerial videos with moving background. However, in onboard monocular videos, background motion is more complex than that in either surveillance videos or aerial videos. Thus, instead of tracking points in the background and clustering them, in our framework, only those interest points in the detected region

are tracked thereby background motion does not need to be directly handled. Basically, the average motion of the top 20 interest point with the least errors is used to represent the relative motion of the detected pedestrians to the vehicle in the image coordinate. If ***m*** denotes the average motion of all the interest points within the rectangle, and p_{2_img} denotes the location of the pedestrian in the next frame (see Figure 6-1), we have:

$$p_{2\_img} = p_{1\_img} + m \qquad (6-1)$$

6.4 Relative Position and Speed Calculation

With the pedestrian detected and motion *m* obtained, we developed a method to calculate the relative position and speed through monocular vision. In the image coordinate, as defined in last sub-section, p_{1_img} and p_{2_img} are the pedestrian locations in two frames (see Figure 6-2(a)). We calculate their corresponding points (see Figure 6-2(b)) in the top-view of the real-world coordinate through a camera model as follows.

Let $C(u_0, v_0)$ be the center of the image coordinate and (u_1, v_1) is the position of p_{1_img}, then

$$du = u_1 - u_0 \qquad (6-2)$$

$$dv = v_1 - v_0 \qquad (6-3)$$

where *du* and *dv* are the differences between p_{1_img} and the image center.

To find the correspondence, four camera parameters are needed: camera focal length *f*, pixel length *l*, camera installation height *h*, and camera tilt angle θ. In the top-view of the real-world coordinate, the origin $O(0,0)$ is the camera center, whose location and motion are basically the same as the vehicle. Points p_{1_wld} and p_{2_wld} are the correspondences of p_{1_img} and p_{2_img}, respectively. Let x_1 and y_1 be the *x*-coordinate and *y*-coordinate of p_{1_img}. Then x_1 and y_1 are related to *du* and *dv* by the following equations:

$$\phi = arctan(\frac{l \times dv}{f}) + \theta \quad (6-4)$$

where ϕ is the angle between ground and the line connecting p_{1_wld} and $O(0,0)$. Thus, the depth value $y_1$can be obtained, that is,

$$y_1 = \frac{h}{arctan(\phi)} \quad (6-5)$$

Then, with $y_1$and du known, $x_1$can be computed by the relation

$$x_1 = \frac{l \times du}{f} \times y_1 \quad (6-6)$$

In this way, the relative position of the pedestrian to the vehicle is obtained. Similar to the calculation of x_1 and y_1, x_2 and $y_2$can be calculated. Let *fr* be the frame rate, then the relative speed *v* between pedestrian and the vehicle is

$$v = fr \times \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (6-7)$$

Specifically, for relative speed components v_x and v_y in *x*-axis and *y*-axis respectively, we have

$$v_x = fr \times (x_2 - x_1) \quad (6-8)$$

$$v_y = fr \times (y_2 - y_1) \quad (6-9)$$

6.5 NEAR-CRASH DETECTION

With the relative position and speed estimated through monocular vision, events can be judged by calculating near-crash indicators. The most commonly used indicator is TTC and we also use TTC as the major near-crash indicator in this study, which can be obtained with the following equation

$$TTC = \frac{y_1}{v_y} \quad (6-10)$$

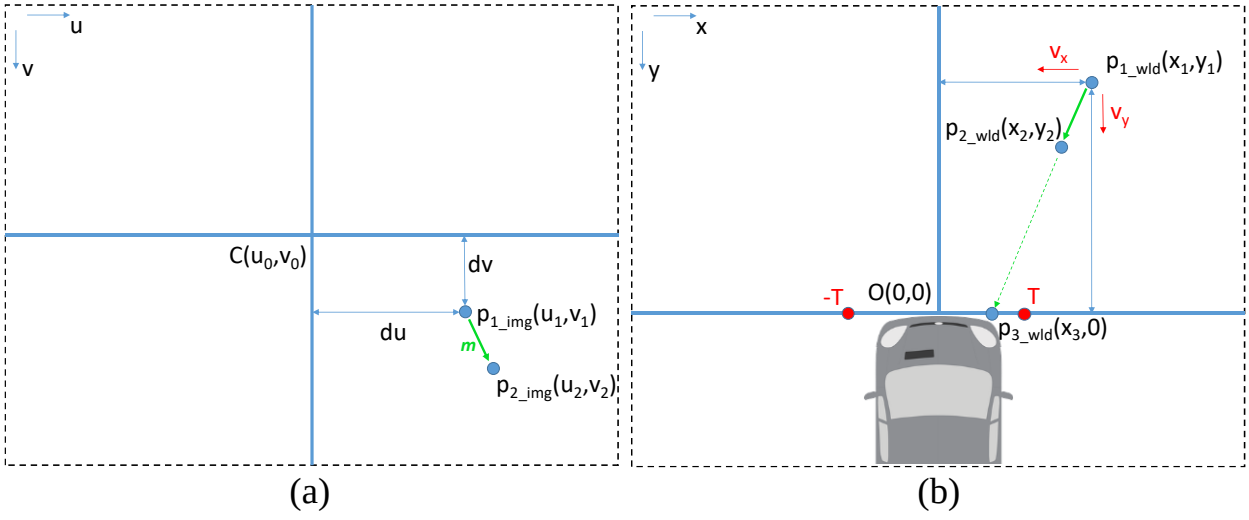
where y_1 is the y-coordinate of the detected pedestrian in the real-world coordinate (see Figure 6-2(b)).

However, Eq. (6-10) alone is not sufficient to determine whether there is a near-crash, because even if the value got by Eq. (6-10) is very small, it is possible the horizontal component of the relative speed, i.e., v_x, is very large so that the pedestrian would not hit the vehicle following the current moving direction. Thus, another indicator is needed to be set to judge if the conflict will happen following the current relative speed on x-axis. We define this indicator as distance-to-safety (DTS), which can be calculated as follows

$$DTS = v_x \times \frac{y_1}{v_y} \qquad (6-11)$$

Therefore, if both TTC and DTS are within their respective ranges for near-crash detection, i.e., $TTC < TTC_{threshold}$ and $-T < DTS < T$ (where $TTC_{threshold}$ and T are the thresholds), an event is detected. T is shown in Figure 6-2(b) and it should be set not smaller than half of the vehicle width.

Table 6-1 lists all the parameters appeared in this sub-chapter.



(a) (b)

Figure 6-2 Method to find the correspondence between image coordinates and real-world coordinates.

Table 6-1 All the parameters appeared in the description of this near-crash detection framework

Variable Name	Variable Unit	Narrative Explanation
p_{1_img}	pixel	The location of the pedestrian in the current frame
p_{2_img}	pixel	The location of the pedestrian in the next frame
m	pixel	The motion of the pedestrian between two frames
$C(u_0, v_0)$	pixel	The center of the image
(u_1, v_1)	pixel	The image coordinate of p_{1_img}
(du, dv)	pixel	The differences between p_{1_img} and the image center
p_{1_wld}, p_{2_wld}	feet	The correspondences of p_{1_img} and p_{2_img} in the world coordinate system
θ	rad	Camera tilt angle
l	pixel	Camera pixel length
f	feet	Camera focal length
$O(0,0)$	feet	Origin of world coordinate
ϕ	rad	The angle between ground and the line connecting p_{1_wld} and $O(0,0)$
h	feet	Camera height
(x_1, y_1)	pixel	The x-coordinate and y-coordinate of p_{1_img}
fr	frames-per-second	The frame rate of the input video
v	feet/s	The relative speed between vehicle and pedestrian
v_x	feet/s	The x component of speed v
v_y	feet/s	The y component of speed v
TTC	second	Time-to-collision
DTS	feet	Distance-to-safety

6.6 EXPERIMENTAL RESULTS

The data was collected on a King County Metro transit bus, which was operated in Seattle area. The onboard monocular video data used as input to our system was part of the Rosco Dual-Vision system. The comparison dataset of vehicle-pedestrian conflict was collected by the Rosco/Mobileye Shield+ system. Rosco/Mobileye Shield+ system is a vision-based collision avoidance warning system specifically designed for large vehicles (e.g., buses, trucks) which includes four cameras working cooperatively. The video for testing our method, i.e., the onboard front-facing video collected by Dual-Vision system, has a resolution of 640 × 480 pixels (width × height).and a frame rate of 7.5 frames-per-second (fps).

More than 30 hours of onboard monocular video data was used to test the performance of the proposed near-crash detection method. Figure 6-3 shows two representative samples identified as near-crashes by our system. In (a), the vehicle was approaching a stop sign when two pedestrians were crossing the street. One of the pedestrians was detected as having the potential to collide with the vehicle if no evasive action was taken. In (b), a pedestrian standing at a bus stop was detected by when the bus approached the stop and changed lanes.

The processing speed of our system is about 6 – 10 fps on a regular desktop computer, satisfying real-time operation requirement. The video detection results are compared with events logged by the Rosco/Mobileye Shield+ system with multiple camera sensors. Different TTC thresholds are used in the experiments, and the results are presented in Table 6-2. In general, the corresponding detection overlap rate ($Overlap\ rate = (N_{TotalDetection} - N_{DifferentDetection}) / N_{TotalDetection}$) between the two systems ranges from 81.5% to 90.7%, with an average overlap rate of 86.9%. The largest overlap rate occurs at when the TTC threshold is set to 2s. The results show that our video system detects majority of near-crashes picked up by the Shield+ system but

difference still exists. We manually checked those video clips showing events that are not detected by both systems at the same time. Generally, we find there are three main reasons:

1) Some events occur at the side of the bus and these events are not recorded by the onboard monocular camera. These events cannot be detected by our system because the target object (i.e., the pedestrian) does not appear in the view of the front-facing camera.
2) Some events detected by our system involve a pedestrian running towards the front of a stopped bus; a bus with no speed deactivates the Rosco/Mobileye system's vehicle-pedestrian near-crash detection function but the relative motion calculated by our system still indicates a potential conflict.
3) Some interest points inside the detected rectangle may come from objects other than the pedestrian such as corner points of lane markings, which could result in inaccurate motion estimation.

Besides safety surrogate data collection, another purpose for developing a cost-effective vehicle-pedestrian near-crash detection framework is to automatically identify hotspots and geographic distributions of events, to help drivers anticipate potential collisions in higher-risk locations. With the event data collected by our system, several plots displaying the distribution of the events are shown in Figure 6-4. It can be seen that most events occur at the right of the vehicle. This is reasonable since when a vehicle travels on roadway, normally pedestrians appear to the right of it; the left of the vehicle is traffic moving along the opposite direction thereby few pedestrians appear. However, at intersections, pedestrians are likely to appear at different spots (rather than just right of the vehicle) from the driver's perspective. By manually checking those frames with near-crashes occurring at the left or middle of the vehicle, we find most of them do

occur at intersections. For example, an event may occur when a left-turning vehicle has a conflict with a pedestrian crossing the street.

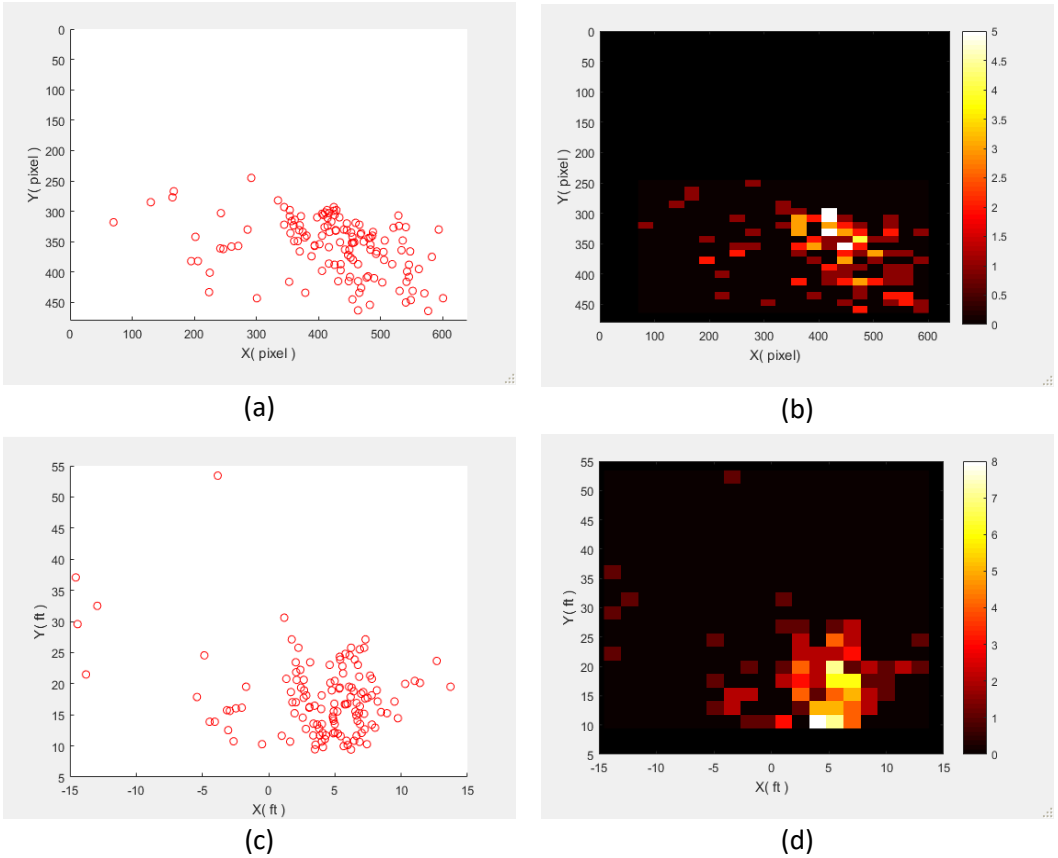
Also, we can see that the region with densest events are different in the image coordinate ((a), (b)) and the real-world coordinate ((c), (d)): the densest region in the image coordinate is the top right region, but in the real-world coordinate it is the bottom right region. That is to say, most near-crashes occur at a relatively farther distance to the vehicle in the image coordinate intuitively, but closer to the vehicle in the real-world coordinate. This result is surprising at first glance, but the reason is that in the image coordinate, objects of same size at a farther distance to the camera occupy less pixels than those closer; in other words, a pixel represents larger real-world size at a farther location to the camera. Thus, although the fact is more near-crash events occur in the region closer to the vehicle, it looks like more near-crashes occur at a relatively farther distance in the image space. These findings may help drivers improve driving behavior and overall safety by knowing the distribution of the near-crashes.

Table 6-2 Summary of the comparison results with the Rosco/Mobileye Shield+ system

$TTC_{threshold}$	4s	3s	2s	1s
Number of different detections	20	10	4	1
Number of total detections	108	81	43	8
Detection overlap rate	81.5%	87.7%	90.7%	87.5%

(a) (b)

Figure 6-3 Sample frames showing near-crash events detected by the proposed system.



(a) (b)

(c) (d)

Figure 6-4 Scatter plots and heat maps showing the distribution of near-crashes in image coordinates (a) and (b) and top-view of real-world coordinates, (c) and (d).

6.7 APPLICATION TO THE EVALUATION OF THE MOBILEYE SHIELD+ SYSTEM

6.7.1 *Background*

Onboard sensing data can be a valuable source for real-time autonomous driving features, such as collision avoidance system. The Rosco/Mobileye Shield + system mentioned in the last sub-chapter is a state-of-the-practice collision avoidance warning system (CAWS) for transit vehicles. The CAWS includes four cameras: a master attached to the center of the inside windshield, a camera attached to the inside windshield positioned to cover the blind zone on the left front created by the "A" pillar, and one external forward-facing camera on each side of the bus towards the rear, to cover blind zones behind the driver. The rear external cameras are encased in ruggedized, heated enclosures mounted 78 to 82 inches (198-208 cm) above the ground. Figure 6-5 illustrates the locations of the system components on a typical bus.

The system provides coverage of blind zones where vulnerable road users may be hidden from the driver's view, and by alerting the driver to avoid potential collisions. The Mobileye Shield+ system illuminates one of three indicators located on the windshield to draw the driver's attention towards a potential pedestrian collision. The indicator shows a yellow light if a pedestrian or bicyclist is calculated to be within 2.5 seconds or less of colliding with the bus. The indicator flashes red and an alarm sounds if a pedestrian or bicyclist are within one second or less of colliding with the bus. An indicator mounted in the center of the windshield also provides forward collision warning, headway monitoring and following time, lane departure warning, and speed limit violation warning. Because buses routinely change lanes in low speed operation while pulling into and out of stops, the lane departure feature was disabled in this pilot to avoid unnecessary distraction for the driver.

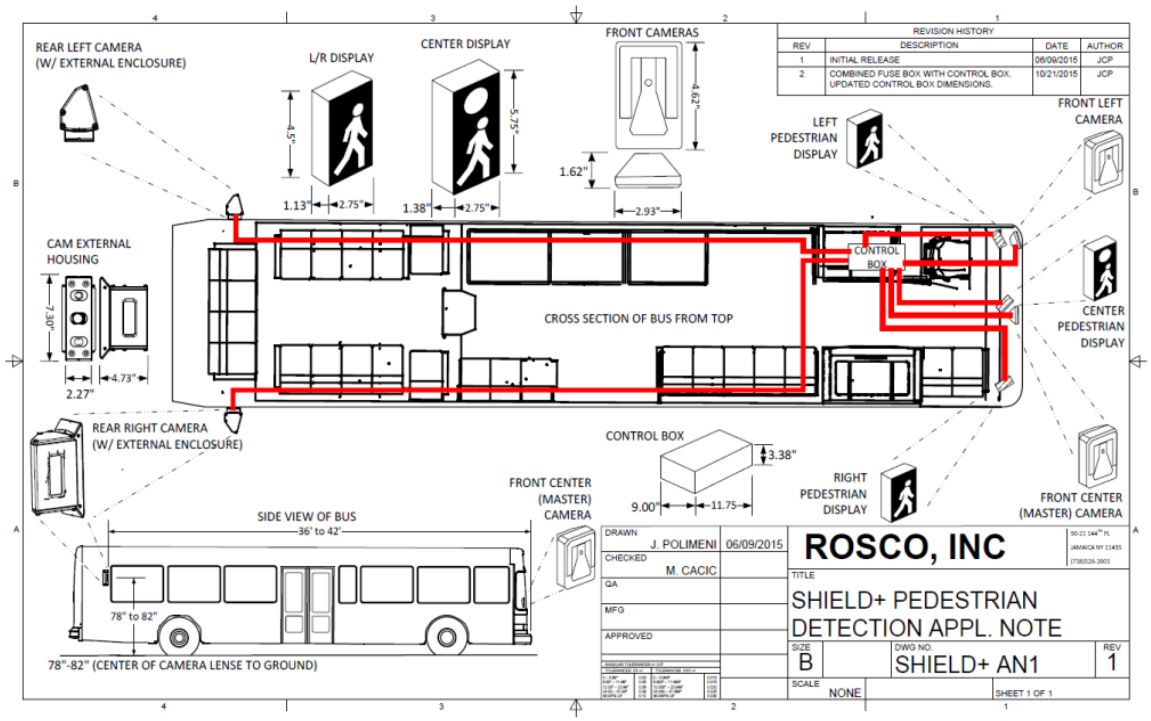


Figure 6-5 Diagram of typical Shield+ system component layout (Spears et al. 2017).

6.7.2 *Testing for False-Positives and False-Negatives*

A key task for the pilot was to evaluate the accuracy of the CAWS in correctly identifying incidents involving near-misses with pedestrians and filtering out incidents which posed no imminent risk of collision with pedestrians. Evaluating this aspect of CAWS performance involved reviewing video and telematics data to detect false positives and false negatives. A false positive (FP) is defined as the presence of pedestrian/bicyclist event in the telematics data that is not confirmed by the video. A false negative (FN) is defined as an incident in which a pedestrian with an estimated time to collision (TTC) less than a specified threshold is not detected by the CAWS. False positives generate warnings that can annoy drivers and divert their attention from the driving task. False negatives are potentially more serious because they could place pedestrians at risk.

The developed near-crash detection program was applied to automatically checking the front-facing videos and filtering out most of the frames without events. Another round of manual checking was conducted to further verify the detection results. In order to set an appropriate TTC threshold for evaluation, we use a detection overlap rate (*OR*) to find the TTC threshold that would maximize *OR*. *OR* is defined in Eq. (6-12),

$$OR = \frac{A \cap B}{A \cup B} \qquad (6-12)$$

where *A* is the set of detections identified by our near-crash detection program and *B* is the set of detections identified by Shield+. *OR* ranges from 0 to 1 and a larger *OR* indicates a TTC threshold that more closely approximates the detection performance of Shield+. All events with TTC less than 2.5s detected by the program were identified for manual checking.

To identify FPs the processor is run on video clips labeled with events. If the processor detects the event in the video, it is considered a true-positive ($A \cap B$). However, if no event is detected by the processor in the video clip, further checking is required. Audio alerts can be heard when the clips are played on the Rosco viewer. Manual checking process for FPs runs as follows: 1) find the time of audio alert; 2) check both the front facing video and side videos to see if there is a conflict; 3) if there is no conflict observed such as no appearance of vulnerable road users or no obvious aggressive movement around the time of alert, the event would be considered a FP ($\text{FP} \in A \cup B - A$).

The identification of FNs is more time consuming. The method aims to minimize checking time and maximize the probability of finding all FNs. The first step in identifying FNs is to run the program on the whole video dataset, to largely reduce further checking of events. False detections of road users are filtered out first in the manual checking process. For example, a tree mistakenly recognized as a pedestrian will be discarded immediately. The remaining detected

events are considered true events with the assumption that the KLT-based motion estimation process has no significant error. In the end, the events detected by the system but not Shield+ are regarded as the FNs (FN $\in A \cup B - B$).

Two typical FP patterns were found during the testing period as seen in Figure 6-6. The first pattern was false detection of road users, in which a PCW was generated by movement of the bus toward an object similar in shape to a pedestrian. For example, a standalone stop sign did not generate a warning, but for some Ben Franklin buses during April and May 2016, a stop sign with other objects around it did. The second typical pattern for false positives involved pedestrians/bicyclists moving parallel to and on the left of the bus either in the same or opposite direction. In some instances, pedestrians were on sidewalks at some distance and not on a trajectory to collide with the bus. The second pattern did not generate FP's for all buses, and may be caused by individual installation or parameter settings.

Very few FN's were identified and no strong patterns emerged. Late detections were defined as FN's. Two example false-negatives identified by the processor are shown in Figure 6-7. Both (a) and (b) were detected by the Shield+ system but the warnings were late. In (b), the warning was generated after the bus passed had the pedestrian.

Table 6-3 shows summary statistics based on the sample of videos that had been fully processed prior to this publication. The total FP rate is about 3.21% and the FN rate is about 0.30%. In summary, the Shield+ system rarely missed potential conflicts and was found to be robust in challenging scenarios such as adverse weather, low lighting condition, direct sunlight, and shadows.

Figure 6-6 Typical patterns for false positives.

Figure 6-7 Examples of late detections identified as potential false negatives.

Table 6-3 Summary statistics for identification of false positives and false negatives

	Ben Franklin Transit	Community Transit	King County Metro	Kitsap Transit	Pierce Transit	Total
Events	1640	1062	430	1477	1461	6070
FP	111	24	7	39	14	195
FN	3	4	4	2	5	18
FP Rate	6.77%	2.26%	1.63%	2.64%	0.96%	3.21%
FN Rate	0.18%	0.38%	0.93%	0.14%	0.34%	0.30%

Chapter 7. EDGE COMPUTING SYSTEM FOR REAL-TIME NEAR-CRASH DETECTION OF INTELLIGENT VEHICLES

7.1 Overview and Contribution

Near-crash has been a critical surrogate safety measure for transportation safety research (Kataoka et al. 2018; R. Ke, Lutin, et al. 2017; Klauer et al. 2006; Makizako et al. 2018; Talebpour et al. 2014; J. Wu et al. 2018). The key reason is that the number of actual collisions in a certain scenario is often insufficient to support big data analytics or even traditional statistical models. Near-crashes are traffic incidents that have the potential to develop into collisions. They reflect the safety situations and designs, and usually appear in much larger numbers than actual collisions. The most commonly used source for near-crash extraction is traffic surveillance video, given its low cost, wide deployment, and rich information. The researchers at the University of British Columbia have been one of the leading groups dedicated to near-crash identification, extraction, and analysis using surveillance videos (Ismail et al. 2009; Ismail, Sayed, and Saunier 2010; Saunier, Sayed, and Ismail 2010; Sayed, Zaki, and Autey 2013). Recently, the City of Bellevue, WA has been leading a collaborative effort with Microsoft and the University of Washington to develop large-scale video analytics for near-crash research using city-wide surveillance cameras towards achieving the mission of Vision Zero (Loewenherz, Bahl, and Wang 2017).

With the emergence of intelligent vehicle (IV) concepts and technologies, near-crash becomes an even more valuable data source for not only traditional traffic safety research but also IV safety. The latest IVs have been demonstrated to be able to handle most situations they may encounter. However, the lack of corner cases for training and testing IVs is a major cause that is slowing down the pace to achieve the goal of Level-5 (L5) fully autonomous driving (Bolte et al. 2019; Chou et al. 2018). While corner cases can be generated in simulations to support some IV research, it is a

must step for future IVs to be tested with as many real-world corner cases as possible to ensure safety. Near-crash data can actually serve as part of the corner case data, and has enormous potential to support IV safety research once collected in large scale in the open road.

To this end, this chapter introduces a light-weight edge computing system for real-time near-crash detection and data collection. The system is a low-cost and standalone system that is backward-compatible with existing vehicles. The system is developed based on a recently-released IoT platform, Nvidia Jetson TX2. The key method in the system's main thread is a video analytics near-crash detection method with real-time processing efficiency, high detection accuracy, and excellent transferability to different dashboard cameras. The near-crash detection method starts with understanding the motion patterns in the camera view, and then the selection of deep-learning-based object detection and online tracking. With the bounding boxes generated by the detection and tracking, we model the target road user's size changes and moving directions with linear-regression complexity to effectively estimate time-to-collision (TTC) and relative motions. Based on the TTC values and relative motions, we propose several new rules to near-crash identification. Moreover, the study proves and shows that the proposed near-crash detection method is insensitive to intrinsic camera parameters, and this property makes it adaptable to different dashboard camera. We design experiments to validate the system and demonstrate its potential to accelerate IV innovation and research in multiple ways.

The contributions of this chapter are summarized as follows:

- An efficient near-crash detection algorithm to model the bounding boxes generated by deep learning-based object detector for TTC and motion estimation. The algorithm is designed to be insensitive to intrinsic cameras parameters, thus, it has great transferability to different dashcams.

- A multi-thread, lightweight, and standalone edge computing system architecture that achieves real-time near-crash detection and data fusion (camera, LiDAR, GPS).
- The system is demonstrated to be backward-compatible to different existing vehicles, such as cars and transit vehicles, hence, it shows enormous potential for large-scale deployment to partially support IV corner case collection.

7.2 Related Work

A key question for simulation-based IV testing is: how well does simulation match the field implementation? The answer varies from case to case, but no matter how well IV features and systems perform in simulation, they have to be tested in the real world. An intermediate step between simulation and real-world testing is combination testing (S. Feng, Feng, Yan, et al. 2020; Y. Feng et al. 2018; Fremont et al. 2020; Horváth et al. 2019; L. Li et al. 2019; J. Ma et al. 2018; Tettamanti et al. 2018; Uchida, Tagawa, and Sato 2017; Z. Xu et al. 2017). In some places, this type of virtual-real testing is called hardware-in-the-loop (HIL) testing. Researchers at the University of Michigan built an augmented reality (AR) environment that combined the VISSIM simulator and MCity test track (Y. Feng et al. 2018). The testing vehicles were synchronized with the simulation. This pioneering virtual-real platform has been the support for more recent IV studies.

Feng et al. took one step further by integrating their innovative TSLG theory (S. Feng, Feng, Sun, et al. 2020; S. Feng, Feng, Yu, et al. 2020) into this platform to generate critical scenarios in the simulation and then interact with the real vehicles at the test track (S. Feng, Feng, Yan, et al. 2020). Similar research was led by UC Berkeley, in which they presented a formal method to generate test scenarios for IV in simulation and select and implement tests on the road (Fremont et al. 2020). They also made quantitative and qualitative comparisons between the recorded data

from the simulation and the test track for the same synthesized testing cases. Another virtual-real platform developed by Japanese researchers was an AR vehicle, which ran on the test track with three cameras and three big monitors (Uchida, Tagawa, and Sato 2017). When the driver was driving the real car on the real road, virtual traffic situations were shown on the three monitors in front of the driver. With this setup, driver performance in critical scenarios can be evaluated and reproduced without the risk of real collisions. Tettamanti et al. designed an environment for IV testing with a Smart car and simulation (Tettamanti et al. 2018). Instead of using roadside unit (RSU) for data transmission and VISSIM as the simulation platform (Y. Feng et al. 2018), in this study, the two-way message transmission was done via CAN communication, and the simulation was developed in SUMO. Li et al. applied the parallel vision techniques to transfer the real-world sensing data in the normal daytime to virtual-world sensing data in less frequently encountered situations for testing (L. Li et al. 2019). Virtual-real testing with real tracks or vehicles can still be costly, labor-intensive, and time-consuming to set up (Fremont et al. 2020). Xu et al. showed that it was feasible to build an inexpensive and safe HIL platform to test certain IV functions with scaled vehicles and roadways (Z. Xu et al. 2017).

Real-world IV testing is expensive in both time and cost, but it is a must-step for any IV technology to be ready for production. While large-scale real-world tests are typically led by high-tech industrial companies or vehicle OEMs, many academic institutions have been actively contributing to real-world IV testing as well, especially on the topics of algorithm/system innovation (Broggi et al. 2013, 2015; Kunz et al. 2015; Nothdurft et al. 2011), technology evaluation (Harper, Hendrickson, and Samaras 2016; Soule et al. 2020; Spears et al. 2017), field tests for traffic modeling (Raboy et al. 2020; Xiangmo Zhao et al. 2020), field data generation (Geiger et al. 2013; de Gelder et al. 2019; R. Ke, Lutin, et al. 2017; Z. Kong et al. 2020), etc. In

the Stadtpilot project, Nothdurft et al. tested autonomous driving on Braunschweig's inner-city ring road with a vehicle called Leonie (Nothdurft et al. 2011). A few other projects have been conducted to test certain IV features designed by the team on open roads, such as the test of a sensor-independent fusion approach at Ulm University (Kunz et al. 2015), the test of the BRAiVE prototype by VisLab including a trip from Itay to China (Broggi et al. 2013), and the test on Parma urban roads and freeways (Broggi et al. 2015).

The Smart Transportation Applications and Research (STAR) Lab at the University of Washington has been collaborating with industrial partners and the government to conduct IV technology evaluation (Soule et al. 2020; Spears et al. 2017). This type of collaboration involving academia, industry, and government can accelerate IV testing by sharing resources and expertise. In the sub-category of field tests for traffic modeling, tests on tracks or open roads with mixed traffic are designed to reveal the impact of IV to traffic and driver behaviors. Raboy et al. conducted a field experiment on lane-change maneuvers, and their proposed testing platform also can be used to model and evaluate other research subcomponents of IV (Raboy et al. 2020).

Another sub-category of research focuses on generating new data for benchmarking IV-related algorithms and approaches. For example, KITTI is a representative benchmark suite that provides a variety of sensing data for IV algorithms such as road detection, object detection, object tracking, depth estimation, optical flow estimation, etc. Other studies such as vehicle-pedestrian near-crash data collection (R. Ke, Lutin, et al. 2017) and physical-world-resilient adversarial data generation (Z. Kong et al. 2020) have been conducted as well.

7.3 Understanding Relative Motion Patterns for Near-Crashes

Relative motions between the ego-vehicle and other road users are important cues for near-crash detection using a single camera (R. Ke, Lutin, et al. 2017; Kilicarslan and Zheng 2019).

Relative motion patterns as well as the relationship between a pattern in the camera view and its corresponding pattern in the real world must be understood (see Figure 7-1). The relative motion patterns between two road users vary from case to case. Roadway geometry, road user's behavior, relative position, traffic scenario, and so on, are all factors that may affect the relative motion patterns. For example, from the ego-vehicle's perspective, its motion relative to a vehicle it is overtaking in the neighbor lane and that to another vehicle it is following in the same lane are different.

Relative motion that has the potential to develop into a crash /near-crash is characterized from the ego-vehicle's perspective as the target road user moving towards it. This kind of relative motion is shown as a motion vector of the target road user moving vertically towards the bottom side of the camera view. Examples are shown as solid red arrows in Figure 7-1. In the real-world top view, the three solid red arrows represent the relative motions between the ego-vehicle and each of three road users (a pick-up truck, a car, and a pedestrian). Each of the three camera sight lines aligns with a relative motion vector (Z2, Z4, and Z7). In the camera view, the lines of sight are shown as vertical bands. The relative motion vectors for near-crashes in the top view correspond to vectors moving towards the bottom in the camera view aligning with Z2, Z4, and Z7.

Two road users have a relative motion at any time. In addition to the near-crash cases defined above, other patterns may occur. First, a target road user may move towards the ego-vehicle, move away from the ego-vehicle, or stay at the same distance to the ego-vehicle. These can be identified as object image size changes in the camera. This property will be utilized later in our approach. Image size decreasing or no size change would not indicate a potential crash or near-crashe. For size increasing, there are three cases. The first cases are the potential crashes, shown as the solid red arrows in Figure 7-1. The second are the warning cases, shown as the dotted orange arrows,

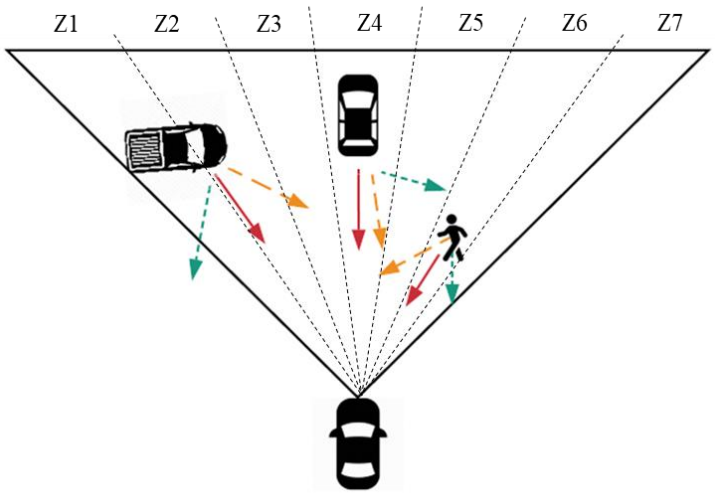
in which the relative motion is towards the center line of sight of the camera (the pick-up truck and the pedestrian), or the relative motion is slightly different from the solid red arrow while the target road user is at the center line of sight (the car). The warning cases could develop into crashes if there are slight changes in the speeds or headings of either the target or the ego-vehicle. The third case is the safety case that relative motion is moving away from the center line of sight, shown as the dotted green arrows in Figure 7-1.

7.4 EDGE COMPUTING SYSTEM ARCHITECTURE

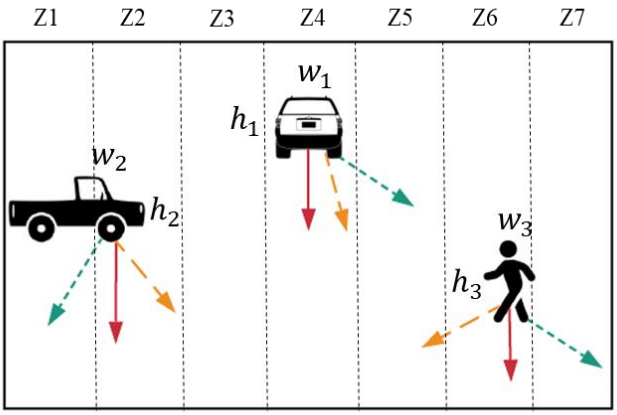
The overall system architecture on the edge computing platform is shown in Figure 7-2. The two major functions of the system are near-crash detection and data collection. Given the real-time operation requirement for both functions, the design should be simple enough to support to be highly efficient and sophisticated enough to use the Nvidia Jetson's computational power for high accuracy and reliability. The near-crash detection method also should be insensitive to camera parameters to accommodate large-scale deployments.

The system is implemented in a multi-thread manner. Four different threads are operating simultaneously: main thread, data collection thread, video frame reading thread, and CAN receiving thread. Note that the CAN receiving thread for receiving LiDAR event messages are only available for our tests on buses. For normal cars, there are just the other three threads. The proposed near-crash detection method is implemented in the main thread. When near-crash events are detected, a trigger will be sent to the data collection thread, and it will record video frames from a queue (a global variable) and other data that are associated with the near-crash event. The third thread for video frame reading keeps the latest video frame captured from the camera in another queue and will dump previous frames when the capturing speed is faster than the main thread's frame processing speed. The optional thread assumes there are other systems running at

the same time for event detection, and the other system will send a trigger message via the Controller Area Network (CAN) to the proposed system so that data can be collected as well regarding the other system's event. This optional thread can be adopted for the purposes of sensor data fusion, system evaluation, and event video collection.



(a) Top view



(b) Camera view

Figure 7-1 The corresponding relative motions, relative locations, and lines of sights between the ego-vehicle and three other road users. In the case of target's size increasing in the camera view, there are still three types of relative motions between the ego-vehicle and the target road user — solid red arrows: potential crashes; dotted yellow arrows: warnings; dotted green arrows: safety.

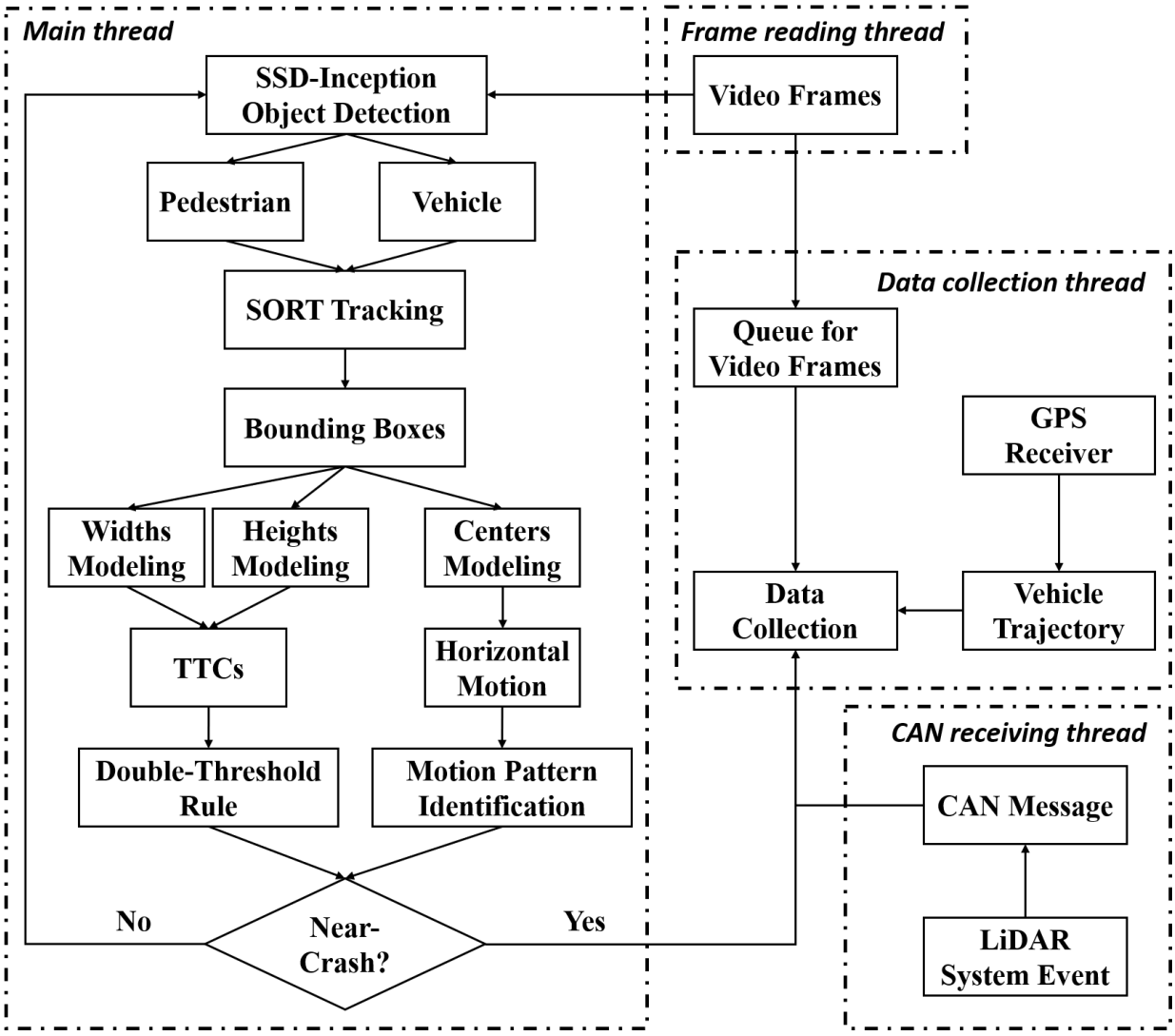


Figure 7-2 The system architecture on the edge computing device

7.5 Real-Time Camera-Parameter-Free Near-Crash Detection Method

In this sub-section, we introduce the design and development of the near-crash detection method in detail.

7.5.1 *Deep-Learning-Based Road User Detection and Tracking*

The main thread starts with applying a deep-learning-based object detector to every video frame. Deep-learning-based object detection can simultaneously localize and classify objects with

high accuracy (Ruimin Ke et al. 2018; Yifan Zhuang, Ke, and Wang 2020). It has become a standard feature of the latest IV technologies. However, one disadvantage of deep-learning-based inference is its high computational cost, which prevents it from being deployed for certain applications. As one of the most powerful IoT devices at present, Nvidia Jetson TX2 is capable of running some deep object detectors in real-time and running the inference with TensorRT-optimized inference neural networks.

For traditional IoT devices, Tiny YOLO (You Only Look Once) and SSD (Single Shot Multibox Detector)-Mobilenet are two of the most popular deep-learning-based detectors given their high inference efficiency. However, we chose a more complicated detector, SSD-Inception, a real-time detector on Jetson TX2 with nearly 30 frames-per-second (FPS) detection speed and better accuracy. The system keeps detected pedestrians and vehicles for further processing.

The object detection creates bounding boxes and identifies the types of road users in each video frame. To associate the information from each frame and find each road user's movement, a standard step following object detection is object tracking. SORT (Simple Online Real-time Tracking) tracking is a recent benchmark for object tracking with online and real-time performance. It achieves good tracking accuracy without the need for any complicated features but solely the bounding box information. It also can eliminate some false-positives and false-negatives that are generated in the detection phase. Some studies demonstrated it to be a suitable tracking method for intelligent transportation applications (Ruimin Ke, Feng, et al. 2020; Ruimin Ke, Zhuang, et al. 2020).

7.5.2 *Modeling bounding boxes in linear-regression complexity for camera-parameter-free TTC estimation*

An object appears larger in the camera view as it is approaching the camera, and smaller as distance to the camera increases. Researchers at Mobileye published a paper as early as 2004 to show that it was possible to determine TTC using size changes (Dagan et al. 2004). In this study, the proposed approach for TTC estimation mainly considers: (1) leveraging the power of recent achievements in deep learning, (2) making the computation as efficient as possible to support real-time processing on Jetson, and (3) transferability to any dashboard camera without knowing the camera's intrinsic parameters.

Use of deep learning was discussed in the last sub-section. SSD+SORT detects and tracks road users with high accuracy. However, the next step, which is the near-crash identification, must be simple and effective. Otherwise, the real-time requirement would not be satisfied by the IoT device.

Object detection and tracking provide the locations, categories, and sizes of objects in the bounding boxes information. However, bounding boxes are just approximate sizes of the objects and are not used for accurate determination of object size. Particularly, given two consecutive frames, the size change of an object is subtle; and in many cases, this change is not recognizable due to noise in the bounding box generation. In our initial experiment, we also found that the size of an object in the previous frame may be even larger than that in the next frame.

Another reason for inaccurate size change detection in neighboring frames is that the time is too short in between two consecutive frames. Given a video with a frame rate of 24 FPS, the next frame is captured in less than 0.05 seconds. Thus, for size change detection we use more frames to compensate for the noise in each frame and increase the time interval for the detection. Linear regression is used for bounding boxes' heights or widths over a group of consecutive frames. We

found that 10 to 15 frames are enough to compensate for noise and the time associated with 10 to 15 frames is still small enough (about 0.5 second) to assume that the road user's motion is consistent.

Therefore, the input to the linear regression is a list of heights or widths extracted from the bounding boxes, and the slope outputted by the regression will be the size change rate.

Let us denote the size change rate as r_t, and the size of the road user in the video frame as s_t at time t. At the same time, in the real world, the longitudinal distance between the target road user and the ego-vehicle is D_t, the relative longitudinal speed is V_t, the target road user's size is S_t, and the camera focal length is f. Based on the pinhole camera model, there is

$$\frac{s_t}{f} = \frac{S_t}{D_t} \tag{7-1}$$

Relative speed is the first derivative of relative distance, and that size change rate is the first derivative of the object size over time

$$V_t = \frac{dD_t}{dt}, \qquad r_t = \frac{ds_t}{dt} \tag{7-2}$$

Since the real-world target road user's size does not change over time, there is the following equation

$$0 = \frac{dS_t}{dt} = \frac{d\left(\frac{D_t s_t}{f}\right)}{dt} \tag{7-3}$$

And since the focal length does not change over time, we have

$$0 = \frac{d(D_t s_t)}{dt} = \frac{dD_t}{dt} \cdot s_t + \frac{ds_t}{dt} \cdot D_t = V_t s_t + r_t D_t \qquad (7-4)$$

Thus,

$$TTC = -\frac{D_t}{V_t} = \frac{s_t}{r_t} \qquad (7-5)$$

According to Eq. (7-5), TTC can be calculated as the size of the bounding box at time t divided by the size change rate at time t. It is not related to the focal length or other intrinsic camera parameters. The TTC value can be either positive or negative, where being positive means the target is approaching the ego-vehicle, and being negative means it is moving away from the ego-vehicle.

7.5.3 *Height or Width?*

There are two options for the size of the road user in the camera view, height and width. We argue that height is a better indicator than width. From the ego-vehicle's perspective, it may observe a target vehicle's rear view, front view, side view, or a combination of them, depending on the angle between the two vehicles. That is to say, the bounding box's width change may be caused by either the relative distance change or the view angle change. For example, when the ego-vehicle is overtaking the target vehicle, or the target vehicle is making a turn, the view angle changes and will lead to the bounding box's width change.

However, the bounding box's height of the target vehicle is not influenced by the view angle; it is solely determined by the relative distance between the two vehicles. Similarly, a pedestrian walking or standing on the street may have different bounding box widths due to not only the relative distance to the ego-vehicle but also the pose of the pedestrian; but the height of a pedestrian is relatively constant.

Despite the challenge of using width to determine an accurate TTC, it still provides valuable information. Since we are using only less than one second of frames for the calculation, the view change does not contribute as much as the distance change, so width still roughly shows the longitudinal movement of the road user. This is very important in some cases. For instance, a vehicle moving in the opposite direction of the ego-vehicle is truncated by the video frame boundary. In this case, the height of the vehicle increases while the width decreases. This is not a near-crash case at all, but the TTC can be very small and falsely indicate a near-crash by only looking at the height change.

We propose a double-threshold rule: if the TTC threshold for determining a near-crash is δ, we will set this δ as the TTC threshold associated with the height regression. At the same time, we have another TTC threshold φ associated with the width regression. The second threshold φ is to ensure that the width and height changes are in the same direction. The rule is represented as

$$0 < \frac{h}{r_h} < \delta\,, \qquad 0 < \frac{w}{r_w} < \varphi, \qquad \delta < \varphi \tag{7-6}$$

where r_h and r_w are the change rates for height h and width w. It is a necessary condition for a near-crash.

7.5.4 *Modeling Bounding Box Centers for Horizontal Motion Pattern Identification*

As shown in Figure 7-1, there are three scenarios for the case that a road user approaches the ego-vehicle; they correspond to potential crashes, warnings, and safe scenarios. Besides TTC, these scenarios can be differentiated with the relative horizontal motion between the ego-vehicle and the target. This needs to be calculated with computationally fast methods as well. We propose to apply another linear regression using a list of bounding box's centers of the target road user. The regression result would be able to indicate the moving direction of the road user in the camera view.

In general, when the target's location is closer to the bottom and closer to the center line of sight, the risk of a collision is higher, so the threshold for the moving direction ω is looser. We propose a rule to show this judgment as

$$\alpha < \omega \cdot (C_x - C_{los}) \cdot \left(B_y - B\right) < \beta \qquad (7-7)$$

where C_x is the center's x coordinate, C_{los} is the center line of sight, B_y is the bottom side of the bounding box, and B is the bottom of the video frame. Since cameras have different resolutions, $(C_x - C_{los})$ is normalized to [-1, 1] and $\left(B_y - B\right)$ is normalized to [0, 1]. The two thresholds are α and β; α should be set to negative to capture the potential warning scenarios (the orange dotted arrows in Figure 7-1). And β should be just slightly larger than zero to capture the potential crashes (the solid red arrows in Figure 7-1) and filter out most of the safe scenarios (the green dotted arrows in Figure 7-1). Eq. (7-6) and Eq. (7-7) together identify near-crash events.

7.6 EXPERIMENT RESULTS

7.6.1 *Experiment Design*

Local experiments with locally stored videos at Jetson and real-world experiments with onboard real-time video feeds were selected as two groups for testing the system. Local video resources covered a lot of historical near-crash scenarios as well as other corner cases. It was a better source to evaluate the near-crash detection method we proposed in this paper. Real-time video stream data was captured by the system on cars and buses. Over 1000 hours of tests have been conducted so far. Local video data were also collected from online platforms (e.g., YouTube) and some team members' dashboard cameras (see Figure 7-3 for some examples). Real-world tests have been conducted on two Honda cars and four Pierce Transit buses for over three months in the year 2020. Figure 7-4 shows the system and testing buses for the real-world test. From top to bottom: the systems ready to be installed (before installation), three of the testing buses at Pierce Transit, the radio box behind bus driver's seat where the system works, and the system being tested in the radio box.

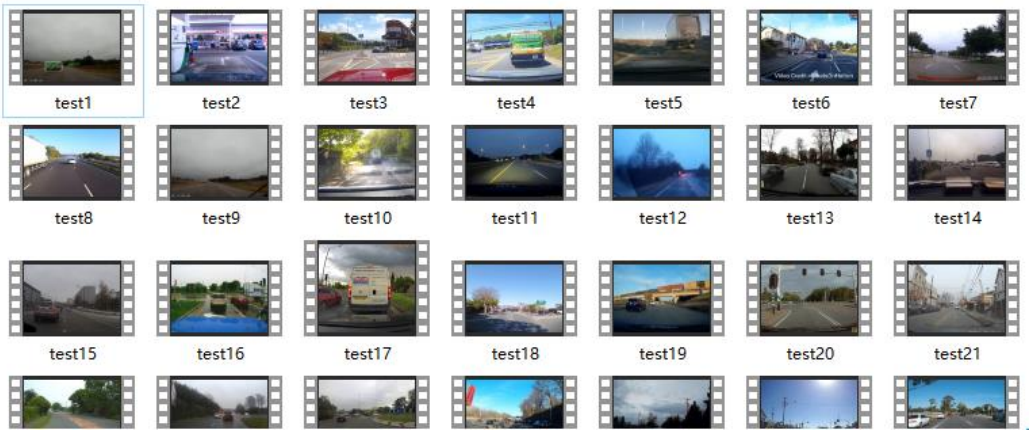


Figure 7-3 Video data samples for the local tests on the edge computing system for near-crash detection.

Figure 7-4 The system prototypes, buses for the real-world testing, and the bus radio box where the system works.

7.6.2 *Hardware Components*

The system consists of an Nvidia Jetson TX2, a dashcam (can be USB camera or IP camera), a GPS receiver, an in-vehicle power inverter, a PEAK CAN adapter for CAN bus communication, an external circuit based on Arduino board for auto bootup, a shell for the Jetson device, an ethernet cable, two power cables, an internet switch, mounting materials, and a cloud server. The Nvidia Jetson device is the key processing unit of the system, running the near-crash detection, data collection, sensor fusion, data transmission threads and algorithms. The Jetson was powered by in-vehicle (either car or bus) 12V DC power through the power inverter. The Arduino circuit is connected to the Jetson, and when the vehicle's power is on, it will auto boot up the system.

7.6.3 *Parameter Settings*

Several key parameters needed to be set properly: SSD detector confidence threshold, the number of frames for size regression, the number of frames for center regression, TTC threshold δ, TTC threshold φ, horizontal motion threshold α, horizontal motion threshold β, and Jetson power mode. Given that the SSD detector tended to have fewer false-positives than false-negatives (Ruimin Ke, Zhuang, et al. 2020), some false-positives can be filtered out at the tracking step, and more false-positives (if any) will be filtered out by the near-crash detection algorithm, we set the detection confidence threshold to be 0.3 – 0.5.

For the number of frames for size regression, we suggested setting them to be around 10 to 15 frames. This range was large enough to compensate for the bounding box noises and small enough to assume the target's motion is consistent. The number of frames for center regression can be a little larger to capture the horizontal motion better, and the suggested number was in the range of 15 to 20. For δ and φ, as defined by many previous studies, the TTC threshold for a near-crash was around 2 to 3 seconds, which was our suggested value for δ. And we found that setting

φ to about 2 to 2.5 times δ worked well. We suggested setting α to the range of [-1, -0.5] and β to [0.02, 0.1]. Jetson power mode was recommended to be set as Max-N to fully utilize its computational power, though our system still operated in real-time (but lower FPS) with Max-Q mode.

7.6.4 *Evaluation of Near-Crash Detection Accuracy*

Essentially, near-crash is a type of traffic anomaly. To evaluate the proposed method's accuracy, we used the evaluation process of the Traffic Anomaly Detection task (Track 4) of the 2020 AI City Challenge as the reference (Naphade et al. 2020). First, the task dataset has 100 video clips with some anomalies. It is unknown exactly how many anomalies are in the test dataset, but the number is between 0 and 100, as mentioned in the introduction to Track 4. Likewise, we made a local test dataset with 5000 video clips with 500 near-crash events. As aforementioned, the test videos were from online resources, dashboard cameras on private cars and transit buses. This dataset is not being published due to potential privacy and copyright issues. There is a plan to create such a video dataset for near-crash detection in the future.

We manually labeled all the near-crash events with their occurrence videos and times. As in AI City Challenge Track 4, we defined a true-positive (TP) as a predicted near-crash within 10 seconds of the true near-crash. A false-positive (FP) is a predicted near-crash that is not a TP for a near-crash. A false-negative (FN) was a true near-crash that was not predicted. We used the F1 score to evaluate accuracy. F1 score was the harmonic mean of the precision and recall, where the best value = 1 and the worst value = 0.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{2TP}{2TP + FP + FN} \qquad (7-8)$$

Sample near-crash detection results are shown in Figure 7-5. The top three rows were three vehicle-vehicle near-crashes, and the bottom two rows were two of the vehicle-pedestrian near-crashes. The bounding boxes turned red to indicate a predicted near-crash, while other detected road users had green bounding boxes. A few more sample detection results can be found in the video published on YouTube (https://www.youtube.com/watch?v=8qu-cNqfWkg).

As summarized in Table 7-1, our system correctly predicted 496 out of the 500 labeled near-crashes and missed just 4. It generated 8 FPs in the 5000 video clips. Based on Eq. (7-8), the final F1 score was 0.988, and the average processing speed with Max-N mode was about 18 frames-per-second (FPS). The performance was promising, considering that we intentionally included a variety of near-crash scenarios and some very challenging cases in the dataset. There were adverse weather conditions, nighttime situations, traffic congestion, urban/rural traffic scenes, and so on. It is worth mentioning that the 5000 video clips are from a lot of different cameras and the proposed system knew nothing about the camera parameters of any of these cameras. This result benefited from the near-crash detection method. It again highlighted the possibility for low-cost and highly efficient large-scale application of the edge computing system to partially fulfill the purposes of safety data generation, IV corner case collection, and collision avoidance.

We carefully examined the FN and FP cases and summarized the causes. One of the four FNs that the system missed was a vehicle-pedestrian near-crash at night on a rural freeway with no streetlight. The pedestrian violated traffic rules by crossing the freeway, and the driver did not see him until almost running into him. The pedestrian was entirely in the dark so that the object detector missed him. Though there were more FPs than FNs, we considered only 8 FPs out of 5000 video clips acceptable and encouraging given the tradeoff in the efficiency of the system. While the proposed near-crash detection method can compensate for bounding box size noise in most

cases, it was not perfect. In the fourth case (the fourth row) of Figure 7-5, right before the correct detection of this vehicle-pedestrian near-crash, there was a vehicle-vehicle FP caused by a significant error in vehicle size detection. It was included in our YouTube demo video. To further improve detection performance, a practical solution is to improve object detection by transfer learning with more data.

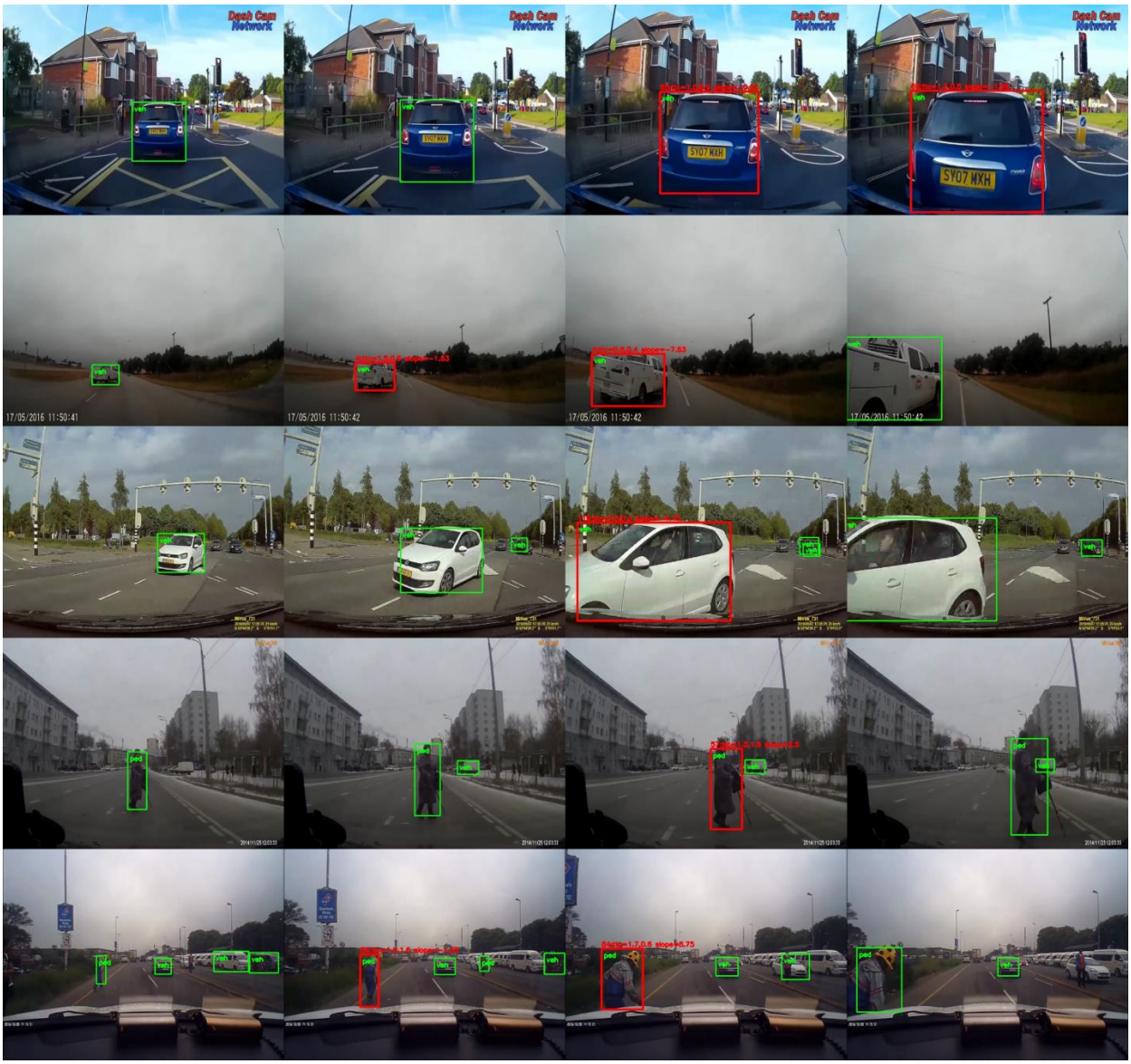


Figure 7-5 Sample near-crash detection results, where red bounding boxes indicate the potential conflict with the road user. Each row is a four-frame sequence of one near-crash event.

Table 7-1 Near-crash detection evaluation results

# of videos	# of events	TP	FP	FN	F1 Score	FPS
5000	500	496	8	4	0.988	18

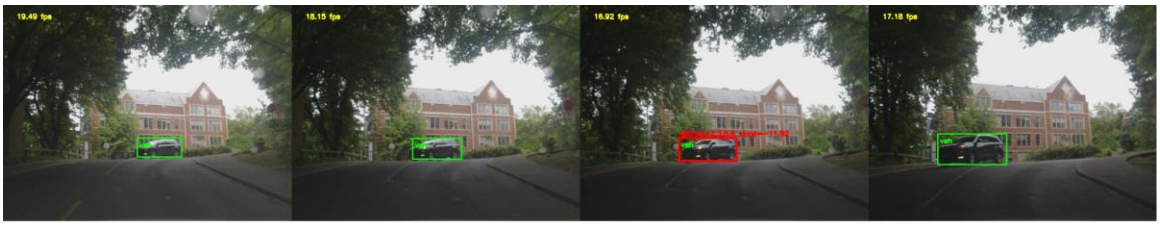
7.7 Practical Issues and Event Location Mapping

While in the local test, Jetson processed the local videos frame by frame; in the real-world test, different camera hardware, settings, and different software design resulted in different frame-reading speed and stability. For this reason, we used the separate video capture thread to ensure small delay in video frame reading. Also, when doing the regressions for near-crash detection, the system included the corresponding time for each value (height, width, and center) because the intervals between each pair of neighboring frames may not be uniform. Moreover, the camera type may influence system performance. Video captured on the bus had about 2-3 seconds of latency, while on the car, there was no observable latency. Latency in the video feed on the bus was due to using an IP camera connected via ethernet cables to the edge computing system. Jetson TX2 does not support auto boot-up. An external circuit driven by Arduino board was developed to automatically boot up the system when the bus power is on.

It was also observed that the GPS coordinates collected by the GPS receiver were not in any of the standard formats. It took some time to figure out there was a linear relationship between the raw GPS coordinate and the WGS84 coordinate format. The conversion is shown as follows in Eq. (7-9) and we hope this information will be helpful to others planning to use the same GPS receiver.

$$\begin{cases} Lat_{WGS84} = 1.666 \times Lat_{raw} - 31.30174 \\ Lon_{WGS84} = 1.666 \times Lon_{raw} + 81.25186 \end{cases} \quad (7-9)$$

Figures 7-6 and 7-7 present a sample of event data and GPS data collected in the real-world experiment by cars. Figure 7-6 included three near-crashes around the University of Washington (UW) campus. The first one was on campus with a car, and the second one was on the 15th Street in the University District with a King County Metro bus, and the last one was west of campus near University Village. All of the data associated with these events, including event video clips, TTC values, event types, GPS locations, and occurrence times were successfully recorded. Vehicle speeds were calculated using the GPS data. The GPS trajectory and event location data are valuable sources for analyses such as hotspot mapping and clustering. Figure 7-7 showed the trajectories and two near-crash events' spots on the OpenStreet Map with the corrected GPS coordinates during a trip on the UW campus. Figure 7-8 displayed all the near-crash events in October for Pierce Transit buses #230, #232, and #233, classified by bus identity and near-crash type (vehicle-vehicle or vehicle-pedestrian).



(a) Real-world tests on the University of Washington (UW) campus

(b) Real-world tests in the University District area, west of UW campus

(c) Real-world tests in the University Village area, east of UW campus

Figure 7-6 Three near-crashes captured around UW area.

Figure 7-7 The mapping of sample GPS trajectories (red curves) and near-crashes (blue circles) collected by cars.

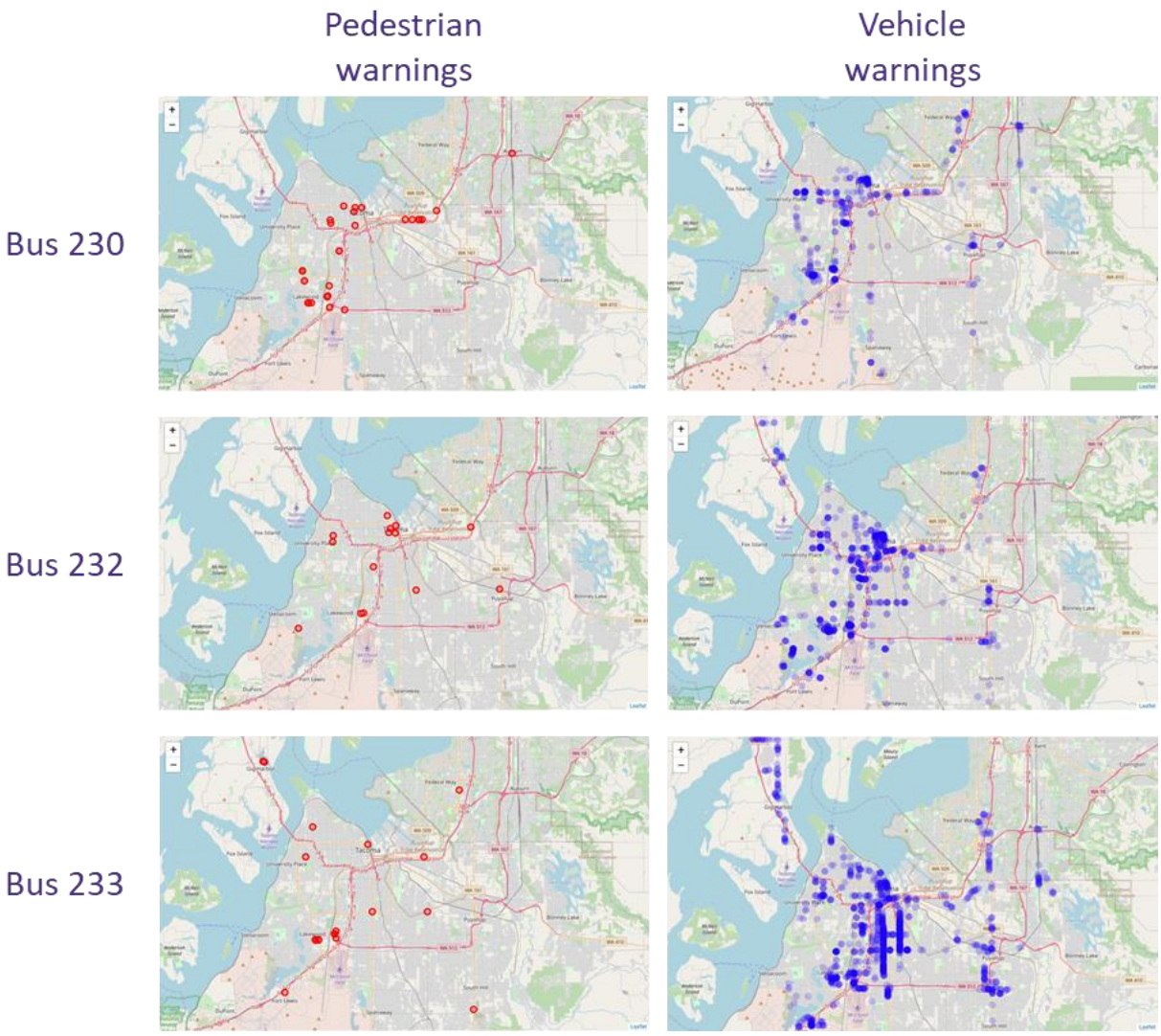


Figure 7-8 The distribution of pedestrian-related near-crashes (red circles) and vehicle-related near-crashes (blue circles) in October 2020 for three Pierce Transit buses.

PART IV: AERIAL SENSING FOR ROAD TRAFFIC

Chapter 8. REAL-TIME MACROSCOPIC TRAFFIC FLOW PARAMETER ESTIMATION FROM UAV VIDEO BASED ON ENSEMBLE CLASSIFIER AND OPTICAL FLOW

8.1 Overview and Contribution

The use of unmanned aerial vehicle (UAV) in traffic monitoring applications is becoming more and more popular. Compared to traditional monitoring devices, UAV is considered more cost-effective. Most traditional traffic monitoring devices capture traffic conditions at fixed and discrete locations, hence it requires many units to monitor a single road segment. In contrast, a few UAV(s) can cover a continuous stretch of roadway or even a traffic network. UAV's view point is another advantage for traffic monitoring: by achieving a top-view perspective, occlusions between road users in conventional surveillance videos are not likely to appear in UAV videos. Also, in surveillance video frames, pixels at different locations very likely correspond to different real-world sizes due to the camera tilt angle, but the corresponding real-world sizes of pixels in an UAV video frame are very close to one another, thus, camera calibration and estimation of real-world distances from an UAV video are easier. As a flexible platform with high mobility, UAV can also provide rapid reconnaissance and assessment of incident sites where few stationary sensors are placed for emergency response. Compared to manned aircrafts, UAVs have much lower operating and purchase cost. In addition, they fly closer to the ground, and thus have better robustness to adverse weather.

In addition to several practical concerns such as short battery life and privacy issue, the biggest technical challenge in automatic UAV-based traffic monitoring is the ego-motion issue, which can be generated either intentionally (e.g., by pilot) or unintentionally (e.g., by wind). The video background movement caused by UAV ego-motion makes the traditional vehicle detection and

traffic flow estimation methods designed for stationary surveillance videos not work well. To simplify the examination of UAV applications in traffic monitoring, UAV videos with little ego-motion were used for some preliminary studies. With the good foundation laid by these studies, more efforts have been put to address the ego-motion issue. While the objectives of monitoring tasks could be different, existing methods on addressing ego-motion issue can be divided into two categories: image registration (Angel et al. 2003; X. Cao et al. 2012; Y. Du et al. 2017; Shastry and Schowengerdt 2005; Yalcin et al. 2005; Yamazaki, Liu, and Vu 2008), and optical flow-based ego-motion estimation (X. Cao et al. 2012, 2014; Ruimin Ke et al. 2015, 2016; Yalcin et al. 2005; Yu and Medioni 2009). Image registration aims to turn the moving background into fixed background, thus makes it possible to apply traditional methods such as background subtraction. Image registration is probably the most intuitive method for addressing ego-motion issue and has been adopted by many previous studies. In these studies, a common assumption is that feature points mostly come from the video background rather than the vehicles (foreground). However, this assumption may be invalid in dense traffic situations, in which many feature points belong to the video background, hence the background motion could be inaccurately estimated.

Optical flow is a powerful method for video analysis due to its ability to extract the motion pattern. This method has been examined in UAV video processing. In previous studies, optical flow is normally combined with image registration or unsupervised learning to estimate the UAV ego-motion. It acts as an efficient feature extraction and feature matching tool to speed up the image registration (X. Cao et al. 2012; Yalcin et al. 2005). When optical flow collaborates with clustering algorithms, video background and foreground can be separated (Ruimin Ke et al. 2015, 2016). But the ways previous studies employ optical flow are still not suitable for dense traffic scenarios. Besides the aforementioned concern in image registration-based ego-motion estimation,

optical flow-based clustering would classify some interest points on the vehicles with low speed as background points, which may cause errors as well. These considerations motivate us to come up with the idea of integrating optical flow with supervised learning-based vehicle detection. In this way, vehicle pattern, which is not sensitive to traffic density, will be used for the separation of video foreground and background, instead of interest points or motion information.

This chapter proposes a framework for traffic flow parameter estimation from UAV videos that incorporates supervised learning-based vehicle detection methods and optical flow. This framework aims at filling the gaps in traffic flow parameter estimation from UAV videos with ego-motion. It is designed to work for both free flow and dense traffic. The framework is composed of four stages: The first two stages are designed for vehicle detection and the last two traffic flow parameter (speed, density, volume) estimation. Specifically, the first two stages in this framework combine Haar cascade and CNN as an ensemble classifier. Although Haar cascade classifier and CNN have been examined in UAV-based vehicle detection separately (Gaszczak, Breckon, and Han 2011; Guido et al. 2016; Xi Zhao et al. 2017), the proper combination of them in our study improves both the efficiency and accuracy of vehicle detection. Haar cascade acts as the region proposal method to largely reduce the number of region of interest (ROI) efficiently and CNN then determine the final detection results with its high accuracy. Stage three and four are designed as a general process for traffic flow parameter estimation in UAV videos. In other words, this process would still work even if our ensemble classifier were replaced by other vehicle classifiers.

To the authors' best knowledge, until now, no previous studies have achieved real-time traffic parameter estimation from UAV videos for both free-flow and congested traffic conditions. This study targets this issue by proposing a new framework, which is presented in the following sections. The contributions of the paper can be summarized as follows:

- A new real-time framework for traffic flow parameter estimation is proposed;
- Supervised learning is incorporated for the first time for traffic flow parameter estimation with background motion in the UAV video;
- This is the first framework that work for both free flow and dense traffic with respect to traffic flow parameter estimation in UAV video with background motion;
- A new method that addresses UAV height changes in real-time is developed;
- Stage three and four of the framework is a new general process designed for traffic flow parameter estimation;
- The ensemble classifier (Haar cascade + CNN) is examined for the first time for vehicle detection (in UAV video);
- A publicly available dataset containing 20,000 training samples for UAV-based vehicle detection is published as a benchmark at http://www.uwstarlab.org/research.html.

8.2 FRAMEWORK INTRODUCTION

The proposed framework contains four main stages (see Figure 8-1). The first two stages deal with vehicle detection and the last two about traffic flow parameter estimation. In the first stage, the Haar cascade classifier trained using randomly generated Haar-like features is applied as the region proposal method in order to determine the ROI. Haar cascade is very efficient thus can largely reduce the searching space for the final strong classifier. CNN is a powerful neural network model that has been proved effective in many detection tasks. In the second stage, CNN is tested and selected as the final vehicle classifier. In the third stage, the traffic real traffic motion is estimated by subtracting background motion from vehicle motion. Both the background motion and vehicle motion are estimated using KLT optical flow tracker based on the detection results. In the fourth stage, the traffic counts and estimated vehicle motion in pixels per frame are converted

to traffic density and speed with the conversion rate (converting pixel length to physical length) computed using reference markings. Then, traffic volume can be calculated using the basic traffic flow equation. Implementation details and further illustrations will be discussed in the next several sub-sections.

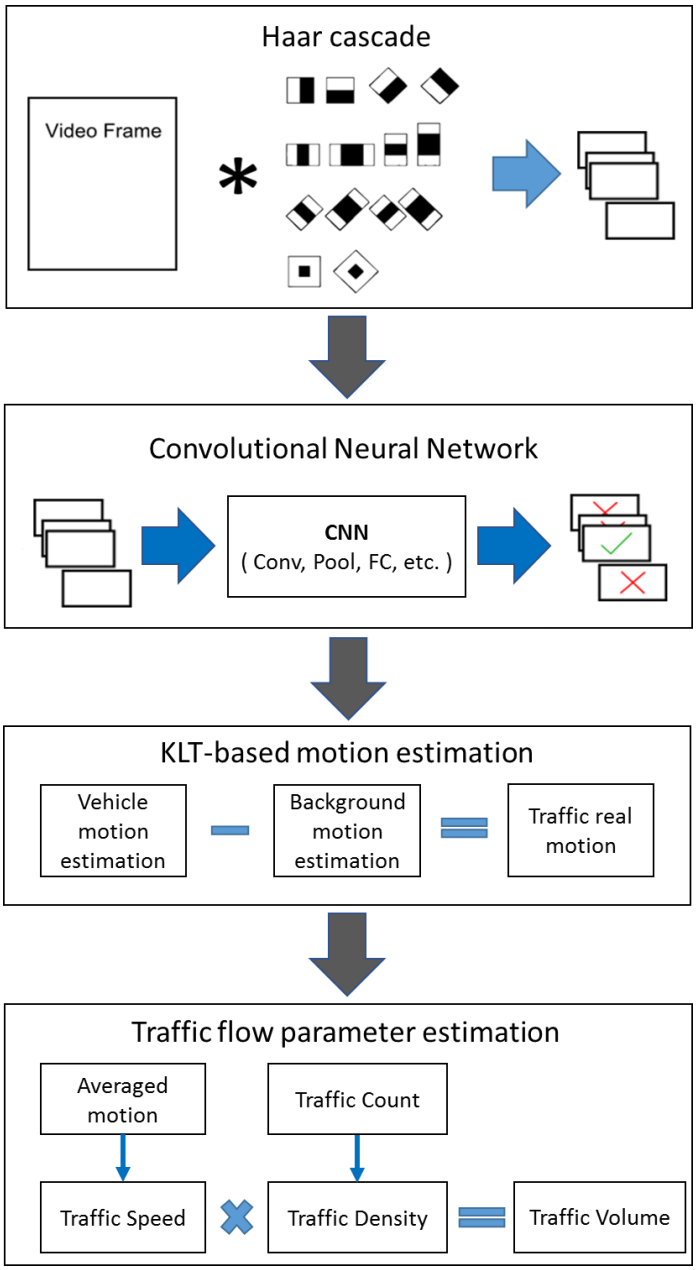


Figure 8-1 Summary of the four-stage framework.

8.3 DATA DESCRIPTION

Several UAV video clips taken from UAVs flying over different roadway segments were specifically used for training samples collection. In total 20,000 samples have been collected. Positive samples are mostly the top-view of cars. Only a small portion of positive samples include other types of vehicles such as buses and trucks due to the data availability, more trucks and buses will be collected in the future with more UAV videos taken. Negative samples are randomly cropped background images. Both the positive and negative samples were manually cropped from the training videos in the original size of 60 × 40 (width × height). In the Haar cascades training, the sizes of the samples were kept 60 × 40. However, in CNN training, the samples were scaled to the size of 20 × 13 (before any pooling action) in order to reduce the dimensionality of the input feature vector from 2400 to 260, thus to speed up the training processes. The dataset was split into 18,000 samples for training and 2,000 for testing. It was composed of 40% positive samples and 60% negative samples. This split ratio was selected based on the consideration that background patterns had more variations than vehicle patterns. It is worth noting that the images for training the vehicle classifiers in this study were collected by the authors and it is available at http://www.uwstarlab.org/research.html. This dataset is one of the first publicly available datasets for UAV-based vehicle detection.

8.4 STAGE 1: HAAR CASCADE CLASSIFIER FOR REGION PROPOSAL

Originally, Haar cascade classifier was employed as a statistical approach to handle the large variety of human faces. Haar-like features and AdaBoost learning algorithm are the two major components of the approach.

In this study, the Haar cascade classifier was trained using OpenCV 2.4.12 library. Besides setting the training image size to 60 × 40 as aforementioned, several other key parameters need to be set in the Haar cascade training process. First, the number of training samples for each stage needs properly set. The basic idea here is not to train too few stages which would make the region proposal effect not significant in terms of reducing candidate windows, but too many stages would very likely filter out some true positives. Based on this consideration and the total number of samples, each stage took 2,500 samples for training (1,000 positive samples and 1,500 negative samples), thus resulted in 7 stages.

Another two parameters needed for the training are min hit rate and max false-positive rate per stage. Since Haar cascade is the region proposal method in this study, min hit rate should be set close to 1 to make sure we recall all vehicles. The max false-positive rate per stage should be set not larger than 0.5. But if it is too small, say close to 0, the training time and the chance of overfitting would both sharply increase. As long as our recall is high, it is totally acceptable that each stage of our Haar cascade is just slightly better than a random classifier. Thus, in our training, these two parameters are set to 0.999 and 0.5. Non-maximum suppression is adopted as the last part of stage 1 to further reduce the candidate windows CNN needs to examine.

8.5 Stage 2: Convolutional Neural Network for Vehicle Detection

With the regions of interest proposed by Haar cascade in the first stage, a convolutional neural network, or CNN, is designed as the final classifier for vehicle detection. In this way of combination, the high efficiency of Haar cascade and the high accuracy of CNN are well utilized, thus can enable real-time vehicle detection with high detection rate. This is the first time that the ensemble classifier (Haar cascade + CNN) is examined in UAV-based vehicle detection. CNN has a typical layer called convolutional layer, which is the core building block of it. Compared to fully-

connected (FC) multi-layer perceptron (MLP) neural network, CNN has fewer parameters because of convolutional layer's local connectivity, thus the chance of overfitting can be reduced. Moreover, convolutional layer captures more representative features, particularly for image inputs. Another type of layer in CNN architecture is called pooling layer, which performs downsampling operation along the spatial dimension. Convolutional layer and pooling layer enable more effective feature selection and more efficient learning of features at different scales. FC layer and activation function are still important components of CNN architecture with respect to image classification problems.

In our study, CNN was developed using Keras in python and trained on an Nvidia GTX 1080 GPU. With a trial and error process, the architecture of our CNN was chosen to contain two convolutional layers, one pooling layer and one hidden FC layer (see Figure 8-2). The two convolutional layers have a same dimension of $32 \times 2 \times 2$ with sigmoid activation function; then the pooling layer is added to downsample the second convolutional layer's outputs by a scale factor of 2; and the FC layer with 128 nodes is added between the pooling layer and the final outputs. Compared to other popular CNNs such as AlexNet or VGG (Krizhevsky, Sutskever, and Hinton 2017; Simonyan and Zisserman 2014) with deeper structures and more output nodes, the proposed CNN structure is lightweight with much fewer layers and parameters. This is motivated by our requirement for real-time operation and a smaller number of categories (i.e., vehicle and background). It is found that two convolutional layers can already satisfy the accuracy requirement. It is worth noting that there is no pooling layer in between the two convolutional layers. This is because the training and testing losses turn out to be higher while the overall detection speed is not significantly improved if adding the pooling layer. Based on our tests and analyses, the increased losses are mainly caused by the small image size after pooling. Since the

dimension of our image samples is 20×13, if they are downsampled by the pooling layer, the features extracted by the second convolutional layer would not be as representative.

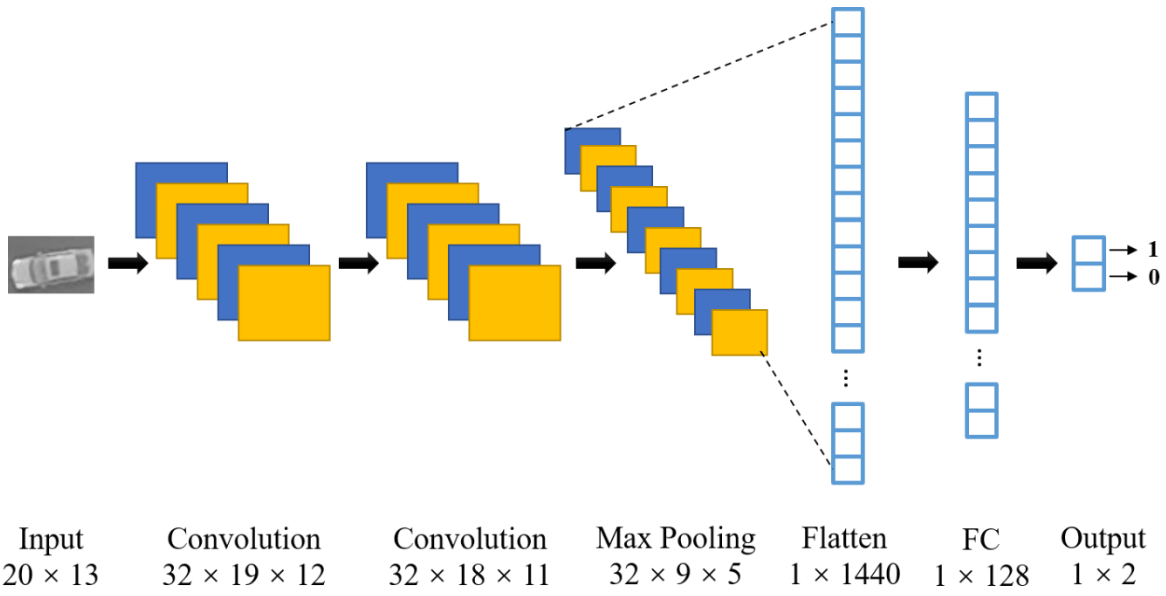


Figure 8-2 The proposed lightweight CNN model for vehicle detection in UAV video.

The training of CNN was done on 18,000 samples and testing on 2,000 samples. RMSprop (Root Mean Square Propagation) (Tieleman and Hinton 2012) was selected as the optimizer because of its better performance than others like traditional SGD (Stochastic Gradient Descent) (Bottou 2010) in similar cases based on experience and tests. The batch size for optimization was set to 30. Our CNN vehicle classifier reached 99.55% classification accuracy on the test data in 100 epochs of training, which was very encouraging. The model accuracy curves during the training process are shown in Figure 8-3.

As aforementioned, the top figure of Figure 8-4 shows the candidate windows proposed by Haar cascade classifier. It can be seen there are still some false-positives, but compared to letting the strong classifier (i.e., CNN in our framework) slide the whole frame in different scales, the

number of candidate windows have been largely reduced by Haar cascade. CNN is then applied to check all the candidate windows and gives out the final vehicle detection results. One example frame of the vehicle detection results is shown in the bottom figure of Figure 8-4.

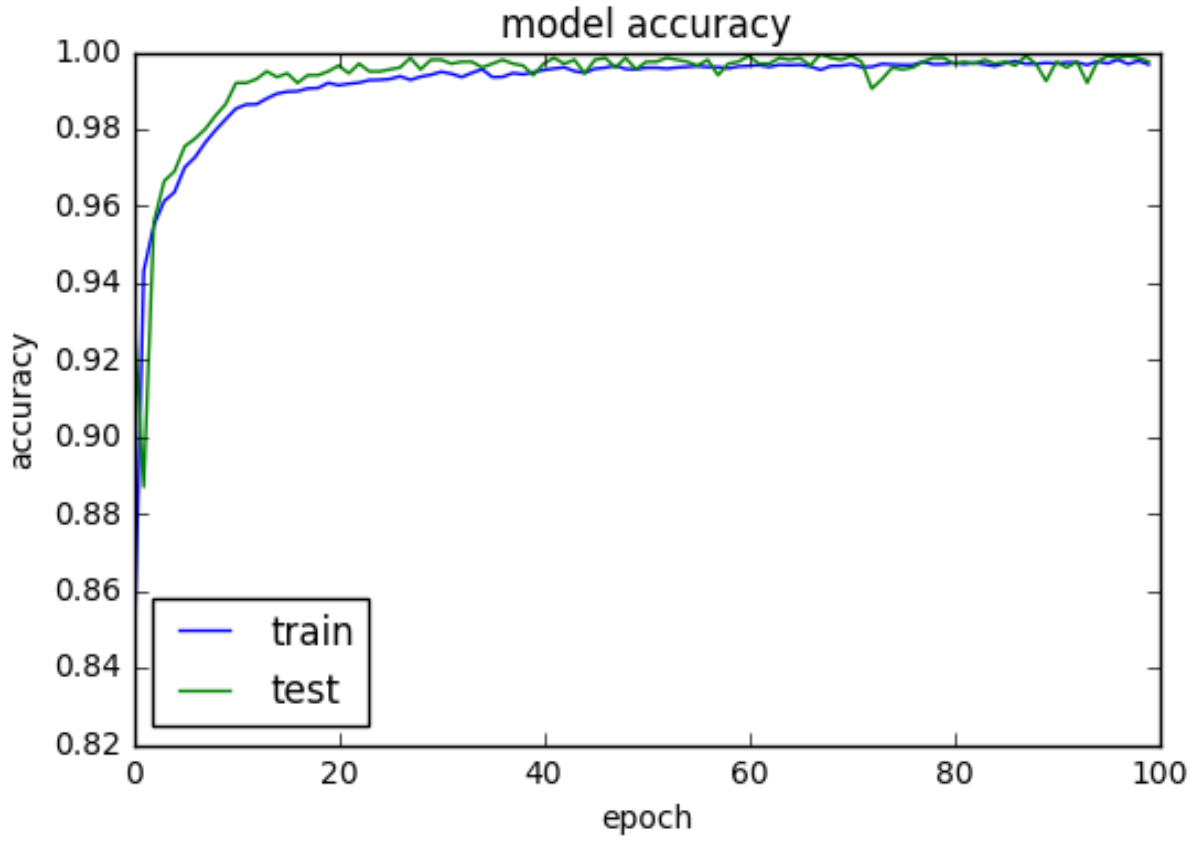


Figure 8-3 CNN model accuracy curves (train and test) during the 100-epoch training process.

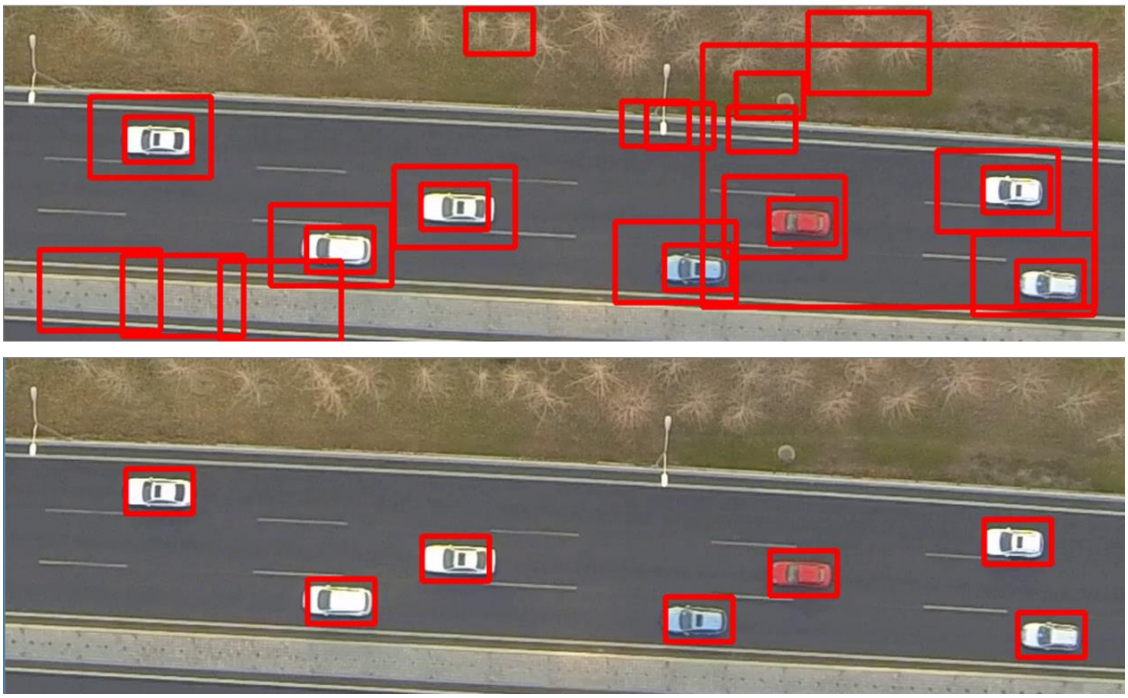


Figure 8-4 Haar cascade classifier acts as the efficient region proposal method (top) and then CNN does the final detection (bottom) by examining far less candidate windows than sliding the whole frame.

8.6 Stage 3: KLT-based Motion Estimation

With the detection results obtained, stage 3 and 4 define a general process for traffic motion estimation and traffic flow parameter estimation. To maximize the UAV's view point benefit, we assume an orthographic camera projection which is an approximation with a downward facing UAV camera. This general process would work with any supervised learning-based vehicle detector to complete traffic flow parameter estimation from UAV videos. Motion estimation is often based on object tracking and a known video frame rate. However, in UAV videos with background motion, many methods such as Kalman filter and particle filter cannot achieve accurate traffic motion estimation even if they may achieve multiple-vehicle tracking. This is due to their inability to estimate ego-motion (background motion). KLT method (Lucas, Kanade, and

others 1981) is a tracking method based on interest point, thereby it has the ability to estimate background motion in light traffic conditions (X. Cao et al. 2012). But with relatively denser traffic, directly applying KLT tracker would lead to large errors in motion estimation. Our efficient Haar + CNN vehicle detection process is particularly designed to addresses this issue.

The vehicle detection results split a video frame into two types of regions: traffic (inside the detection windows) and background (outside the detection windows). Hence, KLT can be applied to estimate vehicle motion and background motion after CNN detection. In Figure 8-5, the top image shows the motion-vectors extracted inside detection windows and the bottom image outside detection windows. The average of all the motion-vectors in the same category (inside or outside detection windows) represents traffic motion (with ego-motion added) and background motion, respectively. Suppose $\overrightarrow{mt_i}$ and $\overrightarrow{mb_j}$ denote the *i*-th motion-vector extracted for traffic and the *j*-th motion-vector for background, respectively, the true traffic motion $\vec{m}$ in pixels per frame is calculated as follows in Eq. (8-1):

$$\vec{m} = \frac{\sum_{i=1}^{Nt} \overrightarrow{mt_i}}{Nt} - \frac{\sum_{j=1}^{Nb} \overrightarrow{mb_j}}{Nb} \qquad (8-1)$$

where *Nt* is the total number of motion-vectors extracted for traffic and *Nb* is for background.

It is worth noting that the way we calculate $\vec{m}$ in Eq. (8-1) using the mean values of $\overrightarrow{mt_i}$ and $\overrightarrow{mb_j}$ may not be very accurate in the cases where there are quite a few outliers in the process of motion-vector estimation. These cases could be caused by low video quality, low lighting conditions, etc. In such cases, we suggest using median values of $\overrightarrow{mt_i}$ and $\overrightarrow{mb_j}$ to calculate the true traffic motion $\vec{m}$ instead of mean values due to median's better robustness to noise.

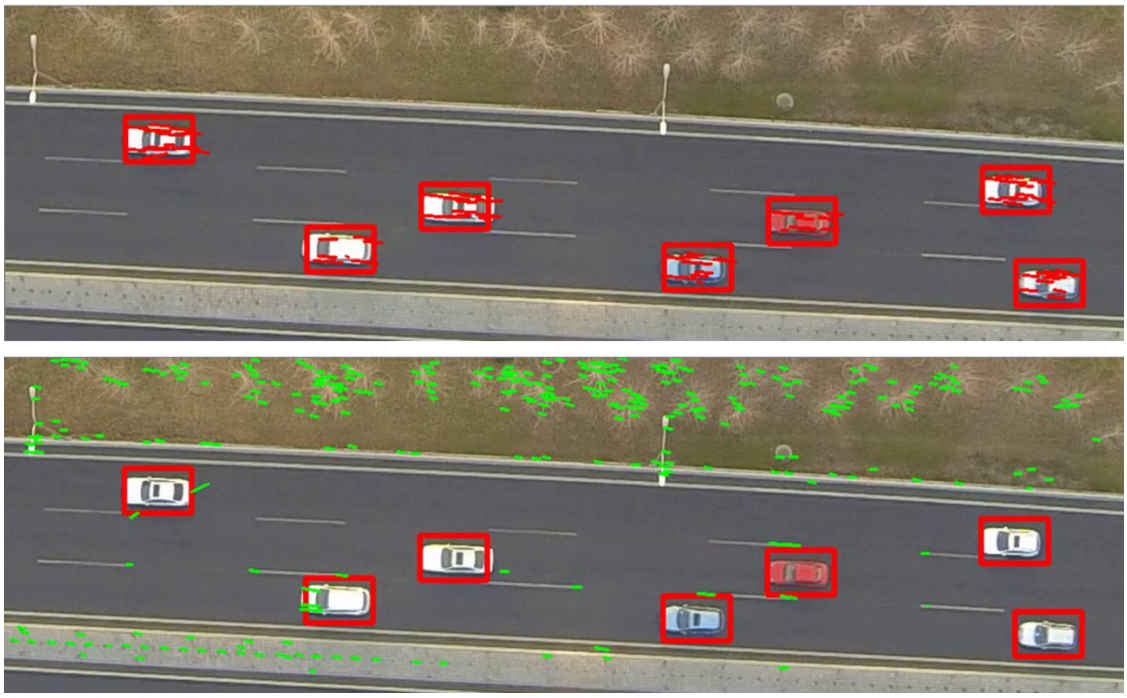


Figure 8-5 The proposed method for traffic motion estimation using KLT tracker. Based on the detection results, vehicle motion (red dots at the top figure) and background motion (green dots at the bottom figure) can be estimated. The motion-vector representing the traffic real motion in pixels per frame is computed by subtracting average background motion from average vehicle motion.

8.7 Stage 4: Traffic Flow Parameter Estimation

In transportation engineering, speed, density and volume are the most important three parameters to describe traffic flow, and their relationship is given by Eq. (8-2):

$$volume = speed \times density \times NOL \qquad (8-2)$$

where *NOL* denotes number of lanes. With the vehicles detected in stage 2 and traffic motion estimated in stage 3, traffic density and speed can then be calculated with reference markings. Density is defined as vehicle counts per lane per unit freeway length (mile, kilometer, etc.). Speed

will be converted from pixels per frame to miles/kilometers per hour. Reference markings, such as standard school buses and lane markings, are able to avoid complicated camera calibrations and often sufficient for computing real-world sizes in UAV videos.

At a starting frame, we assume a known object to have a real-world size l_1 and pixel size l_2, thus the conversion rate is $r = l_1/l_2$, and the road segment length L pixels. The initial pixel lengths of l_2 and L are measured offline. Frame rate fr is assumed to be constant during monitoring. If real-time height information is recorded in the data, it would be helpful to determine if the conversion rate needs to be changed. However, most UAV video datasets do not contain real-time height information, so we proposed a method to estimate whether the height has significant changes. Normally, UAV tends not to change its flight height during a short time, thus it is unnecessary to check possible height changes in every pair of consecutive frames. The pixel to real-world size conversion rate in the first frame is denoted r_1, and r_n for the n-th frame f_n. Then, we compare the mean window width of detected vehicles in frame f_n and f_{n-1} to see if there is a significant difference using t-test with 0.05 as the threshold for statistical significance. If no significant changes detected, we will continue using the rate last updated for traffic flow calculation; if yes, r_n will be updated as in Eq. (8-3):

$$r_n = r_{n-1} \times \frac{w_n}{w_{n-1}} \tag{8-3}$$

where w_n is the mean width of detection windows in f_n, and w_{n-1} in f_{n-1}. With these definitions and calculations above, traffic speed and density are calculated using Eq. (8-4) and (8-5):

$$speed = \vec{m} \times fr \times r \tag{8-4}$$

$$density = \frac{N}{L \times r \times NOL} \qquad (8-5)$$

where *N* is the number of vehicles detected in the current frame and *r* is the last updated real-world to pixel conversion rate. Another basic traffic flow parameter, i.e. volume, is calculated using Eq. (8-2).

8.8 Experimental Results

8.8.1 *Vehicle Detector Evaluation*

To analyze the performance of the proposed framework, we first tested our vehicle detector's performance and compare them with the state-of-the-art vehicle detectors developed for UAV video data (X. Cao et al. 2011; Gaszczak, Breckon, and Han 2011; Guido et al. 2016; Ruimin Ke 2016; Xi Zhao et al. 2017). Specifically, Haar cascade, CNN, MLP, HOG + SVM, Haar cascade + MLP, and Haar cascade + CNN (proposed ensemble classifier) were tested and compared using our collected vehicle samples.

The Haar cascade classifier was trained using OpenCV 2.4.12 build-in functions, and the CNN classifier was trained using a python program developed by our team. This program made use of the Keras deep learning library and ran on the Theano backend. Both Haar cascade and CNN classifiers were stored as xml files. Another standalone python program was then developed to implement the entire detection and estimation process. This program first loaded the two xml files and read video frames one by one as inputs, then it ran the proposed pipeline and outputted traffic flow density, speed, and volume. For MLP and SVM training as well as performance evaluation we adopted Keras and Sklearn libraries. CNN and MLP were trained on Nvidia GTX 1080 GPU

while other training, detection, and data management programs were implemented on an Intel Core i7-6700 CPU.

All the detectors were trained using the same training samples (i.e., the 18,000 vehicle samples cropped from UAV videos) and tested on the remaining 2,000 samples. Besides accuracy evaluation, efficiency of these detectors were tested on an UAV video clip with 1000 × 300 resolution. Though video resolution may impact processing speed of a detector, the result here presents the relative order with respect to detector efficiency, which is not likely to change.

Precision, which is the fraction of relevant instances among the retrieved instances, and recall, which is the fraction of relevant instances that have been retrieved over total relevant instances, are commonly used in detectors performance evaluation. They are defined as in the following equations,

$$precision = \frac{TP}{TP + FP} \tag{8-6}$$

$$recall = \frac{TP}{TP + FN} \tag{8-7}$$

where TP is short for true-positive, FP for false-positive, and FN for false-negative. Precision and recall both reach their best value at 1 and worst at 0.

In the 2,000 test samples, 814 of them are labeled as positive samples (vehicles) and 1,186 negative samples (backgrounds). Detailed detection performance evaluation results are presented in Table 8-1. Previous studies that used Haar cascade for UAV-based vehicle detection normally dealt with relatively simpler traffic scenes, and had fewer samples for training. For specific purposes, some study even just included one individual vehicle in the training samples (Guido et

al. 2016). While Haar cascade performed well in previous studies, with more variety in vehicle color, vehicle type, etc. appearing in our dataset, Haar cascade generated many FPs as shown in Table 1. However, Haar cascade retained a good recall (0.957) and very fast processing speed (81 fps). Standalone CNN was the best detector in terms of accuracy, which generated just 4 FPs and 1 FNs. But it is very slow in computing with 0.29 fps processing speed. MLP achieved good precision and recall rates, i.e., 0.845 for precision and 0.856 for recall. However, they were not comparable to CNN or the ensemble classifier in term of either precision or recall, let alone its slow processing speed. Haar cascade + MLP method (Ruimin Ke 2016) had a similar processing logic as the proposed ensemble classifier, thus the processing speed was fast. But the precision and recall values were at the same level with standalone MLP. HOG + SVM is another popular detector that has been applied in different tasks, it was examined by Cao et al. (X. Cao et al. 2011) in UAV-based vehicle detection. They achieved a very high precision value with much less FPs generated than Haar cascade or MLP. Its processing speed was also faster than CNN or MLP. But its FN rate was high thus led to a recall rate lower than 0.80. Our ensemble classifier achieved 0.995 precision, 0.957 recall, and 67 fps processing speed. The excellent performance of our detector was beneficial from the combination of Haar cascade and CNN. Haar cascade generated many FPs but they were eliminated by CNN at stage 2, thus resulting in very high precision rate. Then, as the recall rates of both Haar cascade and CNN stayed high, the combination of them still had a high recall. The processing speed of the ensemble classifier was very close to that of Haar cascade or Haar + MLP, and was much faster than other detectors. As aforementioned, this was because the final strong classifier (i.e., CNN) examined much less candidate windows than sliding the whole frame.

Table 8-1 Detector performance evaluation and comparison results

	Haar Cascade	**CNN**	**MLP**	**Haar + MLP**	**HOG + SVM**	Ensemble Classifier
Num. False-Positive	386	4	128	120	29	**4**
Num. False-Negative	35	1	117	122	179	**35**
Precision	0.669	0.995	0.845	0.852	0.956	**0.995**
Recall	0.957	0.999	0.856	0.850	0.781	**0.957**
Processing Speed (fps)	81.00	0.29	0.44	69.00	3.85	**67.00**

8.8.2 *Traffic Flow Parameter Estimation Results*

In the experiment, in total about thirty minutes' video clips were tested. Specifically, two representative 400-frame video footages were manually examined in detail frame by frame as examples to evaluate the effectiveness of the framework. The two video clips were not used for any training samples collection. Video #1 was taken by a UAV moving over a freeway segment, monitoring a three-lane freeway with smooth traffic flow. Video #2 was another video clip taken over an urban arterial where the traffic was dense. Continual background motion that caused by ego-motion including cruising, rotation and vibration existed in the test videos.

Figure 8-6 shows some randomly selected sample frames in the two video clips with the detection windows and motion-vectors marked. The three frames on the left are from video #1 and right video #2. In practice, reporting instantaneous traffic flow parameters frame by frame is not meaningful, therefore averaged speed, density and volume were calculated and listed in Table 8-2. The average traffic speed in video #1 was 31.5 mph, which was reasonable for an urban arterial in Beijing, China. The average density was 31.9 pc/mi/lane (passenger cars per mile per lane) and the volume was 3010.6 pc/h (passenger cars per hour). The estimated speed of video #2 was much lower than that of video #1 and the density was higher. From Table 8-2, the speed of video #2 was

only 1.7 mph, and the density was 45.5 pc/mi/lane. The volume then turned out to be 309.4 pc/h, which was much lower than that of video #1.

8.8.3 *System Performance Evaluation and Analysis*

To validate the traffic flow parameter estimation accuracy, vehicle speed and vehicle count were selected as the metrics. Ground truth speed data was manually collected using an on-screen pixel measurement tool. The speeds of individual vehicles were measured over intervals of five consecutive frame pairs. We considered five frames reasonable as the measuring interval. Basically, if the interval was too small, the manual measurement error of speed would be large, and if it was too large, the data resolution would deteriorate. Five frames took about 0.2 seconds, during which the traffic speed could still be viewed as constant. Hence, detected speeds was averaged for each five consecutive frames and used for the speed accuracy analysis. Ground truth vehicle count information was collected by manually counting the vehicles frame by frame.

Plots in Figure 8-7 showed the estimated count, ground truth count, estimated speed and ground truth speed for the two cases. The average speeds, counts and accuracies were presented in Table 8-2. The estimation accuracies were very high for video #1, reaching 97.0% and 95.8% for speed and vehicle count estimation, respectively. It can be seen the estimation was generally more accurate in video #1 than #2. We examined the frames and found that this was mainly caused by the traffic composition. In video #2, more buses and trucks were included, but as aforementioned, the samples for classifier training contained very few buses and trucks due to the amount of training videos available. Moreover, the road surface color was very similar to some vehicles in video #2, thus making it very challenging for vehicle localization tasks. But in general our system still achieved good estimation performance in the second case, resulting in 92.4% accuracy for speed estimation and 85.1% for vehicle count estimation.

Vehicle count estimation accuracy strongly relied on the detector's performance since the count was the number of detection windows in each frame. Speed estimation accuracy also tended to be influenced by the vehicle detection results in the proposed framework. This was because every false-positive or false-negative literally increased not only the count estimation error but also the traffic motion estimation error. According to the proposed processing logic, false-positives would include motion-vectors belonging to the background into traffic motion. Likewise, false-negatives would include motion-vectors which belong to traffic into background motion. Thus, it was reasonable to see a higher speed estimation accuracy in video #1 since its count estimation was more accurate (see Table 8-2). Considering that volume was a product of density and speed, this finding actually showed the importance of vehicle detector performance in the proposed framework.

While our proposed system achieved good traffic flow estimation for both free flow and congestion, one interesting fact worth mentioning was that, the difference between speed estimation accuracy and count estimation accuracy was larger in congestion case than free flow (7.3% vs 1.2%). This was because false-positives and false-negatives would cause less errors to speed estimation in UAV videos with dense traffic. Specifically, in congestion scenarios, vehicles have low or even zero speed. In other words, from the UAV's view, the background motion and traffic motion are closer to each other than uncongested scenarios. Hence, false detections or missed detections would have less influence on motion estimation in our proposed framework.

As real-time traffic information is so important for traffic control or route guidance, the processing speed of our method is considered as a critical performance measurement. The experiments were conducted on a desktop computer with an Intel i7-6700 CPU @ 3.40GHz processor and 20G of memory. Under current parameter settings, the average processing speeds

for the two videos were 29.7 fps and 25.4 fps, respectively. Considering the frame rates of most videos are no more than 25 fps, real-time traffic flow parameter estimation from UAV videos can be supported.

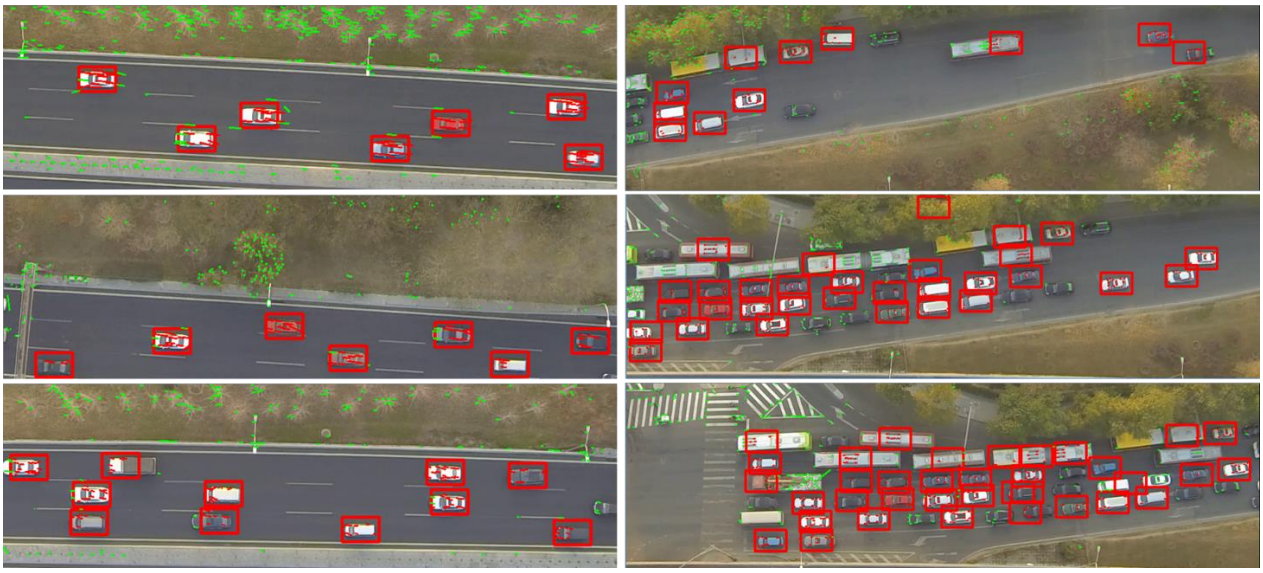
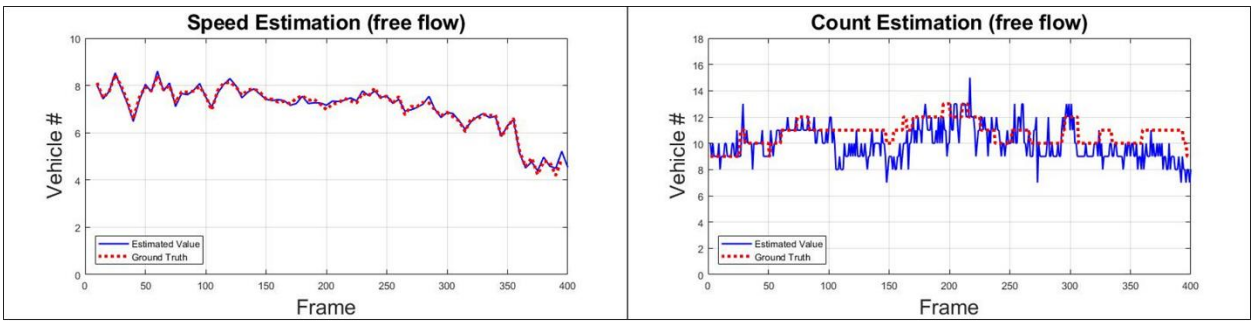


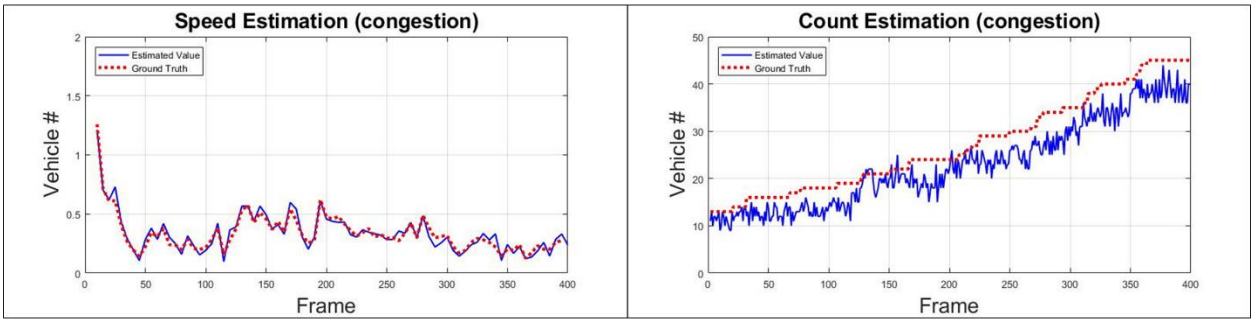
Figure 8-6 Sample frames in video #1 (left, free flow) and video #2 (right, dense traffic) showing the detection and motion estimation results. Continual ego-motion including cruising, rotation and vibration exist in either video clip.

Table 8-2 Summary of estimated traffic flow parameters and performance evaluation results

	Test Video #1	Test Video #2
Total Frames	400	400
Dimension of ROI	1000 × 300	1220 × 350
Traffic Condition	Free Flow	Congested
Estimated Speed (mph)	31.5	1.7
Estimated Count (per frame)	9.9	23.0
Estimated Density (pc/mi/lane)	31.9	45.5
Estimated Volume (pc/h)	3010.6	309.4
Speed Estimation Accuracy (%)	97.0	92.4
Count Estimation Accuracy (%)	95.8	85.1
Processing Speed (fps)	29.7	25.4



(a)



(b)

Figure 8-7 Plots presenting the accuracy of traffic flow speed estimation and vehicle count estimation. Subfigure (a) presents the speed estimation results and count estimation results in video #1; subfigure (b) shows the speed and count estimation results in video #2. Ground truth and estimated value overlay in each of the four plots, where the dotted red curves represent the ground truths and solid blue curves are the estimated values.

Chapter 9. ADVANCED FRAMEWORK FOR MICROSCOPIC AND LANE-LEVEL MACROSCOPIC TRAFFIC PARAMETER ESTIMATION

9.1 Overview and Contribution

Unmanned aerial vehicle (UAV) has demonstrated great potentials for traffic monitoring in a series of transportation studies due to its high flexibility, wide monitoring range, top-view perspective, and low cost, compared to conventional traffic sensors (Angel et al. 2003; Barmpounakis, Vlahogianni, and Golias 2016; Coifman et al. 2004; Y. Du et al. 2017; Freeman et al. 2019; Kanistras et al. 2015; H. Zhou et al. 2014). With such benefits, the prospect of UAV for cost-effective and advanced traffic management is brilliant. However, new algorithms and frameworks are needed to turn the data collected by UAV into useful resources. One of the most common UAV sensors is the video camera. While there are many existing algorithms in the area of video-based traffic detection (Ruimin Ke, Lutin, et al. 2017), most of them are designed for roadway surveillance cameras at fixed locations (Ruimin Ke, Pan, et al. 2017). Hence, it is difficult to apply them to UAV video due to the nature of UAV's ego-motion, which causes irregular video background movements.

In preliminary studies, researchers explored the possibilities of using UAV for traffic monitoring with a focus on either UAV videos with static backgrounds (Kaufmann et al. 2018; M. A. Khan et al. 2017; McCord et al. 2003; Salvo, Caruso, and Scordo 2014) or some straightforward research tasks (e.g., vehicle detection) that could be completed without handling the UAV ego-motion issue (Ammour et al. 2017; X. Cao et al. 2011; Yongzheng Xu et al. 2017). Recently, with the needs and vision for advanced ITS applications and significant progress in video processing techniques, some new studies have been conducted on addressing this issue (X. Cao et al. 2014;

P. Chen et al. 2017; Ruimin Ke 2016; Ruimin Ke et al. 2015, 2018; Ruimin Ke, Li, et al. 2017; Shastry and Schowengerdt 2005). These achievements in handling UAV ego-motion have laid the foundation for more sophisticated research on UAV-based traffic sensing, where a primary topic is traffic parameters estimation from moving UAV platforms (Ruimin Ke 2016; Ruimin Ke et al. 2015, 2018; Ruimin Ke, Li, et al. 2017; J. Li et al. 2019; Shastry and Schowengerdt 2005).

While most previous studies on UAV-based traffic parameters estimation focus on extracting aggregated macroscopic traffic data, there has been extensive attention on higher-resolution traffic data such as microscopic traffic parameters and lane-level macroscopic traffic parameters due to the fact that they can help us deeply understand traffic patterns and individual vehicle behaviors (Barmpounakis et al. 2017; Kim et al. 2019). Despite the recent accomplishments, little existing research realizes the estimation of microscopic and lane-level macroscopic traffic parameters from moving UAVs. UAV video, as a cost-effective data source for future transportation, needs to be examined for its availability of providing such data. The key limitations in the existing research can be summarized into three aspects: *1)* instead of making use of long-term temporal information, existing traffic parameters estimation methods for UAV sensing are usually based on two-frame video information, which reduces the reliability and applicability of the application; *2)* there are few processing pipelines to extract sufficient lane information from UAV videos for traffic parameters estimation; *3)* there are few modules for UAV videos to automatically combine and convert vehicle information, lane information, motion information, and traffic flow theories into fundamental traffic parameters.

In this chapter, we test the availability of UAV video for microscopic and lane-level macroscopic traffic parameters estimation by proposing an effective processing framework. Specifically, three functional modules consisting of multiple processing streams, algorithms, data

structures, and the interconnections among them are carefully designed within the framework with the consideration of UAV video features and traffic flow characteristics (S. Feng et al. 2018). This research pushes off the boundaries of the applicability of UAVs in intelligent transportation systems and has an enormous potential to support applications like advanced traffic sensing, traffic enforcement, and post-disaster traffic management.

The proposed framework targets a critical problem in the field of intelligent transportation systems: increasing the types of fundamental traffic parameters that can be automatically extracted from an emerging sensing platform – UAV. The design of the framework targets balancing efficiency, accuracy, and robustness with a combination of existing algorithms and proposed new algorithms. To our best knowledge, the contributions of this study are summarized as follows:

- The development of the framework is among the first efforts to examine moving UAV's availability for microscopic and lane-level macroscopic traffic parameters estimation. Interactions among different modules and streams within the framework are designed to address background motions in UAV video and support reliable traffic parameters estimation.
- A new multiple-vehicle tracking method is specifically developed for UAV videos, which improves the data collection accuracy and enables the estimation of microscopic traffic parameters.
- An efficient and robust lane information extraction pipeline is built to obtain the number of lanes, lane positions, and lane lengths from moving UAV videos.

9.2 Related Work

The most recent studies in UAV-based traffic surveillance focus on vehicle detection (Ammour et al. 2017; Breckon et al. 2009; X. Cao et al. 2011; Gomaa, Abdelwahab, and Abo-

Zahhad 2018; J. Li et al. 2016; Najiya and Archana 2018; Rodriguez-Canosa et al. 2012; Teutsch and Krüger 2012; Tsao et al. 2018; Yongzheng Xu et al. 2017), vehicle tracking (X. Cao et al. 2014; Carletti et al. 2018; D. Du et al. 2018b; Gomaa, Abdelwahab, and Abo-Zahhad 2018; M. A. Khan et al. 2017; J. Li et al. 2016; Rodriguez-Canosa et al. 2012; Teutsch and Krüger 2012; Tsao et al. 2018), traffic pattern recognition (Kaufmann et al. 2018; M. Khan et al. 2018), and traffic parameters estimation (Barmpounakis et al. 2017; Y. Du et al. 2017; Ruimin Ke 2016; Ruimin Ke et al. 2015, 2018; Ruimin Ke, Li, et al. 2017; Kim et al. 2019; J. Li et al. 2019; McCord et al. 2003; Shastry and Schowengerdt 2005; J. Zhu et al. 2018). Vehicle detection and vehicle tracking are usually the initialization process for traffic pattern recognition and traffic parameter estimation. Thus, developing vehicle detectors and trackers with high efficiency, accuracy, and robustness is very important for advanced traffic surveillance tasks. While traditional vehicle detection in UAV videos tended to use background modeling or conventional machine learning with handcrafted features (Breckon et al. 2009; X. Cao et al. 2011; Gomaa, Abdelwahab, and Abo-Zahhad 2018; Rodr\'\iguez-Canosa et al. 2012; Teutsch and Krüger 2012; Tsao et al. 2018), more and more studies started to design or implement deep learning based vehicle detectors for UAV surveillance due to the high accuracy of deep neural networks in image classification and localization (Barmpounakis et al. 2017; Carletti et al. 2018; M. Khan et al. 2018; Najiya and Archana 2018; J. Zhu et al. 2018). Vehicle detection itself is able to determine traffic parameters like density without the need for motion analysis or vehicle tracking. Zhu et al. presented an enhanced single shot multibox detector (Enhanced-SSD) for vehicle detection with their own manually annotated data (J. Zhu et al. 2018). It helps collect traffic density with high accuracy.

However, in order for the estimation of other traffic parameters such as speed and volume, motion analysis or vehicle tracking is a must on top of vehicle detection. Great efforts have been

made in testing vehicle tracking methods for UAV videos and designing motion analysis frameworks for traffic parameters estimation. Most previous works implemented and tested popular tracking algorithms for UAV videos. Kalman filter, particle filter, and optical flow are the most widely adopted trackers or motion estimators in the existing literature (X. Cao et al. 2014; Gomaa, Abdelwahab, and Abo-Zahhad 2018; M. A. Khan et al. 2017; J. Li et al. 2016; Rodr\'\iguez-Canosa et al. 2012; Teutsch and Krüger 2012). Among these three, the particle filter is more appropriate than a naïve Kalman filter in real-world traffic scenarios due to the particle filter's non-linearity property (X. Cao et al. 2014). Optical flow, normally Kanade-Lucas-Tomasi (KLT) tracker, is a very flexible tracker and has been widely used in UAV-based applications. Lately, a study compared performances of the state-of-the-art trackers for UAV tracking (D. Du et al. 2018). Among them, simple online and real-time tracking (SORT), achieves the best overall performance (Bewley et al. 2016) considering tracking accuracy and real-world applications.

Traditional traffic parameter estimation research focused on using static UAV videos to extract macroscopic traffic flow parameters. One of the earliest studies was conducted by McCord et al. (McCord et al. 2003), in which many critical macroscopic traffic parameters were successfully estimated such as annual average daily traffic. Later on, research was conducted to address UAV ego-motion issues and estimate macroscopic traffic parameters at the same time. A pioneering study of this task was proposed by Shastry et al. (Shastry and Schowengerdt 2005). They developed a method that made use of image registration and motion information to achieve the estimation of basic traffic flow parameters. Recently, Ke et al. developed a couple of machine-learning-based methodologies to estimate aggregated traffic flow parameters (speed, density, and volume) and conducted thorough experiments in a variety of scenarios (Ruimin Ke et al. 2018; Ruimin Ke, Li, et al. 2017). In additional to macroscopic parameters, because of the needs for

higher-resolution traffic data, researchers have been exploring the possibility of UAV for microscopic traffic parameter estimation (Barmpounakis et al. 2017; Kim et al. 2019). For example, Barmpounakis et al. conducted a study in microscopic traffic parameter estimation from UAV video. In their research, naturalistic trajectory data from UAV video footage at a low-volume intersection and a pedestrian passage is extracted (Barmpounakis et al. 2017). However, these new studies on microscopic data were still based on static UAV videos. In this paper, we try to study the availability of moving UAV videos for high-resolution traffic parameters estimation (microscopic traffic parameters and lane-level macroscopic traffic parameters).

9.3 FRAMEWORK INTRODUCTION

The proposed framework contains three modules: core functional module, data storing module, and traffic parameters estimation module (see Figure 9-1). The core functional module has three processing streams for motion-vector processing, multiple vehicle detection/tracking, and lane information extraction, respectively. Each of them functions as an independent stream, but at the same time, has interconnections with one another. This design enables the automatic and robust extraction of multiple types of traffic parameters. In the core functional module chart of Figure 9-1, the rectangular green boxes represent the outputs that will be further processed within the framework.

The data storing module takes outputs from the core functional module as its inputs, and stores data into different data structures. The traffic parameters estimation module takes the organized data from the data storing module and estimates seven key traffic parameters. Based on traffic flow theories and the proposed preceding-vehicle determination process, the traffic parameters estimation module takes the organized data from the data storing module and estimates four key

microscopic traffic parameters including vehicle position, vehicle speed, space headway, and time headway, and the three lane-level macroscopic traffic parameters including traffic flow speed, traffic flow density, and traffic flow volume.

9.4 Core Functional Module: Multiple Vehicle Detection and Tracking

The core functional module has three main functional streams: 1) extracting and grouping motion vectors, 2) detecting and tracking multiple vehicles, and 3) extracting lane information (see Figure 9-1). In this subsection, we introduce our new multiple-vehicle tracking algorithm under the context of UAV traffic sensing. This algorithm is designed for vehicle tracking in moving UAV videos. It follows a detection-based tracking pipeline, which is composed of three steps: detection, prediction, and association. It not only achieves a high tracking rate but also improves vehicle detection precision and recall rates by effectively filtering out false positives (FPs) and false negatives (FNs). How the proposed tracking pipeline reduces FPs and FNs are introduced later in this section, and you can use Table 9-1 for quick reference.

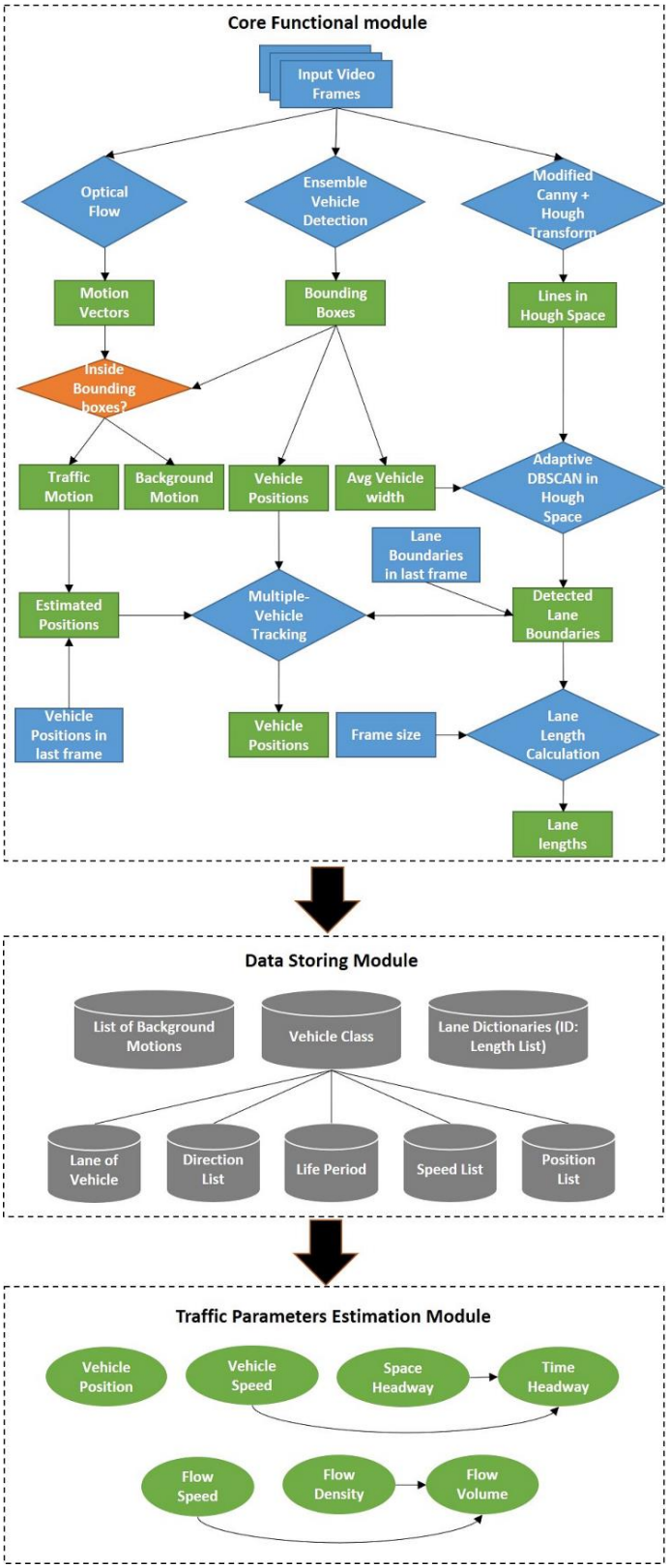


Figure 9-1 The proposed framework for lane-level macroscopic and microscopic traffic flow parameters estimation

Figure 9-2 is the diagram showing the key steps of the proposed detection/tracking algorithm. In the detection step, the state-of-the-art ensemble vehicle detector (Ruimin Ke et al. 2018) is implemented to find the locations and sizes of vehicles from a top-view perspective. The solid blue rectangular boxes in Figure 9-2(a) show the vehicle detection results in a frame. The dashed red boxes in Figure 9-2(b) show where the vehicle bounding boxes are in the previous frame. KLT optical flow tracker (Lucas, Kanade, and others 1981), which has demonstrated its great performance in UAV-based vehicle motion prediction, is selected as the motion prediction tool in the second step of the tracking. Optical flow motion vectors in a frame are classified into two groups: traffic motion vectors and background motion vectors, with the detected vehicle bounding boxes as the region classifier. Each vehicle's predicted motion is calculated by adding the median of the motion vectors in its bounding boxes to its position in the previous frame. (shown as dashed blue boxes in Figure 9-2(b)).

The third step is the association of detections with predictions. In order to better illustrate our association algorithm, the primary processes are summarized as pseudocode in Table 9-1. Detected vehicle boxes and predicted vehicle boxes are the two inputs to the association algorithm. The detected vehicle boxes are stored in a data list, and the predicted vehicle boxes are added to a data dictionary as the values. The dictionary's key is vehicle ID. The algorithm outputs another data dictionary which has the same structure with the predicted vehicle dictionary *pre_dic*, where the key is the vehicle ID and the value is the vehicle bounding box. This dictionary is denoted as *veh_dic* for the identified vehicles, which contains all the updated vehicle positions in the current frame. Intersection-over-Union (IoU) is used to determine whether every prediction box can find a match in the detected boxes list. IoU is selected as the indicator for the association because of its appropriateness for UAV-based vehicle tracking. The appropriateness comes from (1) its low

computational complexity, and (2) the top-view perspective of UAV, which ensures little overlap between any two vehicles. If a prediction matches any detection, the new position of the vehicle, *pos*, in the current frame, is obtained by averaging the positions of its detected boxes and predicted boxes. Then the vehicle dictionary *veh_dic* will be updated by appending *pos*.

In the current frame, sometimes, there could be no detected vehicle box associated with a predicted vehicle box. This situation can be divided into two cases. In the first case, the vehicle appearing in the previous frame is outside of the UAV camera view in the current frame, thereby it will not be added to the latest *veh_dic*. In the second case, the vehicle is still in the camera view, but it is not detected. In other words, it is an FN. Hence, the prediction itself will be taken as the updated vehicle position and appended to *veh_dic*. An FN example is shown in Figure 9-2(a) with a dashed green ellipse. So far in the association algorithm, all the predictions from the previous frame's identified vehicles have been processed. However, in the detection list *det_list*, those that are not associated with any predictions still need more examination. They could be either FPs from the ensemble vehicle classifier or new vehicle appearing in the camera view. We design a rule to determine whether it is an FP or new vehicle with the interaction of the lane information extraction stream: If the center of a detected box is inside the bound of the roadway (which is determined at the same time by the lane extraction stream) and it is within a certain distance to any of the video frame bounds, it is regarded as a new vehicle. Otherwise, the detection is discarded as an FP. A new vehicle will be assigned a new vehicle ID and added to *veh_dic* along with its detected box. In Figure 9-2(a), an example of a new vehicle is masked with a green translucent rectangle, and an FP is with a solid green ellipse. Another case of identifying an FP is that: if a bounding box is not associated with any vehicle detected by the detector in the past five frames, it will be deactivated. This operation reduces the possibility of tracking a non-vehicle that is falsely detected within the

roadway region. In Figure 9-2(c), it shows the final vehicle tracking results with vehicles correctly identified.

Table 9-1 Pseudocode of the association step of the proposed multiple-vehicle tracking algorithm

Input: detected vehicle list *det_list*, predicted vehicle dictionary *pre_dic*,
Output: identified vehicle dictionary *veh_dic*

```
for pre in pre_dic:
    for det in det_list:
        if the IoU for pre and det is larger than threshold_IoU:
            The position pos for vehicle pre_key is calculated as the
            averaged position of pre and det
            veh_dic appends [pre_id , pos] as an identified vehicle

    if no detection is found to be associated with pre:
        if pre is out of video frame bounds:
            Skip adding pre to the active vehicles veh_dic
        elif pre was not associated with any det in the past five frames:
            An FP is found, stop tracking pre
        else:
            An FN is found, veh_dic appends pre as an identified vehicle

for det in det_list that has not been associated with any prediction:
    if det is within the bounds of the roadway & det is within a certain
    distance to video frame bounds:
        new_id is assigned to det
        veh_dic appends [new_id , det] as a new vehicle into the UAV
        view
    else:
        An FP det is found, so det will be discarded.
```

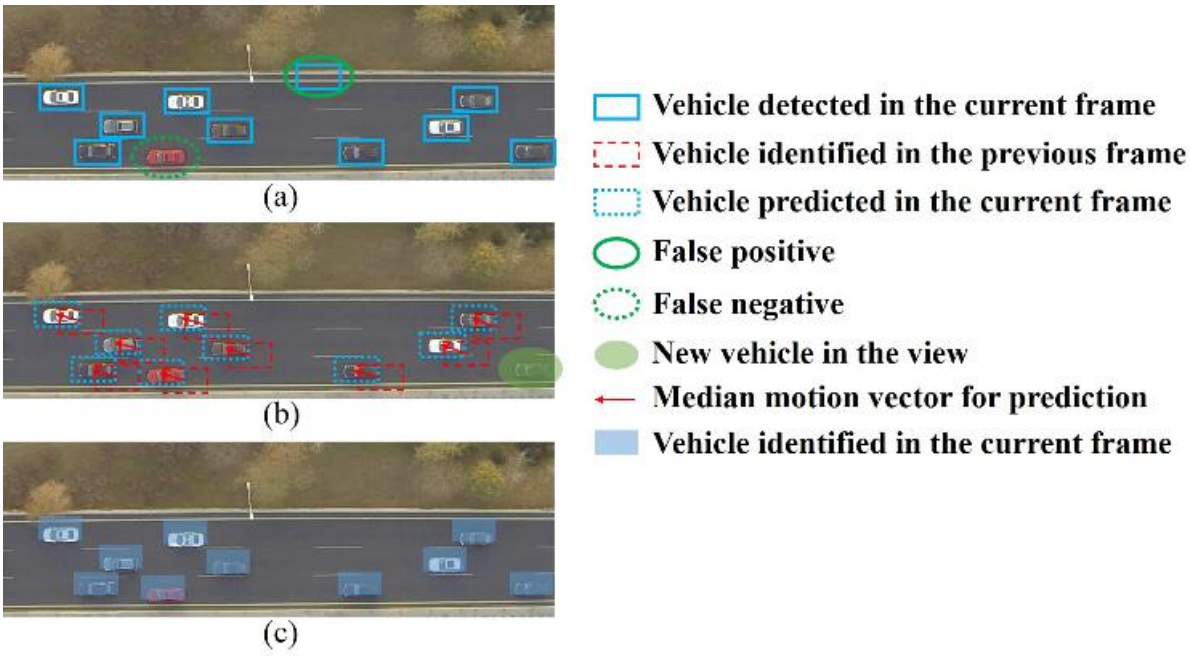


Figure 9-2 Diagram of the proposed long-term multiple-vehicle tracking algorithm.

9.5 Core Functional Module: Lane Information Extraction

Based on the characteristics of UAV videos, we develop a traffic lane information extraction method as one stream in the core functional module. It is composed of four main steps: 1) modified Canny edge detection and Hough transform, 2) adaptive DBSCAN for redundant line filtering, 3) lane boundary tracking in Hough space, and 4) lane length calculation. The first step is based on a traditional line detection process that applies Canny edge detection and Hough transform (Canny 1986; Duda and Hart 1971). Given that most traffic lane boundaries are within a specific color range (in general, white or yellow), we add color information to the edge detection process besides the gradient information. This can filter out pixels that have large gradients but are not from the lane boundaries. The proposed lane detection method is essentially a combination of a traditional yet effective line detection algorithm and a proposed adaptive unsupervised learning algorithm,

with several other rules. We do not use deep learning features here in the lane information extraction with the consideration of processing speed for UAV video analytics.

In the second step, we develop a clustering algorithm for redundant line filtering. This design is based on the problem that a single traffic lane boundary is sometimes associated with multiple detected Hough lines. These boundaries are either double lines or so thick that they are detected as multiple lines. Thus, a clustering method would be a natural solution to the multi-line problem. DBSCAN, which is based on the density of elements, can detect the number of clusters (i.e., the number of traffic lane boundaries) (Ester et al. 1996). Therefore, it is selected over other clustering methods for this problem. Two parameters are critical in DBSCAN: (1) *minPts*, the minimum number of samples in a cluster, and (2) *eps*, the maximum distance between two samples for them to be considered as in the same cluster. The *minPts* is fixed to 1 in the proposed framework since some lane boundary may be just associated with one detected Hough line. Another parameter, *eps*, should be adjusted adaptively in each frame. Note that from a top-view perspective, lanes' widths are supposed to be identical, which means the distances between each pair of neighboring clusters should be identical. In practice, *eps* should be set a little smaller than the lane width to avoid grouping adjacent lane boundary lines into one cluster. Also, lane width may constantly change throughout the video clip due to UAV height changes. In every frame, *eps* is set equal to the average vehicle width (in the unit of a pixel), which is determined by our vehicle detector. This specific design of settings takes advantage of the UAV perspective as well as the use of the vehicle detector, and it turns out to work very well. An example is shown in Fig. 3.

Following the redundant line filtering, a lane boundary tracking process is developed in the Hough space by combining detection and prediction results. As aforementioned, motion vectors are grouped into traffic motion and background motion by the vehicle bounding boxes. The

predicted motions of the lane boundaries are obtained from the background motion vectors. The detected lines are stored in a data list, and the prediction is made by adding the background motion in the current frame to the lane boundary list of the last frame. In case any lane boundary is not detected in the current frame, the predicted position will be added to the lane list for this frame. Otherwise, the updated position for a lane boundary will be calculated as the average of its predicted and detected positions. The tracking process further improves the reliability of lane boundary detection and ensures the robustness of the whole system.

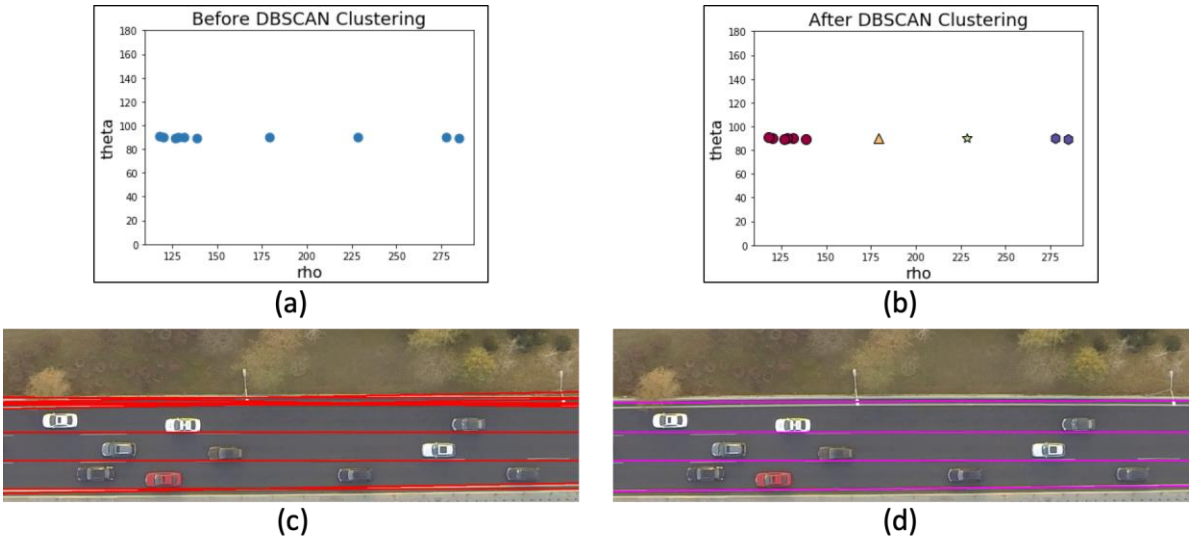


Figure 9-3 The adaptive DBSCAN algorithm to cluster redundant Hough lines in the Hough space for traffic lane boundary detection. Subfigure (a) displays the corresponding Hough lines detected in (c), subfigure (b) presents the clusters in the Hough space, and (d) shows the post-clustering results in the video

Then, we calculate the lanes' lengths. This process is necessary for traffic density calculation. With the traffic lane boundaries extracted, each lane will be assigned a label according to their distance to the origin of the frame. Suppose a traffic lane is bounded by two Hough lines in Eq. (9-1),

$$\begin{cases} rho_1 = x * cos(theta_1) + y * sin(theta_1) \\ rho_2 = x * cos(theta_2) + y * sin(theta_2) \end{cases} \quad (9-1)$$

where rho_1, rho_2, $theta_1$, $theta_2$ are the Hough line parameters. Then, with the video frame's dimension known, we can determine the lane length within the frame based on straightforward geometric calculations. The detailed process is shown in Table 9-2. The length in the unit of a pixel will be converted into physical length using a conversion rate *r*, which is updated in every several frames using a robust online method (introduced in Chapter 8.7). In this method, the initial value of *r* at the beginning of a video needs to be manually inputted by measuring the length in pixels of an object with a known dimension. Then, this conversion rate is updated in an online manner using vehicle bounding boxes and the t-test process.

Table 9-2 The calculation process of the lane lengths in every frame

Input: Lane boundary representations in the Hough space with parameter *theta* and *rho*, video frame height *ht*, and video frame width *wt* **Output**: The length of the lane *len*
For every lane *L:* 1) Assume it is bounded by two Hough lines, $\boldsymbol{rho_1 = x * cos(theta_1) + y * \sin(theta_1)}$ and $\boldsymbol{rho_2 = x * cos(theta_2) + y * \sin(theta_2)}$ 2) Let $\boldsymbol{x = 0}$ and $\boldsymbol{x = wt}$ in the first Hough line equation, get two *y* values $\boldsymbol{y_{11}}$ and $\boldsymbol{y_{12}}$ 3) If $\boldsymbol{y_{11} \in [0, ht]}$, one end point of lane boundary #1 is $\boldsymbol{P_{11} = (0, y_{11})}$; otherwise, there must be $\boldsymbol{y_{12} \in [0, ht]}$, thus $\boldsymbol{P_{11} = (wt, y_{12})}$ 4) Let $\boldsymbol{y = 0}$ and $\boldsymbol{y = ht}$ in the first Hough line equation, get two *x* values $\boldsymbol{x_{11}}$ and $\boldsymbol{x_{12}}$ 5) If $\boldsymbol{x_{11} \in [0, wt]}$, the other endpoint of lane boundary #1 is $\boldsymbol{P_{12} = (x_{11}, 0)}$; otherwise, there must be $\boldsymbol{x_{12} \in [0, wt]}$, thus $\boldsymbol{P_{12} = (x_{12}, ht)}$ 6) Repeat 3) – 5) to get the endpoints $\boldsymbol{P_{21}}$, $\boldsymbol{P_{22}}$ for lane boundary #2. **7)** Calculate the length of the lane as $\boldsymbol{len = \frac{1}{2}(||P_{11} - P_{12}||_2 + ||P_{21} - P_{22}||_2}$, where $\boldsymbol{||\cdot||_2}$ measures the Euclidean norm of a vector

9.6 DATA STORING MODULE

In this module, data structures and functions are built to take the outputs of the core functional module and store information for a whole UAV video clip. There are three low-order data structures in this module, which are 1) a data list of video background motion at every timestamp, 2) a vehicle class that stores each individual vehicle's information, and 3) lane dictionaries in which the key is a lane ID and the value is a list of lane lengths at every timestamp. At the high order, the vehicle class contains five different types of vehicle data: 1) a list of IDs showing which lane the vehicle is in at every timestamp, 2) a list of angles showing the vehicle's moving directions, 3) the life period of the vehicle in the video view, 4) a list of speed values of the vehicle, and 5) a list of coordinates showing the positions of the vehicle.

These data will be further used for the estimation of both microscopic and macroscopic traffic parameters. In the lane length calculation process, the instantaneous lane information in a frame is recorded in a dictionary, where the dictionary key is the lane ID and the dictionary value stores a list of lane length values. The background motion list is also updated for every frame by appending the median background motion vector of that frame. When a new vehicle appears in the UAV camera view, a vehicle object will be constructed with a new vehicle ID. Meanwhile, several empty lists representing the position, speed, moving direction, life period, and lane information variables will be created. The vehicle association algorithm shown in Table 9-1 outputs every vehicle's position in every frame as a 1-D list with a fixed length of four (four corners of the bounding box), and it will be appended to the 2-D position list in the corresponding vehicle class. To calculate the instantaneous speed of a vehicle, the data storing module records the bounding box's center C_{cur} of the vehicle for the current frame, the bounding box's center C_{pre} of the vehicle for the previous frame, the instantaneous background motion vector M_b, the video frame rate fr, and the

conversion rate r of pixel length to physical length. Hence, the instantaneous ground speed v_{veh} and moving direction dir_{veh} for a vehicle is calculated in Eq. (9-2):

$$\begin{cases} v_{veh} = ||C_{cur} - C_{pre} + M_b||_2 \times fr \times r \\ dir_{veh} = tan^{-1}(C_{cur} - C_{pre} + M_b) \end{cases} \quad (9-2)$$

where $||\cdot||_2$ is the Euclidean norm of a vector, and $tan^{-1}(\cdot)$ is the arctangent function. The speed will be appended to the 1-D speed list of the corresponding vehicle object. To determine the instantaneous lane information for a vehicle, we compare the vehicle's center C_{cur} with the lane boundaries to see which lane the vehicle is in. Then we append the lane ID to the 1-D lane list of this vehicle. Note that the "life period" variable in the vehicle class is a two-element list that records when the vehicle enters and exits the UAV view.

9.7 Traffic Parameters Estimation Module

Speed, density and volume are the most critical three macroscopic traffic parameters. Eq. (9-3) describes the fundamental relationship for them.

$$q = v \times k \quad (9-3)$$

where *q*, *v*, and *k* denote volume, speed, and density, respectively. The macroscopic parameters can be aggregated for either each lane or all lanes automatically in our framework since we extracted the lane information. Traffic density can be calculated from vehicle counts and length of the roadway. In this paper, we mainly discuss the estimation of macroscopic parameters for each lane, since this process has not been automated so far in existing UAV-based studies. We have

recorded lane lengths as well as the vehicle information for each frame. Therefore, it is feasible to read the data from the data storing module and calculate lane-level density. In any frame, assuming N_t vehicles are identified on lane t, and the length for lane t is len_t, then according to the definition of traffic density, Eq. (9-4) computes the density k_t for lane t.

$$k_t = \frac{N_t}{len_t} \tag{9-4}$$

The instantaneous macroscopic speed of lane t is obtained by calculating the space mean speed of all vehicles on lane t, which is represented by Eq. (9-5). The volume of each lane is estimated using Eq. (9-3).

$$v_t = \frac{N_t}{\sum 1/\, v_{veh}} \tag{9-5}$$

Microscopic traffic parameters depict the behaviors of individual vehicles at a particular timestamp. Typical microscopic traffic parameters include vehicle position, vehicle speed, space headway, and time headway. The vehicle position and vehicle speed are already stored in the vehicle class. We denote the vehicle position list and vehicle speed list *pos* and *spd*, respectively, and the starting frame of this vehicle is fr_{start}. Thus, Eq.(9-6) yields the instantaneous position pos_i and speed spd_i of vehicle V_i from the data storing module in any frame fr_{cur}:

$$\begin{cases} pos_i = pos[fr_{cur} - fr_{start}] \\ spd_i = spd[fr_{cur} - fr_{start}] \end{cases} \tag{9-6}$$

where $list[n]$ is the *(n+1)-th* element of the list. To extract a vehicle's space headway and time headway in frame fr_{cur}, the first step is to find out the preceding vehicle of the target vehicle. Mathematically, vehicle V_1 is followed by vehicle V_2 if constraints in Eq.(9-7) are all satisfied:

$$\begin{cases} lane_1 = lane_2 \\ |tan^{-1}(pos_1 - pos_2) - dir_2| < \pi \\ |pos_1 - pos_2| = min(pos_{i,down} - pos_2) \end{cases} \quad (9-7)$$

where the first constraint ensures the lane labels of V_1 and V_2 are the same (i.e., on the same lane), and the second constraint ensures that V_1 is downstream the target vehicle V_2, and the third constraint ensures V_1 is closer to V_2 than any other downstream vehicle $V_{i,down}$.

With this searching process, it is then possible to compute the headways of the target vehicle V_2. Assuming the space headway of V_2 is H_{space}, and time headway H_{time}, Eq. (9-8) yields the results of H_{space} and H_{time} in a frame.

$$\begin{cases} H_{space} = ||pos_1 - pos_2||_2 \times r \\ H_{time} = \dfrac{H_{space}}{spd_2} \end{cases} \quad (9-8)$$

where r and $||\cdot||_2$ are the conversion rate and the Euclidean norm mentioned above. It is worth noting that instantaneous headways are not available for the leading vehicle in any lane due to that the vehicle it follows is out of the UAV view. This could be solved by operating multiple UAVs, however, it is out of the scope of this study. With the microscopic traffic parameters extracted from UAV video, microscopic behaviors such as car following and lane changing can be observed and analyzed.

9.8 EXPERIMENTAL RESULTS

9.8.1 *Dataset and Preliminary Results*

In total, over one hour of video clips in various traffic scenarios were collected to test the proposed methodology, which was implemented in Python language. The key extra libraries that were configured for the implementation were OpenCV for image processing and Keras for detector training with Tensorflow backend. Generally, the overall performance was promising that few false detections or missed vehicles showed up. First of all, we carefully examined 600 representative video frames to validate the effectiveness of the methodology. In this video clip, the UAV had irregular background movement patterns including rotation, cruising, and vibration. It also contained other challenging scenarios such as where the UAV was flying over a highway sign gantry that blocked vehicles from the top.

Figure 9-4 shows some example output frames, in which traffic lane boundaries and vehicles are marked with lines and bounding boxes. Vehicles from different lanes are differentiated by bounding box colors, with their vehicle IDs on the left-top of the box. The highway sign gantry can be observed in the first three example frames with vehicles partially blocked. It can be seen that most of these blocked vehicles were well tracked and identified by our method. Motion-vectors were marked as short green/brown arrows in the example frames. The green arrows were outside detected bounding boxes for background motion-vectors extraction, and brown arrows were inside the bounding boxes for vehicle motion extraction. Then, another four videos will be added for further analysis.

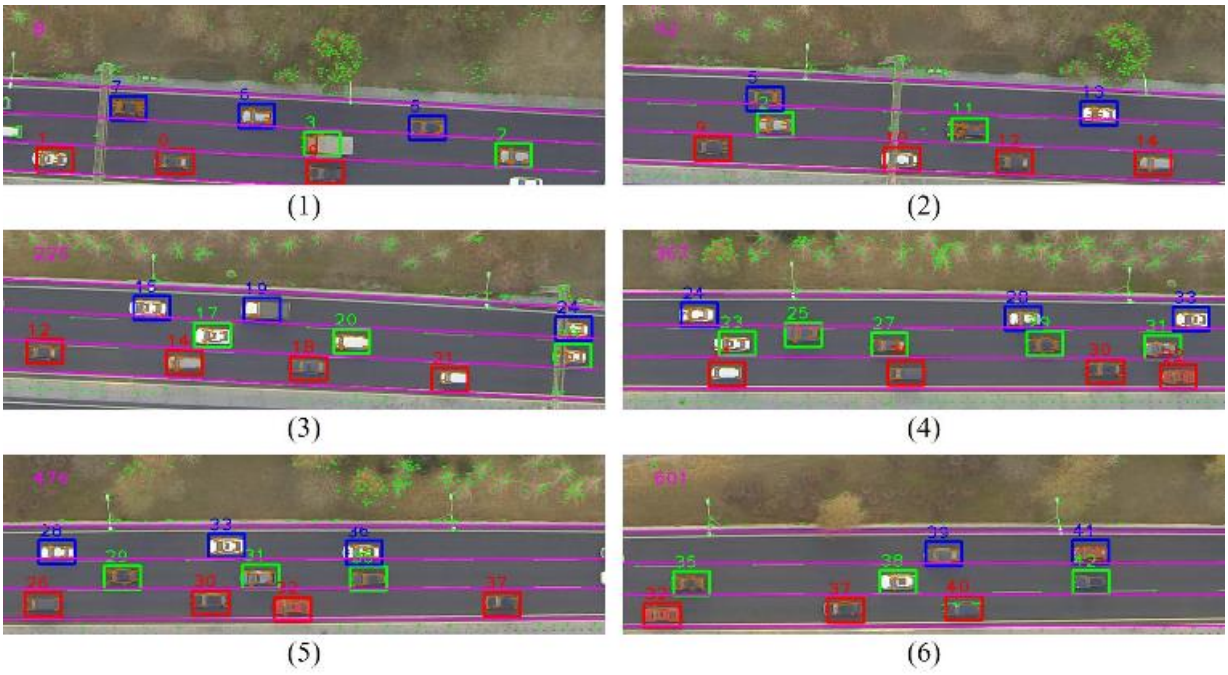


Figure 9-4 Example output frames showing the preliminary results in a UAV video with a moving background. Motion vectors, lane boundaries, and vehicles are marked in the video frame.

9.8.2 *Microscopic Traffic Parameters Estimation and Analysis*

Figure 9-5 to Figure 9-7 present the estimation results of microscopic traffic parameters. Figure 9-5 contains three charts, each of which displays one type of extracted parameter for all vehicles in a single chart. We name the charts as "vehicle-frame charts," which are formed by two dimensions: one is the frame number, or in other words, time dimension, and another is the vehicle ID. The vehicle-frame charts are literally matrices that are produced by stacking each vehicle's one-dimensional microscopic parameters throughout the whole video together. These charts record the number of vehicles, vehicles' life periods, and some microscopic parameters for any given vehicle at any timestamp in a clear and straightforward way.

The top vehicle-frame chart in Figure 9-5 records all vehicles' life periods using a binary image, where the white part illustrates a case that a vehicle with a certain ID is within the UAV

view at a certain frame. In the middle of the figure, the second vehicle-frame chart gives an impression of the speed information, where the lighter color corresponds to a higher speed. The vehicle-frame lane chart at the bottom shows each vehicle's lane information. There are three colors (red, blue, white) in this chart and each of them represents one of the three lanes.

A better understanding of traffic patterns and vehicle behaviors is likely to be achieved by properly utilizing these vehicle-frame charts produced by our system. For example, in the vehicle-frame speed chart, traffic speed patterns can be visualized and give us an intuition on when and how the speed changes. In Figure 9-5, we can observe speed oscillations at around the end of the period for vehicles on all lanes (the vehicles' corresponding lane information is shown in the third chart of Figure 9-5). With this observation, we know that the traffic has a trend to slow down, given the knowledge in traffic flow characteristics and congestion formation. From these charts, we see the distribution of vehicles on each lane, and the relationship between traffic speed and lane-changing behaviors (lane #, timestamps, durations, etc.) can also be roughly visualized.

Detailed microscopic parameters for individual vehicles can be visualized in Figure 9-6 and Figure 9-7. Vehicle positions, speeds, and lane information of three representative vehicles throughout the experimental video are displayed in Figure 9-6. The position sub-figures are the same size as the video frame, thus to display vehicle trajectories in the video. Video frame numbers are the x-axes for speed sub-figures and lane sub-figures, and y-axes are labeled as mph for speed sub-figures and lane # for lane sub-figures, respectively. Note that the traffic moves from the right of the frame to the left.

The vehicle with ID #9 is one of the vehicles appearing in the very beginning of the video recording. Therefore, its starting position is in the middle with respect to the x-axis instead of the rightmost. The speeds for vehicle #9 throughout the video are almost 30 mph and it stays in lane

#3 for its life period. Vehicle #13 enters the UAV view at about frame #60 and leaves at about #218. From the speed curve, a speedup can be observed in its life period from about 30 mph to 35 mph, and it can be seen this vehicle stays at lane #1 with no lane changing behavior. Vehicle #25 enters the view at frame #246, and its speed remains relatively constant with a very small decrease after frame #350.

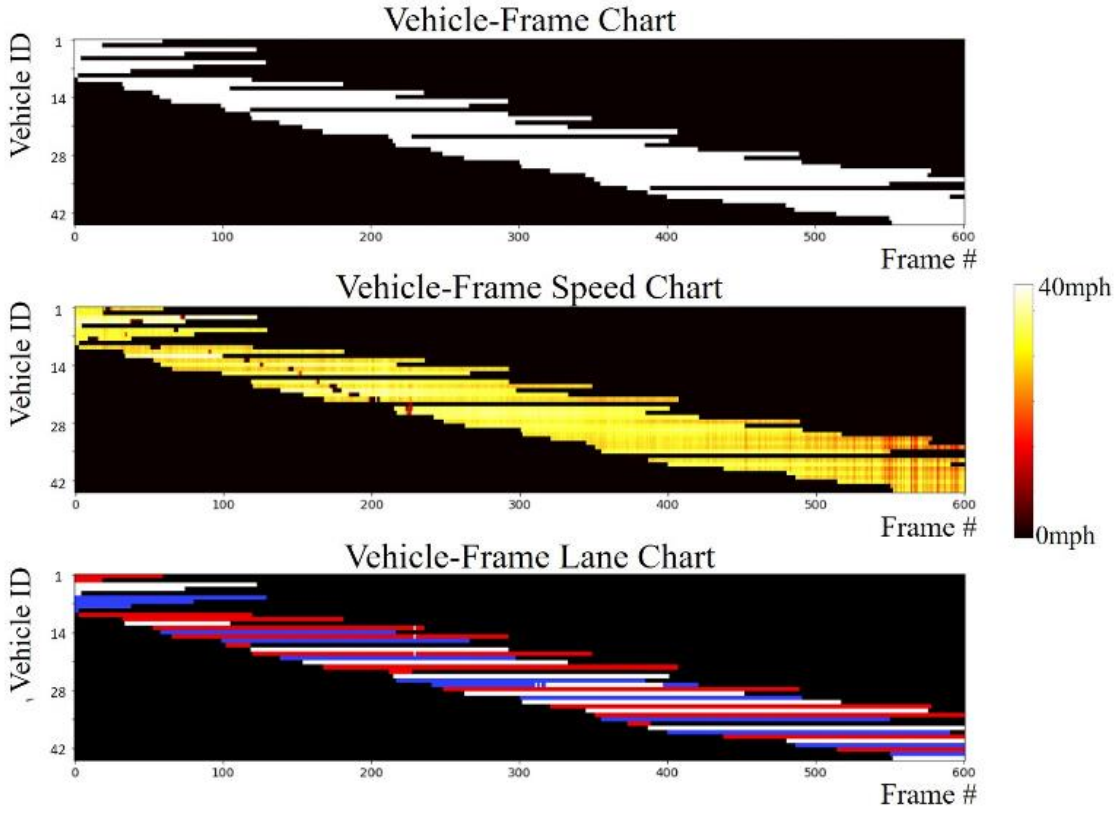


Figure 9-5 These are the vehicle-frame charts showing some of the microscopic information of all vehicles' in 2D matrices, and in each chart, the x-axis denotes the video frame number and the y-axis the vehicle ID. White or colorful pixels mark vehicle appearances or certain parameter values.

In general vehicle #13 and #25 have higher speeds than vehicle #9 during their life periods. Lane changing actions can be observed from the lane information curve of vehicle #25, it moves from lane #1 to lane #2, and then returns to lane #1 after about 80 frames. It is noticed that there

are two small peaks in the lane sub-figure of this vehicle (#25) for its first-time lane changing from lane #1 to lane #2. This is due to detector estimation errors of the vehicle's location. There are always errors in the vehicle location (or bounding box) estimation, though most time it is small with a good detector and cannot be visualized. However, in a lane changing situation, there is time the vehicle's geometric center is right on the boundary of two adjacent lanes. And that is when minor localization error causes the "peaks", or in other words, the vibration pattern in the figure.

As aforementioned, space headway and time headway are key traffic microscopic parameters and can be automatically given out by the proposed framework for traffic data collection and behavior analysis. An example is shown in Figure 9-7, where four vehicles (with vehicle ID 30, 32, 37, and 40) are following one another in the same lane. We analyzed sixty frames from #518 to #577 from vehicle #40 entered the view to vehicle #30 left the view. Seven sub-figures are included in Figure 9-7, where the first one is an original frame with the four target vehicles marked by bounding boxes and vehicle IDs, and the other six sub-figures show the headway curves for the 60-frame period. The first row includes three space headway curves and the second row includes three time-headway curves. It can be observed from the video frame in Figure 9-7 that vehicle #32 follows closely to vehicle #30, and the other two vehicles have relatively larger following distances.

From the space headway curves, it can be seen vehicle #32 and #30 are within 10 meters, which is a short distance given the traffic speed is around 30 mph. The space headway of vehicle #37 and #32 is larger than 20 meters, which is the largest of the three. The space headway of vehicle #40 and #37 starts at about 20 meters, but an apparent decreasing trend can be observed. In fact, all three space headway curves have decreasing trends. It is inferred that the leading vehicle has a slower cruise speed than average traffic flow speed during that period. This inference has

been proved correct based on the extracted speeds of these vehicles. The time headway curves provide additional information to the car following behaviors. Among all three time-headway curves, the first one generally has the smallest values, which makes sense because the associated space headway is much smaller than the others. An interesting observation is that the time headway for vehicle #37/#32 is basically much larger than that for vehicle #40/#37, even though the associated space headways are not that different. This means that, besides having a larger acceleration rate, vehicle #40 also has a larger speed than vehicle #37. In terms of the drivers' car following behaviors, these observations and analyses lead to the finding that vehicle #32 and #40 are more aggressive than vehicle #30 and #37.

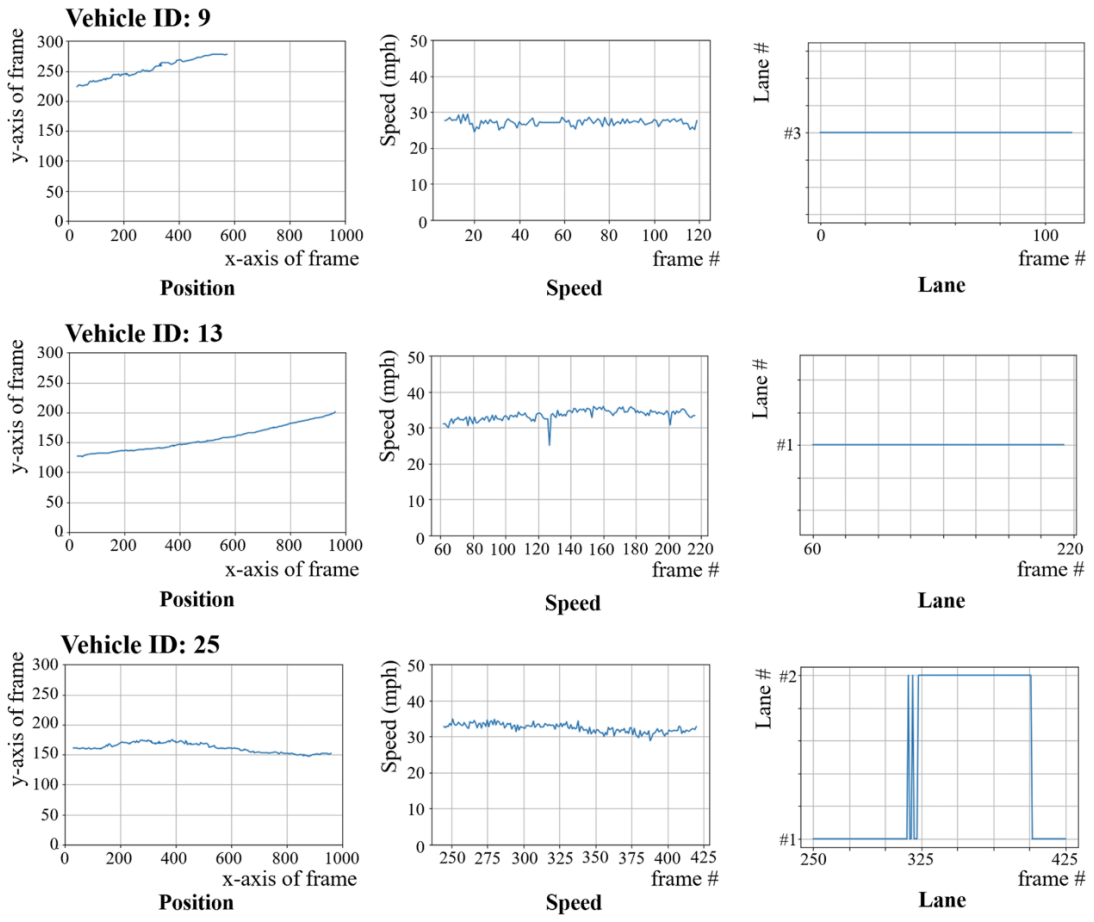


Figure 9-6 Examples of individual vehicle tracks, speeds, and lane information extracted using the proposed framework.

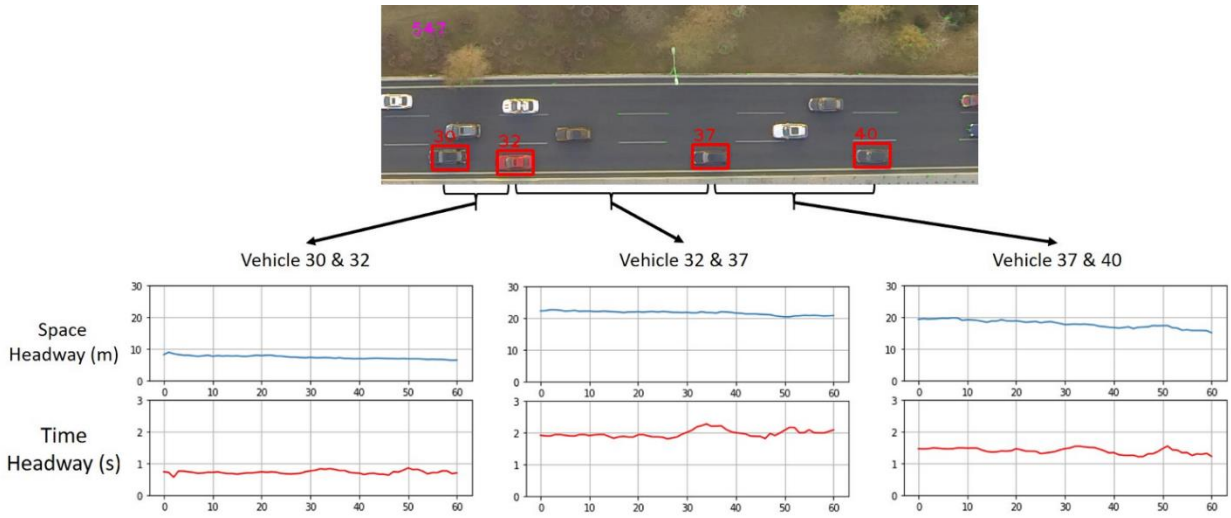


Figure 9-7 Fig. 7 Space headway and time headway extraction and car following behavior analysis.

9.8.3 *Lane-Level Macroscopic Traffic Parameters Estimation and Analysis*

Since reporting instantaneous macroscopic traffic parameters frame by frame is not meaningful in practice, the averaged traffic speeds, densities, and volumes of all frames are computed and displayed in Table 9-3 for every and each lane. In this table, we also show the speed estimation accuracies and vehicle count estimation accuracies. Despite a variety of traffic parameters given out by our method, the most fundamental metrics for evaluation of the system performance are lane estimation, vehicle speed, and vehicle counting. The summarized overall evaluation results are shown in Table 9-3 and Figure 9-8. Throughout the entire video, no FP was generated, no vehicle was falsely identified with multiple labels, and very few FNs were generated. The only vehicle identified as an FN was the missed tracking for one vehicle on the middle lane from frame #106 to #161. This vehicle was missed after passing through the gantry. It was found that this missed vehicle had a relatively small size and similar color to the road surface. Hence, when it was blocked by the sign gantry, only a small portion of the motion-vectors extracted was

truly eligible for vehicle tracking; at the same time, the vehicle detector missed this vehicle as well due to the vehicle pattern misclassification caused by both the obstruction and the color issue. This kind of scenario is not common since it occurs only when there are significant errors in both the vehicle detection and vehicle tracking processes. A possible solution is to add a shape estimation on top of the motion estimation.

In Figure 9-8, the red dotted curves are the ground truths, and others are estimations. We can see the vehicle counting estimation for lane #1 fits perfectly to the ground truth. For lane #2, the only missed vehicle (frame #106-#161) contributes to the major part of estimation error. Only one out of six hundred frames has an incorrect lane boundaries estimation, and this frame number is 230. In Figure 9-8, a sudden count increase for lane #2 and a sudden count decrease for lane #3 can be seen at frame #230. While the vehicle detection/tracking works well, three vehicles on lane #3 are grouped into lane #2. This kind of misclassification could happen due to different reasons. But since this problem is rare, it does not have a big influence on any of the parameter estimations. The vehicle counting accuracies for lane #2 and lane #3 reaches 95.4% and 98.8%. Like in previous studies, the ground truth speed data was manually collected with an on-screen pixel measurement tool (Ruimin Ke et al. 2018; Ruimin Ke, Li, et al. 2017). Every individual vehicle's speed has been measured throughout the video, and the mean accuracies are calculated by comparing them with the estimated values. According to Table 9-3, the speed estimation accuracy tends to have a positive correlation with vehicle counting accuracy, since the speed accuracies are 97.3%, 96.6% and 96.6% for the three lanes.

From Figure 9-8, it can be seen that the original speed estimation curves (solid blue curves) contain some significant underestimations of the speeds, which appear like spikes. The spikes show up when there are vehicles right under the highway sign gantry. As aforementioned, only

one vehicle was missed after passing through the gantry, but the blockage of UAV view did produce errors for the motion estimation since some motion-vectors on the gantry were inside of the vehicles' bounding boxes. To address this issue, we adopt a straightforward method to filter out those spikes, in which a spike is detected if it is significantly smaller or larger than its nearest neighbors. Specifically, every point on the original speed curve is compared with the median value of a 15-frame long time series from 7 frames before it to 7 frames after. If it is significantly smaller or larger than the median, its value would be replaced by the median value. We use 15 as the length here because we consider 15 frames are short enough (around 0.6s) to consider the traffic speed constant and long enough to filter out the sudden changes.

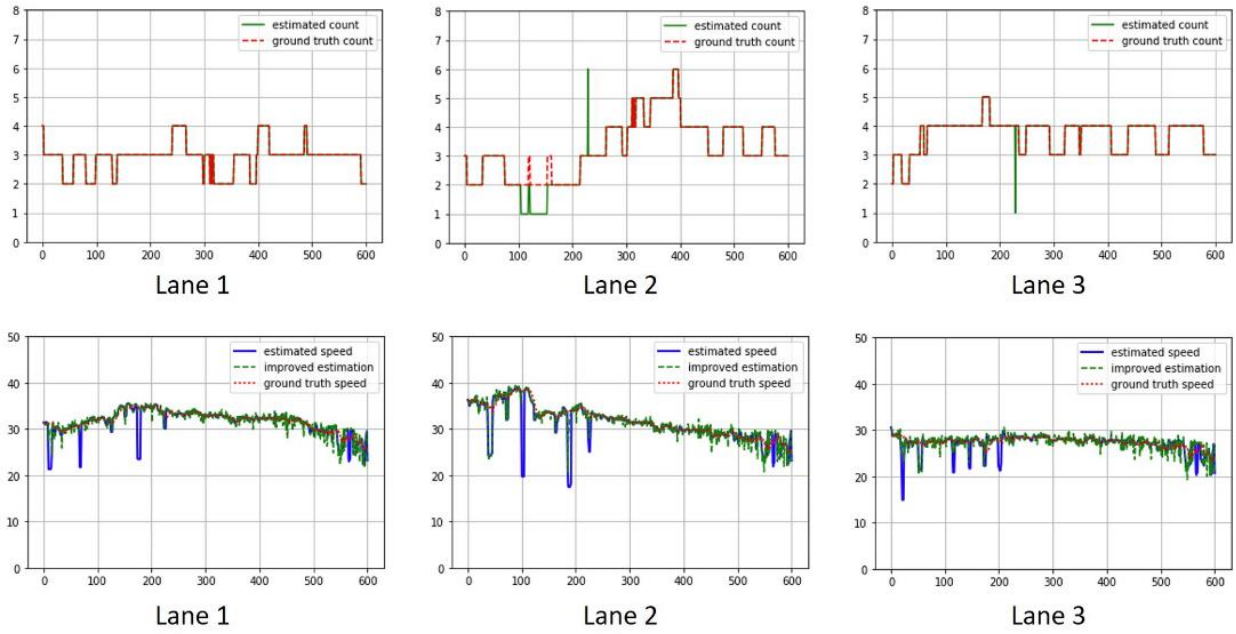


Figure 9-8 Accuracy curves for lane-level vehicle counting and traffic flow speed estimation.

Table 9-3 Macroscopic traffic parameter estimation results and overall system accuracy evaluation

	Lane #1	Lane #2	Lane #3
Estimated speed (mph)	32	33	28
Estimated density (pc/mi/lane)	26	33	38
Estimated volume (pc/h)	905	1200	1155
Speed estimation accuracy (%)	97.3	96.6	96.6
Count estimation accuracy (%)	100	95.4	98.8

9.8.4 *More Validation*

To further validate our proposed framework, we added another four UAV videos which were taken in different scenarios (see Figure 9-9). Video #2 was collected over a seven-lane urban freeway segment, where the right-most two lanes were ramp lanes connected to the mainstream freeway. Video #3 was a two-lane roadway in a rural area, where the traffic was not as dense as in other videos. In this video, the UAV was hovering over the road segment so that the video background was stationary. The camera view of Video #4 covered six lanes on a two-way road segment. For two-way traffic, note that one minor modification was needed in the macroscopic traffic parameter estimation process: instead of aggregating all vehicles' data, it was necessary to first determine which direction the vehicles moved towards using their velocities. Video #5 was a video clip we got from the Washington State Patrol Aviation. The key feature of this video was that it was collected by a manned aircraft rather than a UAV. Therefore, the camera movement was faster than in the other four cases.

Table 9-4 summarizes the properties of the five test videos and the performances of our method on them. The overall estimation accuracy is high and stable. It can be seen that the five videos are different and representative. The number of lanes in the five videos ranges from two to seven, which covers most cases in the real world. We also have one video (Video #4) that monitors a two-way road. The purpose of having this video was to test the applicability of our method in

extracting traffic information from multi-directional roads. The UAV motion patterns in the test videos also vary. We notice that the moving speed of the aircraft has some impacts on the performance. We can see that the overall speed estimation accuracy for Video #5 is significantly lower than the others. By looking at this video, it is found that the fast speed of the aircraft creates extra challenges in traffic speed estimation because large motions often lead to reduced motion estimation accuracy for optical flow and object tracking. We also evaluated the camera zooming cases. In Video #2 and #5, there are some camera zooming cases while others not. We do not observe significant performance differences between the time around camera zoom and the remaining time period. This result is encouraging and reasonable because of our vehicle detector and adaptive DBSCAN algorithm. The vehicle detector can detect vehicles in multiple scales, which ensures the detection accuracy in zooming cases; and the adaptive DBSCAN algorithm in the lane detection is proposed to handle camera zooms and UAV height changes with its adaptive *eps* value. Please note that a sample video for demonstration purposes has been uploaded to this link: https://www.youtube.com/watch?v=1J_3R303r5w.

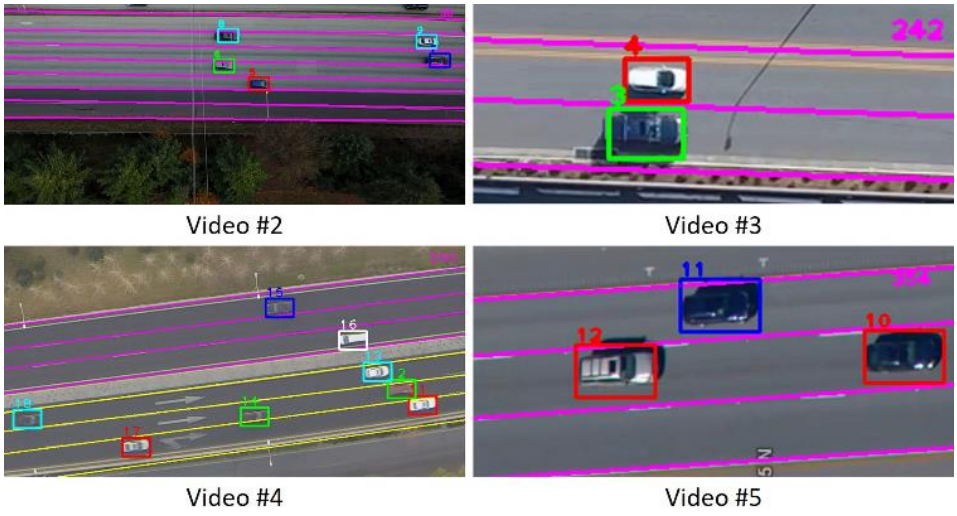


Figure 9-9 Four more UAV videos are collected for validating the proposed framework.

Table 9-4 Validation and analysis of five different videos

	Video #1	Video #2	Video #3	Video #4	Video #5
Num of lanes	3	7	2	6	3
Direction	One	One	One	Two	One
UAV motion	Medium	Slow	Zero	Medium	Fast
Camera zoom	No	Yes	No	No	Yes
Speed acc. %	96.8	94.3	96.5	95.7	88.1
Count acc. %	97.9	93.1	98.3	96.0	94.4

9.8.5 *Comparison with State of the Arts*

In this subsection, we compare the vehicle tracking algorithm and the entire framework with the state of the arts. Table 9-5 shows the comparison of our vehicle tracking algorithm with two state-of-the-art tracking methods that have shown their exceptional ability for UAV-based vehicle tracking (Bewley et al. 2016; X. Cao et al. 2014; D. Du et al. 2018). Four widely used metrics for object tracking were picked: False positive (FP), false negative (FN), identity switches (IDS), and frames per second (FPS). FP, FN, and IDS are for accuracy evaluation and FPS are for efficiency evaluation. As aforementioned, FP is the false detection of vehicle, FN is the missed vehicle. In our context, IDS measures the times that a vehicle's ID switches. It usually happens when a vehicle is missed by the tracker and then re-detected with a new ID. Note that ID switches are less common in UAV videos than traditional traffic surveillance videos due to fewer occlusions in UAV videos given their top-view perspective. The smaller the values of FP, FN, and IDS are, the tracking performance is better.

As presented in Table 9-5, the proposed vehicle tracking algorithm outperforms the state of the arts in FP, FN, and IDS. The numbers in the table are in the unit of vehicle-frame (except FPS), which is the sum of the number of FP/FN/IDS in every frame. In video #1, 0 FPs, 56 FNs, and 2 IDSs occurred for our algorithm. The 56 FNs were caused by the missed tracking of one vehicle.

The 2 IDSs occurred when the tracked vehicle was out of the camera view for a while due to the UAV movements. Video #2 was more challenging to the vehicle detector, thus the tracking performances were also influenced by the detector and errors were larger for all three accuracy metrics. The FPs in video #2 was caused by two false detections tracked by our algorithm. But then they were discarded after not being associated with any detection in the next five frames. For FPS, SORT was the fastest among the three, but our algorithm achieved slightly better speed than Particle filter and comparable to SORT.

Table 9-5 Multiple vehicle tracking comparison with two state of the arts

		Particle(X. Cao et al. 2014)	SORT(Bewley et al. 2016)	Ours
Video #1	FP	5	4	**0**
	FN	178	130	**56**
	IDS	10	9	**2**
	FPS	12	**16**	13
Video #2	FP	24	18	**10**
	FN	246	182	**95**
	IDS	12	9	**3**
	FPS	10	**15**	11

The superiority of our algorithm comes from the special design considering UAV features and traffic information. Besides the FP and FN filtering scheme described in Section 9.4, the IDS is largely reduced because our algorithm tends to keep tracking a vehicle even if the detector loses it for a few frames. However, Particle or SORT will assign a vehicle with a new ID if it is not detected for more than one frame. This design does not directly improve the macroscopic traffic parameter estimation much, but it is super beneficial to microscopic data collection since it extracts individual vehicle information. Moreover, the incorporation of KLT tracker in the algorithm ensures a good tracking accuracy: When just using historical movements to predict the future locations like Kalman filter, Particle filter, or SORT, the prediction error can be large due to

UAV's random movements, especially when predicting multiple frames ahead. However, our algorithm predicts individual vehicles' immediate motions in every frame using motion vectors extracted in the vehicle bounding boxes. For the processing speed, it is crucial for UAV monitoring since we expect real-time UAV monitoring in the future. While our tracking pipeline has multiple FP/FN filtering steps, its computational complexity is still relatively low, because we only use IoU as the association metric by taking advantage of UAV's top-view perspective, which avoids inter-vehicle occlusions.

In addition to tracking performance analysis, we then briefly compare the proposed framework to three state-of-the-art frameworks (Ruimin Ke et al. 2015, 2018; Ruimin Ke, Li, et al. 2017). These studies accomplished several key milestones in the area of UAV-based traffic flow parameter estimation. According to the summary in Table 9-6, the proposed framework in this paper is able to extract many more types of traffic parameters. The robustness has also been further improved. By running these frameworks on the same video datasets, the new one generates much lower errors than the state of the arts. We use a metric called "the percentage of total frames with FP or FN," which means the ratio of the number of frames with any FP or FN to the total number of frames. This metric decreases from 47% in (Ruimin Ke, Li, et al. 2017) and 38% in (Ruimin Ke et al. 2018) to 9%. It is an impressive result, especially considering that the new framework can produce a much broader range of traffic parameters. Note that this metric value is not available (NA) in (Ruimin Ke et al. 2015) because this framework has no vehicle detection function. The cost is the computational complexity: in the three state-of-the-art studies, the processing speeds on a desktop computer are all over 24 fps, which enables real-time operation. The proposed framework runs in a nearly real-time manner with a speed of 4 fps. This is still

impressive with the functionality of this framework. Real-time capability could be achieved by skipping frames, increasing computation power, optimizing system architecture, and so on.

Table 9-6 Comparison results with the state-of-the-art frameworks for traffic parameters estimation from moving UAVs

	(Ruimin Ke et al. 2015)	(Ruimin Ke, Li, et al. 2017)	(Ruimin Ke et al. 2018)	This study
Overall traffic speed	✓	✓	✓	✓
Overall traffic density	×	✓	✓	✓
Overall traffic volume	×	✓	✓	✓
Lane-level traffic speed	×	×	×	✓
Lane-level traffic density	×	×	×	✓
Lane-level traffic volume	×	×	×	✓
Individual vehicle speed	×	×	×	✓
Individual vehicle trajectory	×	×	×	✓
Space headway	×	×	×	✓
Time headway	×	×	×	✓
Processing speed	32 fps	24 fps	25 fps	4 fps
The percentage of total frames with FP or FN	NA	47%	38%	9%

9.9 APPLICABILITY OF THE STUDY

Advanced traffic sensing: Traditional traffic sensors are usually stationary sensors. To monitor a roadway or traffic network, a large number of sensors are needed, which has a high cost in installation and maintenance. Moreover, traditional traffic sensors collect limited types of traffic data. For example, loop detectors, which are the most common sensors for freeway systems, collect only macroscopic traffic speed, volume, and occupancy data. Probe vehicles, as another widely used sensing method, collects vehicle trajectories and a sample of traffic speed, but not able to collect lane information or volume. Our study offers an option to collect both microscopic and

macroscopic traffic parameters using a sensor that is more cost-effective and flexible than traditional sensors.

Traffic enforcement: Over a decade ago, traffic agencies started to install cameras on manned aircraft for the purpose of issuing speeding tickets from the sky. Now some of the agencies have already considered using UAVs for traffic enforcement in order to reduce the cost of aerial ticketing operations. Our framework can be incorporated into unmanned traffic enforcement systems to monitor multiple individual vehicles and extract microscopic parameters from the sky to assist in enforcement decision making. Compared to manned aircraft solution, UAV solution to traffic enforcement has not only a lower cost but also better scalability.

Post-disaster traffic management: Disasters have a big impact on modern cities regarding human and economic losses. As post-disaster lifelines, post-disaster traffic networks are critical to support various operations like evacuation and recovery. However, traffic sensors may more or less stop working after disasters due to the damage to themselves or the traffic systems. At that time, UAVs can serve as alternative sensors to support traffic performance evaluation and post-disaster traffic planning. With UAV's high mobility and flexibility, its surveillance can cover the areas with the highest needs. Our framework and other related studies can be applied to emergent traffic state estimation and management.

Applicability to other traffic scenarios: There are a few more traffic scenarios we would like to discuss regarding the applicability of the framework: curves, urban situations, and occlusions. The current framework is designed for straight roads rather than curves based on two considerations: 1) straight roads are more common than curves so that the parameter extraction would work in most cases, and knowing the traffic situation does not always need the data from every single road segment; 2) detecting curves would need a different and often more complicated

method, which increases the computational cost. As far as urban situations, this framework still applies to urban roadway links but not intersections because intersections have different geometry with roadway links and the set of parameters for analysis is also different (e.g., queue length, number of stops, delay). For occlusion, one advantage of UAV is its top-view perspective, which naturally avoids occlusions in traditional traffic surveillance videos. One example is a truck blocking a car so that the detection and tracking of the car will likely fail. However, from a UAV perspective, this kind of occlusion rarely happens. Also, the proposed tracking method in our framework will keep tracking the vehicle for several frames even if the detector misses it. It means the vehicle's identity will not switch unless it is blocked by something in multiple consecutive frames.

Chapter 10. FINAL REMARKS

With the recent progress in AI and IoT, this dissertation is the first step in exploring new solutions for traffic video analytics that target both high intelligence and real-time capability. We introduced the design and field implementation of video analytics algorithms, frameworks, and systems based on machine learning and/or edge computing for three key components of modern intelligent transportation systems: smart infrastructure, intelligent vehicle, and aerial sensing. The road ahead is full of opportunities along this way given the high demand for video analytics, machine learning, and edge computing in future smart cities. However, challenges co-exist with opportunities. This chapter summarizes a few foreseeable opportunities and challenges in video analytics for smart cities: 1) robust video analytics in the real world, 2) lightweight models for video analytics, and 3) large-scale camera coordination and data fusion.

Robust video analytics in the real world: Video analytics tasks that seem simple could become extremely complicated or unreliable in real-world environments. There are unexpected and challenging scenarios that your models never see in the training process. A typical example is in Chapter 4 that the proposed parking surveillance system performs much more reliable indoor than outdoor due to extreme lighting conditions and adverse weather. Those scenarios are underrepresented in the training sample and challenging itself (direct lighting or strong fog dulls vehicle features). These are the cases where traditional methods can be surprisingly helpful if applied properly, e.g., background modeling in Chapter 4. Moreover, a recent study by Google reveals that under-specification in machine learning is a key problem for poor model performances in the real world. There are so-called spurious features that are strongly associated with the training data label, and this situation does not improve as the sample size increases (D'Amour et al. 2020).

As the smart city and smart transportation applications target running in the real world, robust video analytics, especially in challenging scenarios, will be one of the areas with huge impact.

Lightweight models for video analytics: A trend in machine learning is that the models are becoming light-weighted. Neural network optimization techniques like pruning and quantization are emerging. Model optimization is implemented in this dissertation using TensorFlow Lite and TensorRT on the IoT devices to improve efficiency. Besides developing a neural network and then compress it, another natural idea is to have simple neural network structures starting from the design, such as the design of the lightweight CNN and ensemble classifier in Chapter 8. A recent paper published in Nature Machine Intelligence demonstrates that video-based autonomous vehicle control tasks can be done with just 19 control neurons (Lechner et al. 2020). Also, for smart transportation applications, there are often some characteristics of the scene that can be utilized to innovate the models for high efficiency, e.g., the road mask extraction (Chapter 5) and the near-crash detection algorithms (Chapter 6 & 7).

Large-scale camera coordination and data fusion: When the application scale becomes significantly larger, there rises opportunities and practical challenges. Larger scale means greater information. City-scale camera coordination can lift the intelligence to another level but requires innovative methods and enormous computation power. Traffic video analytics tasks using city-scale cameras, e.g., vehicle re-identification and multi-camera vehicle tracking, are far more computationally expensive than single camera video processing, thus presenting new challenges for real-time capability, especially if operated on the edge. In addition to intra-class coordination, video data fusion with other data sources, e.g., probe vehicle data and mobile phone data, is also a promising research direction. A lot of data sources that have the potential to benefit smart city applications already exist and are being continuously produced. There are just not enough

motivations or effective approaches for the data holders and agencies to share the data. Additionally, the failure probability could turn from negligible for individual sensors into a norm for large-scale sensor networks according to probability theory. One key reason that fully autonomous driving is still not even close to reality is the unreliable sensing performance in corner cases and large scale. Therefore, another research question remains to be answered is: how do we address sensor failures to ensure safety when the applications scale up?

BIBLIOGRAPHY

Abdulrahim, Khairi, Rosalina Abdul Salam, and others. 2016. “Cumulative Frame Differencing for Urban Vehicle Detection.” In *First International Workshop on Pattern Recognition*, , 100110G.

Al-Turjman, Fadi, and Arman Malekloo. 2019. “Smart Parking in IoT-Enabled Cities: A Survey.” *Sustainable Cities and Society* 49: 101608. http://www.sciencedirect.com/science/article/pii/S2210670718327173.

Alam, Muhammad et al. 2018. “Real-Time Smart Parking Systems Integration in Distributed ITS for Smart Cities.” *Journal of Advanced Transportation* 2018. https://www.hindawi.com/journals/jat/2018/1485652/.

Allodi, Marco et al. 2016. “Machine Learning in Tracking Associations with Stereo Vision and Lidar Observations for an Autonomous Vehicle.” *IEEE Intelligent Vehicles Symposium, Proceedings* 2016-Augus(Iv): 648–53.

Amato, Giuseppe et al. 2016. “Car Parking Occupancy Detection Using Smart Camera Networks and Deep Learning.” In *2016 IEEE Symposium on Computers and Communication (ISCC)*, IEEE, 1212–17. http://ieeexplore.ieee.org/document/7543901/.

Amato, Giuseppe et al. 2017. “Deep Learning for Decentralized Parking Lot Occupancy Detection.” *Expert Systems with Applications* 72: 327–34. https://www.sciencedirect.com/science/article/pii/S095741741630598X.

Ammour, Nassim et al. 2017. “Deep Learning Approach for Car Detection in UAV Imagery.” *Remote Sensing* 9(4): 312.

Ananthanarayanan, Ganesh et al. 2017. “Real-Time Video Analytics: The Killer App for Edge Computing.” *computer* 50(10): 58–67.

Angel, Alejandro, Mark Hickman, Pitu Mirchandani, and Dinesh Chandnani. 2003. “Methods of Analyzing Traffic Imagery Collected from Aerial Platforms.” *IEEE Transactions on Intelligent Transportation Systems* 4(2): 99–107.

Asadi, Reza, and Amelia Regan. 2019. “A Convolution Recurrent Autoencoder for Spatio-Temporal Missing Data Imputation.” *arXiv preprint arXiv:1904.12413*.

Ashqar, Huthaifa I., Mohammed H. Almannaa, Mohammed Elhenawy, and Hesham A. Rakha. 2020. “Smartphone Transportation Mode Recognition Using a Hierarchical Machine Learning Classifier and Pooled Features from Time and Frequency Domains.” *arXiv* 20(1):

244–52.

Avola, Danilo et al. 2020. “A UAV Video Dataset for Mosaicking and Change Detection from Low-Altitude Flights.” *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 50(6): 2139–49.

Ban, Xuegang Jeff, Peng Hao, and Zhanbo Sun. 2011. “Real Time Queue Length Estimation for Signalized Intersections Using Travel Times from Mobile Sensors.” *Transportation Research Part C: Emerging Technologies* 19(6): 1133–56.

Ban, Xuegang Jeff, Ryan Herring, J D Margulici, and Alexandre M Bayen. 2009. “Optimal Sensor Placement for Freeway Travel Time Estimation.” In *Transportation and Traffic Theory 2009: Golden Jubilee*, Springer, 697–721.

Barmpounakis, Emmanouil N., Eleni I. Vlahogianni, and John C. Golias. 2016. “Unmanned Aerial Aircraft Systems for Transportation Engineering: Current Practice and Future Challenges.” *International Journal of Transportation Science and Technology* 5(3): 111–22.

Barmpounakis, Emmanouil N, Eleni I Vlahogianni, John C Golias, and Adam Babinec. 2017. “How Accurate Are Small Drones for Measuring Microscopic Traffic Parameters?” *Transportation Letters*: 1–9.

Baroffio, Luca et al. 2015. “A Visual Sensor Network for Parking Lot Occupancy Detection in Smart Cities.” In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, , 745–50. https://ieeexplore.ieee.org/abstract/document/7389147.

Barthélemy, Johan, Nicolas Verstaevel, Hugh Forehead, and Pascal Perez. 2019. “Edge-Computing Video Analytics for Real-Time Traffic Monitoring in a Smart City.” *Sensors* 19(9): 2048.

Bewley, Alex et al. 2016. “Simple Online and Realtime Tracking.” In *2016 IEEE International Conference on Image Processing (ICIP)*, , 3464–68.

Bhaskar, Prem Kumar, and Suet-Peng Yong. 2014. “Image Processing Based Vehicle Detection and Tracking Method.” In *2014 International Conference on Computer and Information Sciences (ICCOINS)*, , 1–5.

Bhat, Goutam et al. 2018. “Unveiling the Power of Deep Tracking.” In *Proceedings of the European Conference on Computer Vision (ECCV)*, , 483–98.

Bochinski, Erik, Tobias Senst, and Thomas Sikora. 2018. “Extending IOU Based Multi-Object

Tracking by Visual Information." In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, , 1–6.

Bock, F, and S Di Martino. 2017. "How Many Probe Vehicles Do We Need to Collect On-Street Parking Information?" In *2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, , 538–43.

Bock, F, S Di Martino, and A Origlia. 2019. "Smart Parking: Using a Crowd of Taxis to Sense On-Street Parking Space Availability." *IEEE Transactions on Intelligent Transportation Systems*: 1–13. https://ieeexplore.ieee.org/abstract/document/8667435/metrics#metrics.

Bock, Fabian, Sergio Di Martino, and Monika Sester. 2017. "Data-Driven Approaches for Smart Parking." In *Machine Learning and Knowledge Discovery in Databases*, eds. Yasemin Altun et al. Cham: Springer International Publishing, 358–62.

Bolte, J, A Bar, D Lipinski, and T Fingscheidt. 2019. "Towards Corner Case Detection for Autonomous Driving." In *2019 IEEE Intelligent Vehicles Symposium (IV)*, , 438–45.

Bottou, Léon. 2010. "Large-Scale Machine Learning with Stochastic Gradient Descent." In *Proceedings of COMPSTAT'2010*, Springer, 177–86.

Bouttefroy, Philippe Loic Marie, Abdesselam Bouzerdoum, Son Lam Phung, and Azeddine Beghdadi. 2008. "Vehicle Tracking by Non-Drifting Mean-Shift Using Projective Kalman Filter." In *2008 11th International IEEE Conference on Intelligent Transportation Systems*, , 61–66.

Breckon, Toby P, Stuart E Barnes, Marcin L Eichner, and Ken Wahren. 2009. "Autonomous Real-Time Vehicle Detection from a Medium-Level UAV." In *Proc. 24th International Conference on Unmanned Air Vehicle Systems*, , 21–29.

Broggi, Alberto et al. 2013. "Extensive Tests of Autonomous Driving Technologies." *IEEE Transactions on Intelligent Transportation Systems* 14(3): 1403–15. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6522193.

Broggi, Alberto et al. 2015. "PROUD-Public Road Urban Driverless-Car Test." *IEEE Transactions on Intelligent Transportation Systems* 16(6): 3508–19.

Van Brummelen, Jessica, Marie O'Brien, Dominique Gruyer, and Homayoun Najjaran. 2018. "Autonomous Vehicle Perception: The Technology of Today and Tomorrow." *Transportation Research Part C: Emerging Technologies* 89(July 2017): 384–406. https://doi.org/10.1016/j.trc.2018.02.012.

Buch, Norbert, Sergio A. Velastin, and James Orwell. 2011. “A Review of Computer Vision Techniques for the Analysis of Urban Traffic.” *IEEE Transactions on Intelligent Transportation Systems* 12(3): 920–39.

Bulan, Orhan et al. 2013. “Video-Based Real-Time on-Street Parking Occupancy Detection System.” *Journal of Electronic Imaging* 22(4): 41109. https://www.spiedigitallibrary.org/journals/Journal-of-Electronic-Imaging/volume-22/issue-4/041109/Video-based-real-time-on-street-parking-occupancy-detection-system/10.1117/1.JEI.22.4.041109.full?SSO=1.

Canny, John. 1986. “A Computational Approach to Edge Detection.” *IEEE Transactions on pattern analysis and machine intelligence* (6): 679–98.

Cao, Chuqing, and Ying Sun. 2014. “Automatic Road Centerline Extraction from Imagery Using Road GPS Data.” *Remote sensing* 6(9): 9014–33.

Cao, Xianbin et al. 2011. “Vehicle Detection and Motion Analysis in Low-Altitude Airborne Video under Urban Environment.” *IEEE Transactions on Circuits and Systems for Video Technology* 21(10): 1522–33.

Cao Xianbin. 2014. “Ego Motion Guided Particle Filter for Vehicle Tracking in Airborne Videos.” *Neurocomputing* 124: 168–77.

Cao, Xianbin, Jinhe Lan, Pingkun Yan, and Xuelong Li. 2012. “Vehicle Detection and Tracking in Airborne Videos by Multi-Motion Layer Analysis.” *Machine Vision and Applications* 23(5): 921–35.

Carletti, Vincenzo, Antonio Greco, Alessia Saggese, and Mario Vento. 2018. “Multi-Object Tracking by Flying Cameras Based on a Forward-Backward Interaction.” *IEEE Access* 6: 43905–19.

Chang, Ming Ching et al. 2020. “AI City Challenge 2020 - Computer Vision for Smart Transportation Applications.” *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* 2020-June: 2638–47.

Chen, Mingliang et al. 2017. “Spatiotemporal GMM for Background Subtraction with Superpixel Hierarchy.” *IEEE transactions on pattern analysis and machine intelligence* 40(6): 1518–25.

Chen, Peng, Weiliang Zeng, Guizhen Yu, and Yunpeng Wang. 2017. “Surrogate Safety Analysis of Pedestrian-Vehicle Conflict at Intersections Using Unmanned Aerial Vehicle Videos.”

Journal of advanced transportation 2017.

Chen, Xinqiang et al. 2020. “Sensing Data Supported Traffic Flow Prediction via Denoising Schemes and ANN: A Comparison.” *IEEE Sensors Journal* 20(23): 14317–28.

Chen, Xinyu, Jinming Yang, and Lijun Sun. 2020. “A Nonconvex Low-Rank Tensor Completion Model for Spatiotemporal Traffic Data Imputation.” *Transportation Research Part C: Emerging Technologies* 117(June): 102673. https://doi.org/10.1016/j.trc.2020.102673.

Chen, Yuanfang et al. 2019. “Deep Learning for Secure Mobile Edge Computing in Cyber-Physical Transportation Systems.” *IEEE Network* 33(4): 36–41.

Chen, Yuanyuan, Yisheng Lv, and Fei Yue Wang. 2020. “Traffic Flow Imputation Using Parallel Data and Generative Adversarial Networks.” *IEEE Transactions on Intelligent Transportation Systems* 21(4): 1624–30.

Chen, Zhaoxue., and Pengfei. Shi. 2004. “Efficient Method for Camera Calibration in Traffic Scenes.” *Electronics Letters* 40(6): 368–69.

Chen, Zhenyu et al. 2013. “Online Sequential ELM Based Transfer Learning for Transportation Mode Recognition.” *Proceedings of the 2013 IEEE Conference on Cybernetics and Intelligent Systems, CIS 2013*: 78–83.

Cheung, Sing Yiu et al. 2005. “Traffic Measurement and Vehicle Classification with Single Magnetic Sensor.” *Transportation Research Record* (1917): 173–81.

Cho, Woon et al. 2018. “Robust Parking Occupancy Monitoring System Using Random Forests.” In *2018 International Conference on Electronics, Information, and Communication (ICEIC)*, , 1–4.

Chou, Glen et al. 2018. “Using Control Synthesis to Generate Corner Cases: A Case Study on Autonomous Driving.” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37(11): 2906–17.

Coifman, B, M McCord, R G Mishalani, and K Redmill. 2004. “Surface Transportation Surveillance from Unmanned Aerial Vehicles.” In *Proc. of the 83rd Annual Meeting of the Transportation Research Board,*.

Cui, Z. et al. 2020. “Learning Traffic as a Graph: A Gated Graph Wavelet Recurrent Neural Network for Network-Scale Traffic Prediction.” *Transportation Research Part C: Emerging Technologies* 115.

Cui, Zhiyong, Kristian Henrickson, Ruimin Ke, and Yinhai Wang. 2019. “Traffic Graph

Convolutional Recurrent Neural Network: A Deep Learning Framework for Network-Scale Traffic Learning and Forecasting." *IEEE Transactions on Intelligent Transportation Systems*.

D'Amour, Alexander et al. 2020. "Underspecification Presents Challenges for Credibility in Modern Machine Learning." *arXiv preprint arXiv:2011.03395*.

Dagan, Erez, Ofer Mano, Gideon P. Stein, and Amnon Shashua. 2004. "Forward Collision Warning with a Single Camera." *IEEE Intelligent Vehicles Symposium, Proceedings*: 37–42.

Dai, Yueyue et al. 2019. "Artificial Intelligence Empowered Edge Computing and Caching for Internet of Vehicles." (June): 12–18.

Datondji, Sokemi Rene Emmanuel, Yohan Dupuis, Peggy Subirats, and Pascal Vasseur. 2016. "A Survey of Vision-Based Traffic Monitoring of Road Intersections." *IEEE Transactions on Intelligent Transportation Systems* 17(10): 2681–98.

Dogru, Nejdet, and Abdulhamit Subasi. 2018. "Traffic Accident Detection Using Random Forest Classifier." *2018 15th Learning and Technology Conference, L and T 2018*: 40–45.

Du, Dawei et al. 2018. "The Unmanned Aerial Vehicle Benchmark: Object Detection and Tracking." In *Proceedings of the European Conference on Computer Vision (ECCV), ,* 370–86.

Du, Yuchuan, Cong Zhao, Feng Li, and Xuefeng Yang. 2017. "An Open Data Platform for Traffic Parameters Measurement via Multirotor Unmanned Aerial Vehicles Video." *Journal of Advanced Transportation* 2017.

Duan, Yanjie, Yisheng Lv, Yu-Liang Liu, and Fei-Yue Wang. 2016. "An Efficient Realization of Deep Learning for Traffic Data Imputation." *Transportation research part C: emerging technologies* 72: 168–81.

Duda, Richard O, and Peter E Hart. 1971. *Use of the Hough Transformation to Detect Lines and Curves in Pictures*.

Dunlap, Matthew, Zhibin Li, Kristian Henrickson, and Yinhai Wang. 2016. "Estimation of Origin and Destination Information from Bluetooth and Wi-Fi Sensing for Transit." *Transportation Research Record* 2595(2595): 11–17.

El-wakeel, Amr S et al. 2018. "Towards a Practical Crowdsensing System for Road Surface Conditions Monitoring." 5(6): 4672–85.

Ester, Martin et al. 1996. “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.” In *Kdd, ,* 226–31.

Everingham, Mark et al. 2010. “The Pascal Visual Object Classes (Voc) Challenge.” *International journal of computer vision* 88(2): 303–38.

Feng, Shuo et al. 2018. “A Better Understanding of Long-Range Temporal Dependence of Traffic Flow Time Series.” *Physica A: Statistical Mechanics and its Applications* 492: 639–50.

Feng, Shuo, Yiheng Feng, Xintao Yan, et al. 2020. “Safety Assessment of Highly Automated Driving Systems in Test Tracks: A New Framework.” *Accident Analysis & Prevention* 144: 105664. http://www.sciencedirect.com/science/article/pii/S0001457520302621.

Feng, Shuo, Yiheng Feng, Chunhui Yu, et al. 2020. “Testing Scenario Library Generation for Connected and Automated Vehicles, Part i: Methodology.” *IEEE Transactions on Intelligent Transportation Systems*.

Feng, Shuo, Yiheng Feng, Haowei Sun, et al. 2020. “Testing Scenario Library Generation for Connected and Automated Vehicles, Part II: Case Studies.” *IEEE Transactions on Intelligent Transportation Systems*.

Feng, Yiheng et al. 2018. “An Augmented Reality Environment for Connected and Automated Vehicle Testing and Evaluation.” In *2018 IEEE Intelligent Vehicles Symposium (IV), ,* 1549–54.

Ferdowsi, Aidin, Ursula Challita, and Walid Saad. 2019. “Deep Learning for Reliable Mobile Edge Analytics in Intelligent Transportation Systems: An Overview.” *ieee vehicular technology magazine* 14(1): 62–70.

Freeman, Brian S. et al. 2019. “Vehicle Stacking Estimation at Signalized Intersections with Unmanned Aerial Systems.” *International Journal of Transportation Science and Technology* 8(2): 231–49.

Fremont, Daniel J et al. 2020. “Formal Scenario-Based Testing of Autonomous Vehicles: From Simulation to the Real World.” https://arxiv.org/abs/2003.07739.

Fritsch, Jannik, Tobias Kuehnl, and Andreas Geiger. 2013. “A New Performance Measure and Evaluation Benchmark for Road Detection Algorithms.” In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013), ,* 1693–1700.

Gao, Tao, Zheng-guang Liu, Wen-chun Gao, and Jun Zhang. 2008. “A Robust Technique for

Background Subtraction in Traffic Video." In *International Conference on Neural Information Processing*, , 736–44.

Garg, Sahil et al. 2018. "UAV-Empowered Edge Computing Environment for Cyber-Threat Detection in Smart Vehicles." *IEEE Network* 32(3): 42–51.

Gaszczak, Anna, Toby P Breckon, and Jiwan Han. 2011. "Real-Time People and Vehicle Detection from UAV Imagery." In *Intelligent Robots and Computer Vision XXVIII: Algorithms and Techniques*, , 78780B.

Geiger, Andreas, Philip Lenz, Christoph Stiller, and Raquel Urtasun. 2013. "Vision Meets Robotics: The Kitti Dataset." *The International Journal of Robotics Research* 32(11): 1231–37.

Geiger, Andreas, Philip Lenz, and Raquel Urtasun. 2012. "Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite." *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*: 3354–61.

de Gelder, Erwin, Jan Pieter Paardekooper, Olaf Op den Camp, and Bart De Schutter. 2019. "Safety Assessment of Automated Vehicles: How to Determine Whether We Have Collected Enough Field Data?" *Traffic Injury Prevention* 20(sup1): S162–70. https://doi.org/10.1080/15389588.2019.1602727.

Geng, Xu et al. 2019. "Spatiotemporal Multi-Graph Convolution Network for Ride-Hailing Demand Forecasting." In *Proceedings of the AAAI Conference on Artificial Intelligence*, , 3656–63.

Goh, C. Y. et al. 2012. "Online Map-Matching Based on Hidden Markov Model for Real-Time Traffic Sensing Applications." *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*: 776–81.

Gomaa, Ahmed, Moataz M Abdelwahab, and Mohammed Abo-Zahhad. 2018. "Real-Time Algorithm for Simultaneous Vehicle Detection and Tracking in Aerial View Videos." In *2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, , 222–25.

Grassi, Giulio, Kyle Jamieson, Paramvir Bahl, and Giovanni Pau. 2017. "Parkmaster: An in-Vehicle, Edge-Based Video Analytics Service for Detecting Open Parking Spaces in Urban Environments." In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, , 16.

Grodi, Robin, Danda B Rawat, and Fernando Rios-Gutierrez. 2016. “Smart Parking: Parking Occupancy Monitoring and Visualization System for Smart Cities.” In *SoutheastCon 2016*, , 1–5.

Gu, Steve, Ying Zheng, and Carlo Tomasi. 2011. “Linear Time Offline Tracking and Lower Envelope Algorithms.” In *2011 International Conference on Computer Vision*, , 1840–46.

Guan, Haiyan et al. 2018. “Robust Traffic-Sign Detection and Classification Using Mobile LiDAR Data with Digital Images.” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 11(5): 1715–24.

Guido, Giuseppe, Vincenzo Gallelli, Daniele Rogano, and Alessandro Vitale. 2016. “Evaluating the Accuracy of Vehicle Tracking Data Obtained from Unmanned Aerial Vehicles.” *International journal of transportation science and technology* 5(3): 136–51.

Guo, Shengnan et al. 2019. “Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting.” In *Proceedings of the AAAI Conference on Artificial Intelligence*, , 922–29.

Gupte, Surendra, Osama Masoud, Robert F K Martin, and Nikolaos P Papanikolopoulos. 2002. “Detection and Classification of Vehicles.” *IEEE Transactions on intelligent transportation systems* 3(1): 37–47.

Han, Feng et al. 2006. “A Two-Stage Approach to People and Vehicle Detection with Hog-Based Svm.” In *Performance Metrics for Intelligent Systems 2006 Workshop*, , 133–40.

Han, Song, Huizi Mao, and William J Dally. 2015. “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding.” *arXiv preprint arXiv:1510.00149*.

Haoui, Amine, Robert Kavaler, and Pravin Varaiya. 2008. “Wireless Magnetic Sensors for Traffic Surveillance.” *Transportation Research Part C: Emerging Technologies* 16(3): 294–306.

Harper, Corey D, Chris T Hendrickson, and Constantine Samaras. 2016. “Cost and Benefit Estimates of Partially-Automated Vehicle Collision Avoidance Technologies.” *Accident Analysis & Prevention* 95: 104–15.

He, Ying et al. 2017. “Software-Defined Networks with Mobile Edge Computing and Caching for Smart Cities : A Big Data Deep Reinforcement Learning Approach.” (December): 31–37.

Helala, Mohamed A, Ken Q Pu, and Faisal Z Qureshi. 2012. “Road Boundary Detection in Challenging Scenarios.” In *2012 IEEE Ninth International Conference on Advanced Video and Signal-Based Surveillance, ,* 428–33.

Herrera, Juan C. et al. 2010. “Evaluation of Traffic Data Obtained via GPS-Enabled Mobile Phones: The Mobile Century Field Experiment.” *Transportation Research Part C: Emerging Technologies* 18(4): 568–83. http://dx.doi.org/10.1016/j.trc.2009.10.006.

Hillel, Aharon Bar, Ronen Lerner, Dan Levi, and Guy Raz. 2014. “Recent Progress in Road and Lane Detection: A Survey.” *Machine vision and applications* 25(3): 727–45.

Horváth, Márton Tamás et al. 2019. “Vehicle-In-The-Loop (VIL) and Scenario-In-The-Loop (SCIL) Automotive Simulation Concepts from the Perspectives of Traffic Simulation and Traffic Control.” *Transport and Telecommunication* 20(2): 153–61.

Hou, Xinyu, Yi Wang, and Lap-Pui Chau. 2019. “Vehicle Tracking Using Deep SORT with Low Confidence Track Filtering.” In *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), ,* 1–6.

Houben, Sebastian et al. 2013. “On-Vehicle Video-Based Parking Lot Recognition with Fisheye Optics.” In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013), ,* 7–12.

Hsu, Hung-Min et al. 2019. “Multi-Camera Tracking of Vehicles Based on Deep Features Re-ID and Trajectory-Based Camera Link Models.” In *CVPR Workshops, ,* 416–24.

Hsu, Ya Wen, Yen Wei Chen, and Jau Woei Perng. 2020. “Estimation of the Number of Passengers in a Bus Using Deep Learning.” *Sensors (Switzerland)* 20(8): 1–18.

“Internet/Broadband Fact Sheet.” 2019. https://www.pewresearch.org/internet/fact-sheet/internet-broadband/.

Ismail, Karim, Tarek Sayed, and Nicolas Saunier. 2010. “Automated Analysis of Pedestrian--Vehicle Conflicts: Context for before-and-after Studies.” *Transportation research record* 2198(1): 52–64.

Ismail, Karim, Tarek Sayed, Nicolas Saunier, and Clark Lim. 2009. “Automated Analysis of Pedestrian--Vehicle Conflicts Using Video Data.” *Transportation research record* 2140(1): 44–54.

Jahangiri, Arash, and Hesham A Rakha. 2015. “Transportation Mode Recognition Using Mobile Phone Sensor Data.” *Ieee Transactions on Intelligent Transportation Systems* 16(5): 2406–

17.

Jazayeri, Amirali, Hongyuan Cai, Jiang Yu Zheng, and Mihran Tuceryan. 2011. “Vehicle Detection and Tracking in Car Video Based on Motion Model.” *IEEE Transactions on Intelligent Transportation Systems* 12(2): 583–95.

Jeon, YongSung, Hong-Il Ju, and Seungyong Yoon. 2018. “Design of an Lpwan Communication Module Based on Secure Element for Smart Parking Application.” In *2018 IEEE International Conference on Consumer Electronics (ICCE)*, , 1–2. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8326112.

Jeyakumar, Jeya Vikranth et al. 2018. “Deep Convolutional Bidirectional LSTM Based Transportation Mode Recognition.” *UbiComp/ISWC 2018 - Adjunct Proceedings of the 2018 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2018 ACM International Symposium on Wearable Computers*: 1606–15.

Jung, Jiyoung, and Sung Ho Bae. 2018. “Real-Time Road Lane Detection in Urban Areas Using LiDAR Data.” *Electronics (Switzerland)* 7(11): 1–14.

Kalinke, Thomas, Christos Tzomakas, and Wemer von Seelen. 1998. “A Texture-Based Object Detection and an Adaptive Model-Based Classification.” In *Procs. IEEE Intelligent Vehicles Symposium*, , 341–46.

Kamyab, Mohsen et al. 2020. “Machine Learning Approach to Forecast Work Zone Mobility Using Probe Vehicle Data.” *Transportation Research Record* 2674(9): 157–67.

Kanhere, Neeraj K, Stanley T Birchfield, and Wayne A Sarasua. 2008. “Automatic Camera Calibration Using Pattern Detection for Vision-Based Speed Sensing.” *Transportation research record* 2086(1): 30–39.

Kanistras, Konstantinos, Goncalo Martins, Matthew J. Rutherford, and Kimon P. Valavanis. 2015. “Survey of Unmanned Aerial Vehicles (Uavs) for Traffic Monitoring.” *Handbook of Unmanned Aerial Vehicles*: 2643–66.

Kataoka, Hirokatsu et al. 2018. “Drive Video Analysis for the Detection of Traffic Near-Miss Incidents.” In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, , 1–8.

Kaufmann, Stefan et al. 2018. “Aerial Observations of Moving Synchronized Flow Patterns in Over-Saturated City Traffic.” *Transportation research part C: emerging technologies* 86: 393–406.

Ke, Jintao, Hongyu Zheng, Hai Yang, and Xiqun Michael. 2017. “Short-Term Forecasting of Passenger Demand under on-Demand Ride Services : A Spatio-Temporal Deep Learning Approach.” *Transportation Research Part C* 85(October): 591–608. http://dx.doi.org/10.1016/j.trc.2017.10.016.

Ke, R., J. Lutin, J. Spears, and Y. Wang. 2017. “A Cost-Effective Framework for Automated Vehicle-Pedestrian Near-Miss Detection Through Onboard Monocular Vision.” In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*,.

Ke, Ruimin. 2016. “A Novel Framework for Real-Time Traffic Flow Parameter Estimation from Aerial Videos.” University of Washington.

Ke, Ruimin, Zhibin Li, et al. 2017. “Real-Time Bidirectional Traffic Flow Parameter Estimation from Aerial Videos.” *IEEE Transactions on Intelligent Transportation Systems* 18(4): 890–901.

Ke, Ruimin et al. 2018. “Real-Time Traffic Flow Parameter Estimation from UAV Video Based on Ensemble Classifier and Optical Flow.” *IEEE Transactions on Intelligent Transportation Systems* (99): 1–11.

Ke, Ruimin, Shuo Feng, Zhiyong Cui, and Yinhai Wang. 2020. “Advanced Framework for Microscopic and Lane-Level Macroscopic Traffic Parameters Estimation from UAV Video.” *IET Intelligent Transport Systems*.

Ke, Ruimin, Sung Kim, Zhibin Li, and Yinhai Wang. 2015. “Motion-Vector Clustering for Traffic Speed Detection from UAV Video.” In *Smart Cities Conference (ISC2), 2015 IEEE First International, ,* 1–5.

Ke, Ruimin, Wan Li, Zhiyong Cui, and Yinhai Wang. 2020. “Two-Stream Multi-Channel Convolutional Neural Network for Multi-Lane Traffic Speed Prediction Considering Traffic Volume Impact.” *Transportation Research Record* 2674(4): 459–70.

Ke, Ruimin, Jerome Lutin, Jerry Spears, and Yinhai Wang. 2017. “A Cost-Effective Framework for Automated Vehicle-Pedestrian Near-Miss Detection Through Onboard Monocular Vision.” In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*,.

Ke, Ruimin, Zewen Pan, Ziyuan Pu, and Yinhai Wang. 2017. “Roadway Surveillance Video Camera Calibration Using Standard Shipping Container.” In *Smart Cities Conference (ISC2), 2017 International, ,* 1–6.

Ke, Ruimin, Yifan Zhuang, Ziyuan Pu, and Yinhai Wang. 2020. “A Smart, Efficient, and Reliable Parking Surveillance System with Edge Artificial Intelligence on IoT Devices.” *IEEE Transactions on Intelligent Transportation Systems*.

Khammari, Ayoub, Fawzi Nashashibi, Yotam Abramson, and Claude Laurgeau. 2005. “Vehicle Detection Combining Gradient Analysis and AdaBoost Classification.” In *Proceedings. 2005 IEEE Intelligent Transportation Systems, 2005.*, , 66–71.

Khan, Muhammad et al. 2018. “Unmanned Aerial Vehicle-Based Traffic Analysis: A Case Study for Shockwave Identification and Flow Parameters Estimation at Signalized Intersections.” *Remote Sensing* 10(3): 458.

Khan, Muhammad Arsalan et al. 2017. “Unmanned Aerial Vehicle--Based Traffic Analysis: Methodological Framework for Automated Multivehicle Trajectory Extraction.” *Transportation Research Record: Journal of the Transportation Research Board* (2626): 25–33.

Kilicarslan, Mehmet, and Jiang Yu Zheng. 2019. “Predict Vehicle Collision by TTC from Motion Using a Single Video Camera.” *IEEE Transactions on Intelligent Transportation Systems* 20(2): 522–33.

Kim, Eui-Jin et al. 2019. “Extracting Vehicle Trajectories Using Unmanned Aerial Vehicles in Congested Traffic Conditions.” *Journal of Advanced Transportation* 2019.

Klauer, Sheila G et al. 2006. “The Impact of Driver Inattention on Near-Crash/Crash Risk: An Analysis Using the 100-Car Naturalistic Driving Study Data.”

Kong, Qing-Jie, Lucidus Zhou, Gang Xiong, and Fenghua Zhu. 2013. “Automatic Road Detection for Highway Surveillance Using Frequency-Domain Information.” In *Proceedings of 2013 IEEE International Conference on Service Operations and Logistics, and Informatics*, , 24–28.

Kong, Zelun, Junfeng Guo, Ang Li, and Cong Liu. 2020. “PhysGAN: Generating Physical-World-Resilient Adversarial Examples for Autonomous Driving.” In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, , 14254–63.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. 2017. “Imagenet Classification with Deep Convolutional Neural Networks.” *Communications of the ACM* 60(6): 84–90.

Kulkarni, Aniket, Nitish Mhalgi, Sagar Gurnani, and Nupur Giri. 2014. “Pothole Detection System Using Machine Learning on Android.” *International Journal of Emerging*

Technology and Advanced Engineering 4(7): 360–64.

Kumar, Niranjil, and C Sureshkumar. 2015. "Background Subtraction in Dynamic Environment Based on Modified Adaptive GMM with TTD for Moving Object Detection." *Journal of Electrical Engineering & Technology* 10(1): 372–78.

Kumar, Pankaj, Surendra Ranganath, and Huang Weimin. 2003. "Bayesian Network Based Computer Vision Algorithm for Traffic Monitoring Using Video." In *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems, ,* 897–902.

Kunz, Felix et al. 2015. "Autonomous Driving at Ulm University: A Modular, Robust, and Sensor-Independent Fusion Approach." *IEEE Intelligent Vehicles Symposium, Proceedings* 2015-Augus(Iv): 666–73.

Lai, Andrew H S, and Nelson H C Yung. 2000. "Lane Detection by Orientation and Length Discrimination." *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 30(4): 539–48.

Lechner, Mathias et al. 2020. "Neural Circuit Policies Enabling Auditable Autonomy." *Nature Machine Intelligence* 2(10): 642–52.

Lee, Sangwon, Dukhee Yoon, and Amitabha Ghosh. 2008. "Intelligent Parking Lot Application Using Wireless Sensor Networks." In *CTS, ,* 48–57.

Levin, Blair, and Larry Downes. 2019. "Cities, Not Rural Areas, Are the Real Internet Deserts." https://www.washingtonpost.com/technology/2019/09/13/cities-not-rural-areas-are-real-internet-deserts/.

Li, Jing et al. 2016. "Multi-Target Detection and Tracking from a Single Camera in Unmanned Aerial Vehicles (UAVs)." In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), ,* 4992–97.

Li, Jing et al. 2019. "An Adaptive Framework for Multi-Vehicle Ground Speed Estimation in Airborne Videos." *Remote Sensing* 11(10): 1241.

Li, Li et al. 2019. "Parallel Testing of Vehicle Intelligence via Virtual-Real Interaction."

Li, Siyi, and Dit Yan Yeung. 2017. "Visual Object Tracking for Unmanned Aerial Vehicles: A Benchmark and New Motion Models." *31st AAAI Conference on Artificial Intelligence, AAAI 2017*: 4140–46.

Li, Xing, and Xiaosong Guo. 2013. "A HOG Feature and SVM Based Method for Forward Vehicle Detection with Single Camera." In *2013 5th International Conference on Intelligent*

Human-Machine Systems and Cybernetics, , 263–66.

Li, Yuantao, Fenghua Zhu, Yunfeng Ai, and Fei-Yue Wang. 2007. "On Automatic and Dynamic Camera Calibration Based on Traffic Visual Surveillance." In *2007 IEEE Intelligent Vehicles Symposium*, , 358–63.

Lin, Lei, Zhengbing He, and Srinivas Peeta. 2018. "Predicting Station-Level Hourly Demand in a Large-Scale Bike- Sharing Network : A Graph Convolutional Neural Network Approach ☆." *Transportation Research Part C* 97(November): 258–76. https://doi.org/10.1016/j.trc.2018.10.011.

Lin, T, H Rivano, and F Le Mouël. 2017. "A Survey of Smart Parking Solutions." *IEEE Transactions on Intelligent Transportation Systems* 18(12): 3229–53. https://ieeexplore.ieee.org/abstract/document/7895130.

Ling, Xiao et al. 2017. "Identifying Parking Spaces & Detecting Occupancy Using Vision-Based IoT Devices." In *2017 Global Internet of Things Summit (GIoTS)*, , 1–6. https://ieeexplore.ieee.org/abstract/document/8016227/metrics#metrics.

Liu, Qun, Suman Kumar, and Vijay Mago. 2020. "SafeRNet : Safe Transportation Routing in the Era of Internet of Vehicles and Mobile Crowd Sensing." : 299–304.

Liu, Wei et al. "SSD : Single Shot MultiBox Detector."

Loewenherz, Franz, Victor Bahl, and Yinhai Wang. 2017. "Video Analytics towards Vision Zero." *Institute of Transportation Engineers. ITE Journal* 87(3): 25.

Lou, Liangliang, Jinyi Zhang, Yong Xiong, and Yanliang Jin. 2019. "An Improved Roadside Parking Space Occupancy Detection Method Based on Magnetic Sensors and Wireless Signal Strength." *Sensors* 19(10): 2348. https://www.mdpi.com/1424-8220/19/10/2348.

Lu, Xiaofeng, Takashi Izumi, Tomoaki Takahashi, and Lei Wang. 2014. "Moving Vehicle Detection Based on Fuzzy Background Subtraction." In *2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, , 529–32.

Lucas, Bruce D, Takeo Kanade, and others. 1981. "An Iterative Image Registration Technique with an Application to Stereo Vision."

Luo, Qi, Romesh Saigal, Robert Hampshire, and Xinyi Wu. 2017. "A Statistical Method for Parking Spaces Occupancy Detection via Automotive Radars." In *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, , 1–5.

Luo, Zhiming et al. 2018. "MIO-TCD: A New Benchmark Dataset for Vehicle Classification and

Localization.” *IEEE Transactions on Image Processing* 27(10): 5129–41.

Lv, Fengjun, Tao Zhao, and Ramakant Nevatia. 2006. “Camera Calibration from Video of a Walking Human.” *IEEE transactions on pattern analysis and machine intelligence* 28(9): 1513–18.

Lv, Yisheng et al. 2015. “Traffic Flow Prediction with Big Data: A Deep Learning Approach.” *IEEE Transactions on Intelligent Transportation Systems*.

Ma, Jiaqi et al. 2018. “Hardware-in-the-Loop Testing of Connected and Automated Vehicle Applications: A Use Case for Queue-Aware Signalized Intersection Approach and Departure.” *Transportation Research Record* 2672(22): 36–46.

Ma, Xiaolei et al. 2015. “Long Short-Term Memory Neural Network for Traffic Speed Prediction Using Remote Microwave Sensor Data.” *Transportation Research Part C: Emerging Technologies*.

Makizako, Hyuma et al. 2018. “Associations of Near-Miss Traffic Incidents with Attention and Executive Function among Older Japanese Drivers.” *Gerontology* 64: 495–502.

Malinovskiy, Yegor, Nicolas Saunier, and Yinhai Wang. 2012. “Analysis of Pedestrian Travel with Static Bluetooth Sensors.” *Transportation Research Record* (2299): 137–49.

Martel-Brisson, Nicolas, and Andre Zaccarin. 2007. “Learning and Removing Cast Shadows through a Multidistribution Approach.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29(7): 1133–46.

Mathur, Suhas et al. 2010. “Parknet: Drive-by Sensing of Road-Side Parking Statistics.” In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, , 123–36.

McCord, Mark et al. 2003. “Estimating Annual Average Daily Traffic from Satellite Imagery and Air Photos: Empirical Results.” *Transportation Research Record: Journal of the Transportation Research Board* (1855): 136–42.

McCormack, Edward, and Mark E. Hallenbeck. 2006. “ITS Devices Used to Collect Truck Data for Performance Benchmarks.” *Transportation Research Record* (1957): 43–50.

Messelodi, Stefano, Carla Maria Modena, Nicola Segata, and Michele Zanin. 2005. “A Kalman Filter Based Background Updating Algorithm Robust to Sharp Illumination Changes.” In *International Conference on Image Analysis and Processing*, , 163–70.

Micusik, Branislav, and Tomas Pajdla. 2010. “Simultaneous Surveillance Camera Calibration

and Foot-Head Homology Estimation from Human Detections." In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, , 1562–69.

Mitsopoulou, Ellen, and Vana Kalogeraki. 2018. "ParkForU: A Dynamic Parking-Matching and Price-Regulator Crowdsourcing Algorithm for Mobile Applications." In *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, , 603–8. https://ieeexplore.ieee.org/abstract/document/8480321.

Mohamed, Reham, Heba Aly, and Moustafa Youssef. 2017. "Accurate Real-Time Map Matching for Challenging Environments." *IEEE Transactions on Intelligent Transportation Systems* 18(4): 847–57.

Morris, Brendan, and Mohan Trivedi. 2006. "Improved Vehicle Classification in Long Traffic Video by Cooperating Tracker and Classifier Modules." In *2006 IEEE International Conference on Video and Signal Based Surveillance*, , 9.

Muresan, Mircea Paul, and Sergiu Nedevschi. 2018. "Multimodal Sparse LIDAR Object Tracking in Clutter." In *2018 IEEE 14th International Conference on Intelligent Computer Communication and Processing (ICCP)*, , 215–21.

Najiya, K V, and M Archana. 2018. "UAV Video Processing for Traffic Surveillence with Enhanced Vehicle Detection." In *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, , 662–68.

Naphade, Milind et al. 2020. "The 4th AI City Challenge."

Neumann, Daniel et al. 2017. "Online Vehicle Detection Using Haar-like, LBP and HOG Feature Based Image Classifiers with Stereo Vision Preselection." In *2017 IEEE Intelligent Vehicles Symposium (IV)*, , 773–78.

Nieto, Rafael Mart\'\in, Álvaro Garc\'\ia-Mart\'\in, Alexander G Hauptmann, and José M Mart\'\inez. 2018. "Automatic Vacant Parking Places Management System Using Multicamera Vehicle Detection." *IEEE Transactions on Intelligent Transportation Systems* (99): 1–12. https://ieeexplore.ieee.org/abstract/document/8371300/metrics#metrics.

Noh, SeungJong, Daeyoung Shim, and Moongu Jeon. 2015. "Adaptive Sliding-Window Strategy for Vehicle Detection in Highway Environments." *IEEE Transactions on Intelligent Transportation Systems* 17(2): 323–35.

Nothdurft, Tobias et al. 2011. "Stadtpilot: First Fully Autonomous Test Drives in Urban Traffic." *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*: 919–24.

Noursalehi, Peyman, Haris N Koutsopoulos, and Jinhua Zhao. 2018. “Real Time Transit Demand Prediction Capturing Station Interactions and Impact of Special Events ☆.” *Transportation Research Part C* 97(February): 277–300. https://doi.org/10.1016/j.trc.2018.10.023.

Nurullayev, Sherzod, and Sang-Woong Lee. 2019. “Generalized Parking Occupancy Analysis Based on Dilated Convolutional Neural Network.” *Sensors* 19(2): 277. https://www.mdpi.com/1424-8220/19/2/277.

Odat, Enas, Jeff S. Shamma, and Christian Claudel. 2018. “Vehicle Classification and Speed Estimation Using Combined Passive Infrared/Ultrasonic Sensors.” *IEEE Transactions on Intelligent Transportation Systems* 19(5): 1593–1606.

Omer, Raqib, and Liping Fu. 2010. “An Automatic Image Recognition System for Winter Road Surface Condition Classification.” In *13th International IEEE Conference on Intelligent Transportation Systems*, , 1375–79.

Park, Wan-Joo et al. 2008. “Parking Space Detection Using Ultrasonic Sensor in Parking Assistance System.” In *2008 IEEE Intelligent Vehicles Symposium*, , 1039–44.

Parsa, Amir Bahador et al. 2020. “Toward Safer Highways, Application of XGBoost and SHAP for Real-Time Accident Detection and Feature Analysis.” *Accident Analysis and Prevention* 136(August 2019): 105405. https://doi.org/10.1016/j.aap.2019.105405.

Parsa, Amir Bahador, Homa Taghipour, Sybil Derrible, and Abolfazl (Kouros) Mohammadian. 2019. “Real-Time Accident Detection: Coping with Imbalanced Data.” *Accident Analysis and Prevention* 129(May): 202–10. https://doi.org/10.1016/j.aap.2019.05.014.

Peng, Cheng-Fang, Jun-Wei Hsieh, Shao-Wei Leu, and Chi-Hung Chuang. 2018. “Drone-Based Vacant Parking Space Detection.” In *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, , 618–22.

Qi, Charles R. et al. 2017. “Frustum PointNets for 3D Object Detection from RGB-D Data.” *arXiv*.

Raboy, Kelli, Jiaqi Ma, Edward Leslie, and Fang Zhou. 2020. “A Proof-of-Concept Field Experiment on Cooperative Lane Change Maneuvers Using a Prototype Connected Automated Vehicle Testing Platform.” *Journal of Intelligent Transportation Systems*: 1–16. https://www.tandfonline.com/doi/full/10.1080/15472450.2020.1775085.

Rahim, Hasliza A et al. 2010. “Vehicle Speed Detection Using Frame Differencing for Smart

Surveillance System." In *10th International Conference on Information Science, Signal Processing and Their Applications (ISSPA 2010)*, , 630–33.

Rastegari, Mohammad, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. "Xnor-Net: Imagenet Classification Using Binary Convolutional Neural Networks." In *European Conference on Computer Vision*, , 525–42.

Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. "You Only Look Once: Unified, Real-Time Object Detection." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, , 779–88.

Redmon, Joseph, and Ali Farhadi. 2018. "Yolov3: An Incremental Improvement." *arXiv preprint arXiv:1804.02767*.

Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun. 2015. "Faster R-Cnn: Towards Real-Time Object Detection with Region Proposal Networks." In *Advances in Neural Information Processing Systems*, , 91–99.

Rianto, Dicky, Iwan M Erwin, Esa Prakasa, and Herlan Herlan. 2018. "Parking Slot Identification Using Local Binary Pattern and Support Vector Machine." In *2018 International Conference on Computer, Control, Informatics and Its Applications (IC3INA)*, , 129–33.

Ridel, Daniela et al. 2018. "A Literature Review on the Prediction of Pedestrian Behavior in Urban Scenarios." In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, , 3105–12.

Rodriguez-Canosa, Gonzalo R et al. 2012. "A Real-Time Method to Detect and Track Moving Objects (DATMO) from Unmanned Aerial Vehicles (UAVs) Using a Single Camera." *Remote Sensing* 4(4): 1090–1111.

Salvo, Giuseppe, Luigi Caruso, and Alessandro Scordo. 2014. "Urban Traffic Analysis through an UAV." *PROCEDIA: SOCIAL & BEHAVIORAL SCIENCES* 111: 1083–91.

Saran, K B, and G Sreelekha. 2015. "Traffic Video Surveillance: Vehicle Detection and Classification." In *2015 International Conference on Control Communication & Computing India (ICCC)*, , 516–21.

Sargolzaei, Arman, Carl D. Crane, Alireza Abbaspour, and Shirin Noei. 2017. "A Machine Learning Approach for Fault Detection in Vehicular Cyber-Physical Systems." *Proceedings - 2016 15th IEEE International Conference on Machine Learning and Applications, ICMLA*

2016: 636–40.

Sarkar, Sayani, Michael W Totaro, and Khalid Elgazzar. 2019. “Intelligent Drone-Based Surveillance: Application to Parking Lot Monitoring and Detection.” In *Unmanned Systems Technology XXI, ,* 1102104.

Satonaka, Hisashi et al. 2006. “Development of Parking Space Detection Using an Ultrasonic Sensor.” In *PROCEEDINGS OF THE 13th ITS WORLD CONGRESS, LONDON, 8-12 OCTOBER 2006,*.

Saunier, Nicolas, Tarek Sayed, and Karim Ismail. 2010. “Large-Scale Automated Analysis of Vehicle Interactions and Collisions.” *Transportation Research Record* 2147(1): 42–50. https://doi.org/10.3141/2147-06.

Sayed, Tarek, Mohamed H Zaki, and Jarvis Autey. 2013. “Automated Safety Diagnosis of Vehicle--Bicycle Interactions Using Computer Vision Analysis.” *Safety science* 59: 163–72.

Schoepflin, Todd N, and Daniel J Dailey. 2003. “Dynamic Camera Calibration of Roadside Traffic Management Cameras for Vehicle Speed Estimation.” *IEEE Transactions on Intelligent Transportation Systems* 4(2): 90–98.

Sen-Ching, S Cheung, and Chandrika Kamath. 2004. “Robust Techniques for Background Subtraction in Urban Traffic Video.” In *Visual Communications and Image Processing 2004, ,* 881–92.

Sen, Rijurekha, Pankaj Siriah, and Bhaskaran Raman. 2011. “RoadSoundSense: Acoustic Sensing Based Road Congestion Monitoring in Developing Regions.” *2011 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, SECON 2011*: 125–33.

Sharma, Anuj, Darcy M. Bullock, and James A. Bonneson. 2007. “Input-Output and Hybrid Techniques for Real-Time Prediction of Delay and Maximum Queue Length at Signalized Intersections.” *Transportation Research Record* (2035): 69–80.

Shastry, Anand C, and Robert A Schowengerdt. 2005. “Airborne Video Registration and Traffic-Flow Parameter Estimation.” *IEEE Transactions on Intelligent Transportation Systems* 6(4): 391–405.

Shi, Weisong et al. 2016. “Edge Computing: Vision and Challenges.” *IEEE Internet of Things Journal* 3(5): 637–46.

Shibata, Keiji, Kazuya Takeuch, Shohei Kawai, and Yuukou Horita. 2014. “Detection of Road

Surface Conditions in Winter Using Road Surveillance Cameras at Daytime, Night-Time and Twilight." *International Journal of Computer Science and Network Security (IJCSNS)* 14(11): 21.

Sifuentes, E, O Casas, and R Pallas-Areny. 2011. "Wireless Magnetic Sensor Node for Vehicle Detection with Optical Wake-Up." *IEEE Sensors journal* 11(8): 1669–76.

Simonyan, Karen, and Andrew Zisserman. 2014. "Very Deep Convolutional Networks for Large-Scale Image Recognition." *arXiv preprint arXiv:1409.1556*.

Soule, Heidi H et al. 2020. "Testing an Automated Collision Avoidance and Emergency Braking System for Buses." *Transportation research record*: 0361198120912431. https://journals.sagepub.com/doi/full/10.1177/0361198120912431.

Spears, Jerry et al. 2017. *Active Safety-Collision Warning Pilot in Washington State*.

Sturgess, Paul, Karteek Alahari, Lubor Ladicky, and Philip H S Torr. 2009. "Combining Appearance and Structure from Motion Features for Road Scene Understanding."

Sun, Zehang, George Bebis, and Ronald Miller. 2006. "On-Road Vehicle Detection: A Review." *IEEE transactions on pattern analysis and machine intelligence* 28(5): 694–711.

Taguchi, Shun, Satoshi Koide, and Takayoshi Yoshimura. 2019. "Online Map Matching with Route Prediction." *IEEE Transactions on Intelligent Transportation Systems* 20(1): 338–47.

Takeuchi, Kazuya, Shohei Kawai, Keiji Shibata, and Yuukou Horita. 2012. "Distinction of Winter Road Surface Conditions Using Road Surveillance Camera." In *2012 12th International Conference on ITS Telecommunications*, , 663–67.

Talebpour, Alireza, Hani S Mahmassani, Fiorella Mete, and Samer H Hamdar. 2014. "Near-Crash Identification in a Connected Vehicle Environment." *Transportation Research Record* 2424(1): 20–28. https://doi.org/10.3141/2424-03.

Tang, Jinjun et al. 2018. "Lane-Changes Pre Diction Base d on Adaptive Fuzzy Neural Network." *Expert Systems With Applications* 91: 452–63. http://dx.doi.org/10.1016/j.eswa.2017.09.025.

Tettamanti, Tamas, Matyas Szalai, Sandor Vass, and Viktor Tihanyi. 2018. "Vehicle-In-the-Loop Test Environment for Autonomous Driving with Microscopic Traffic Simulation." *2018 IEEE International Conference on Vehicular Electronics and Safety, ICVES 2018*.

Teutsch, Michael, and Wolfgang Krüger. 2012. "Detection, Segmentation, and Tracking of Moving Objects in UAV Videos." In *2012 IEEE Ninth International Conference on*

Advanced Video and Signal-Based Surveillance, , 313–18.

Tian, Wei, Martin Lauer, and Long Chen. 2019. "Online Multi-Object Tracking Using Joint Domain Information in Traffic Scenarios." *IEEE Transactions on Intelligent Transportation Systems* 21(1): 374–84.

Tieleman, Tijmen, and Geoffrey Hinton. 2012. "Lecture 6.5-Rmsprop: Divide the Gradient by a Running Average of Its Recent Magnitude." *COURSERA: Neural networks for machine learning* 4(2): 26–31.

Torres, Renato, Orlando Ohashi, and Gustavo Pessin. 2019. "A Machine-Learning Approach to Distinguish Passengers and Drivers Reading While Driving." *Sensors (Switzerland)* 19(14): 1–29.

Tsai, Luo-Wei, Jun-Wei Hsieh, and Kuo-Chin Fan. 2007. "Vehicle Detection Using Normalized Color and Edge Map." *IEEE transactions on Image Processing* 16(3): 850–64.

Tsao, Paul, Tsi-Ui Ik, Guan-Wen Chen, and Wen-Chih Peng. 2018. "Stitching Aerial Images for Vehicle Positioning and Tracking." In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, , 616–23.

Uchida, Nobuyuki, Takashi Tagawa, and Kenji Sato. 2017. "Development of an Augmented Reality Vehicle for Driver Performance Evaluation." *IEEE Intelligent Transportation Systems Magazine* 9(1): 35–41. https://ieeexplore.ieee.org/abstract/document/7823088.

Vitek, Stanislav, and Petr Melničuk. 2018. "A Distributed Wireless Camera System for the Management of Parking Spaces." *Sensors* 18(1): 69. https://www.mdpi.com/1424-8220/18/1/69.

Wang, Qiang et al. 2019. "Fast Online Object Tracking and Segmentation: A Unifying Approach." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, , 1328–38.

Wang, Xiaojie et al. 2018. "ENABLING TECHNOLOGIES FOR SMART INTERNET OF THINGS A City-Wide Real-Time Traffic Management System : Enabling Crowdsensing in Social Internet of Vehicles." (September): 19–25.

Wang, Yan et al. 2019. "Pseudo-Lidar from Visual Depth Estimation: Bridging the Gap in 3d Object Detection for Autonomous Driving." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, , 8445–53.

Wang, Yinghai, and N. L. Nihan. 2000. "Freeway Traffic Speed Estimation with Single-Loop

Outputs.” *Transportation Research Record* (1727): 120–26.

Wang, Yue, Eam Khwang Teoh, and Dinggang Shen. 2004. “Lane Detection and Tracking Using B-Snake.” *Image and Vision computing* 22(4): 269–80.

Wei, Yichen, Jian Sun, Xiaoou Tang, and Heung-Yeung Shum. 2007. “Interactive Offline Tracking for Color Objects.” In *2007 IEEE 11th International Conference on Computer Vision, ,* 1–8.

Wei, Yun et al. 2019. “Multi-Vehicle Detection Algorithm through Combining Harr and HOG Features.” *Mathematics and Computers in Simulation* 155: 130–45.

Wojke, Nicolai, Alex Bewley, and Dietrich Paulus. 2017. “Simple Online and Realtime Tracking with a Deep Association Metric.” In *2017 IEEE International Conference on Image Processing (ICIP), ,* 3645–49.

Wu, Jianqing, Hao Xu, Yichen Zheng, and Zong Tian. 2018. “A Novel Method of Vehicle-Pedestrian near-Crash Identification with Roadside LiDAR Data.” *Accident Analysis & Prevention* 121: 238–49. https://www.sciencedirect.com/science/article/pii/S0001457518305591.

Wu, Qi et al. 2007. “Robust Parking Space Detection Considering Inter-Space Correlation.” In *2007 IEEE International Conference on Multimedia and Expo, ,* 659–62. https://ieeexplore.ieee.org/document/4284736.

Wu, Yi, Jongwoo Lim, and Ming-Hsuan Yang. 2013. “Online Object Tracking: A Benchmark.” In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, ,* 2411–18.

Xu, Chengcheng, Junyi Ji, and Pan Liu. 2018. “The Station-Free Sharing Bike Demand Forecasting with a Deep Learning Approach and Large-Scale Datasets.” *Transportation Research Part C* 95(July): 47–60. https://doi.org/10.1016/j.trc.2018.07.013.

Xu, Huazhe, Yang Gao, Fisher Yu, and Trevor Darrell. 2017. “End-to-End Learning of Driving Models from Large-Scale Video Datasets.” *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* 2017-Janua: 3530–38.

Xu, Xiaolong et al. 2019. “An Edge Computing-Enabled Computation Offloading Method with Privacy Preservation for Internet of Connected Vehicles.” *Future Generation Computer Systems* 96: 89–100. https://doi.org/10.1016/j.future.2019.01.012.

Xu, Yong, Jixiang Dong, Bob Zhang, and Daoyun Xu. 2016. “Background Modeling Methods in

Video Analysis: A Review and Comparative Evaluation." *CAAI Transactions on Intelligence Technology* 1(1): 43–60.

Xu, Yongzheng et al. 2017. "An Enhanced Viola-Jones Vehicle Detection Method from Unmanned Aerial Vehicles Imagery." *IEEE trans Intell Transp Syst* 18(7): 1845–56.

Xu, Zhigang et al. 2017. "PaTAVTT: A Hardware-in-the-Loop Scaled Platform for Testing Autonomous Vehicle Trajectory Tracking." *Journal of Advanced Transportation* 2017. https://www.hindawi.com/journals/jat/2017/9203251/.

Yalcin, Hulya, Martial Hebert, Robert Collins, and Michael J Black. 2005. "A Flow-Based Approach to Vehicle Detection and Background Mosaicking in Airborne Video." In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05),* , 1202--vol.

Yamazaki, Fumio, Wen Liu, and T Thuy Vu. 2008. "Vehicle Extraction and Speed Detection from Digital Aerial Images." In *IGARSS 2008-2008 IEEE International Geoscience and Remote Sensing Symposium,* , III--1334.

Yang, Shuguan, Wei Ma, Xidong Pi, and Sean Qian. 2019. "A Deep Learning Approach to Real-Time Parking Occupancy Prediction in Transportation Networks Incorporating Multiple Spatio-Temporal Data Sources." *Transportation Research Part C* 107(August): 248–65. https://doi.org/10.1016/j.trc.2019.08.010.

Yao, Huaxiu et al. 2018. "Deep Multi-View Spatial-Temporal Network for Taxi Demand Prediction." *arXiv preprint arXiv:1802.08714*.

Yu, Qian, and Gérard Medioni. 2009. "Motion Pattern Interpretation and Detection for Tracking Moving Vehicles in Airborne Video." In *2009 IEEE Conference on Computer Vision and Pattern Recognition,* , 2671–78.

Yuan, Quan et al. 2018. "Toward Efficient Content Delivery for Automated Driving Services : An Edge Computing Solution." (February): 80–86.

Zhang, Jinlei et al. 2020. "Deep Learning Architecture for Short-Term Passenger Flow Forecasting in Urban Rail Transit." *IEEE Transactions on Intelligent Transportation Systems*.

Zhang, Kunpeng et al. 2020. "A Generative Adversarial Network for Travel Times Imputation Using Trajectory Data." *Computer-Aided Civil and Infrastructure Engineering*: 1–16.

Zhang, Shanshan, Christian Bauckhage, and Armin B Cremers. 2014. "Informed Haar-like

Features Improve Pedestrian Detection." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, , 947–54.

Zhang, Yuxuan et al. 2019. "Trafficgan: Network-Scale Deep Traffic Prediction with Generative Adversarial Nets." *IEEE Transactions on Intelligent Transportation Systems*.

Zhang, Zhaoxiang, Tieniu Tan, Kaiqi Huang, and Yunhong Wang. 2012. "Practical Camera Calibration from Moving Objects for Traffic Scene Surveillance." *IEEE transactions on circuits and systems for video technology* 23(3): 518–33.

Zhang, Zhenhua, Qing He, Jing Gao, and Ming Ni. 2018. "A Deep Learning Approach for Detecting Traffic Accidents from Social Media Data." *Transportation Research Part C: Emerging Technologies* 86(November 2017): 580–96. https://doi.org/10.1016/j.trc.2017.11.027.

Zhang, Zusheng, Xiaoyun Li, Huaqiang Yuan, and Fengqi Yu. 2013. "A Street Parking System Using Wireless Sensor Networks." *International Journal of Distributed Sensor Networks* 9(6): 107975.

Zhang, Zusheng, Ming Tao, and Huaqiang Yuan. 2014. "A Parking Occupancy Detection Algorithm Based on AMR Sensor." *IEEE Sensors Journal* 15(2): 1261–69.

Zhao, Junxuan et al. 2019. "Detection and Tracking of Pedestrians and Vehicles Using Roadside LiDAR Sensors." *Transportation Research Part C: Emerging Technologies* 100(December 2017): 68–87. https://doi.org/10.1016/j.trc.2019.01.007.

Zhao, Wenjuan, Edward McCormack, Daniel J. Dailey, and Eric Scharnhorst. 2013. "Using Truck Probe Gps Data to Identify and Rank Roadway Bottlenecks." *Journal of Transportation Engineering* 139(1): 1–7.

Zhao, Xi, Douglas Dawson, Wayne A Sarasua, and Stanley T Birchfield. 2017. "Automated Traffic Surveillance System with Aerial Camera Arrays Imagery: Macroscopic Data Collection with Vehicle Tracking." *Journal of Computing in Civil Engineering* 31(3): 4016072.

Zhao, Xiangmo et al. 2020. "Field Experiments on Longitudinal Characteristics of Human Driver Behavior Following an Autonomous Vehicle." *Transportation Research Part C: Emerging Technologies* 114(February): 205–24. https://doi.org/10.1016/j.trc.2020.02.018.

Zheng, Zengwei et al. 2019. "Enabling Real-Time Road Anomaly Detection via Mobile Edge Computing." *International Journal of Distributed Sensor Networks* 15(11):

1550147719891319.

Zhou, Hailing et al. 2014. “Efficient Road Detection and Tracking for Unmanned Aerial Vehicle.” *IEEE transactions on intelligent transportation systems* 16(1): 297–309.

Zhou, Junhao, Hong-Ning Dai, and Hao Wang. 2019. “Lightweight Convolution Neural Networks for Mobile Edge Computing in Transportation Cyber Physical Systems.” *ACM Transactions on Intelligent Systems and Technology (TIST)* 10(6): 1–20.

Zhou, Zhi et al. 2019. “Edge Intelligence: Paving the Last Mile of Artificial Intelligence with Edge Computing.” *arXiv* 107(8).

Zhu, Hongmei, and Fengqi Yu. 2015. “A Vehicle Parking Detection Method Based on Correlation of Magnetic Signals.” *International Journal of Distributed Sensor Networks* 11(7): 361242. https://journals.sagepub.com/doi/full/10.1155/2015/361242.

Zhu, Jiasong et al. 2018. “Urban Traffic Density Estimation Based on Ultrahigh-Resolution UAV Video and Deep Neural Network.” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 11(12): 4968–81.

Zhuang, Yifan, Ruimin Ke, and Yinhai Wang. 2020. “Edge-Based Traffic Flow Data Collection Method Using Onboard Monocular Camera.” *Journal of Transportation Engineering Part A: Systems* 146(9).

Zhuang, Yifan, Ruimin Ke, and Yinhai Wang. 2018. “Innovative Method for Traffic Data Imputation Based on Convolutional Neural Network.” *IET Intelligent Transport Systems* 13(4): 605-613.